Remote ATtestation ProcedureS
Internet-Draft
Intended status: Standards Track
Expires: 12 January 2023

H. Birkholz
Fraunhofer SIT
T. Fossati
Y. Deshpande
arm
N. Smith
Intel
W. Pan
Huawei Technologies
11 July 2022

Concise Reference Integrity Manifest
draft-birkholz-rats-corim-03

Abstract

Remote Attestation Procedures (RATS) enable Relying Parties to assess the trustworthiness of a remote Attester and therefore to decide whether to engage in secure interactions with it. Evidence about trustworthiness can be rather complex and it is deemed unrealistic that every Relying Party is capable of the appraisal of Evidence. Therefore that burden is typically offloaded to a Verifier. In order to conduct Evidence appraisal, a Verifier requires not only fresh Evidence from an Attester, but also trusted Endorsements and Reference Values from Endorsers and Reference Value Providers, such as manufacturers, distributors, or device owners. This document specifies Concise Reference Integrity Manifests (CoRIM) that represent Endorsements and Reference Values in CBOR format. Composite devices or systems are represented by a collection of Concise Module Identifiers (CoMID) and Concise Software Identifiers (CoSWID) bundled in a CoRIM document.

Discussion Venues

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

Discussion of this document takes place on the RATS Working Group mailing list (rats@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/rats/.

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 12 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 4
  1.1. Terminology and Requirements Language 4
  1.2. CDDL Typographical Conventions 4
  1.3. Common Types 5
    1.3.1. Non-Empty 5
    1.3.2. Entity 5
    1.3.3. Validity 6
    1.3.4. UUID 7
    1.3.5. UEID 7
    1.3.6. OID 7
    1.3.7. Tagged Integer Type 7
    1.3.8. Hash Entry 7
2. CoRIM 8
  2.1. CoRIM Map 8
    2.1.1. Identity 9
    2.1.2. Tags 9
    2.1.3. Locator Map 9
    2.1.4. Profile Types 10
    2.1.5. Entities 10
  2.2. Signed CoRIM 10

2.2.1. Protected Header Map 11
2.2.2. Meta Map 11
2.2.2.1. Signer Map 12
3. CoMID 12
3.1. Structure 12
3.1.1. Tag Identity 13
3.1.1.1. Tag ID 13
3.1.1.2. Tag Version 14
3.1.2. Entities 14
3.1.3. Linked Tag 15
3.1.4. Triples 15
3.1.4.1. Common Types 16
3.1.4.1.1. Environment 16
3.1.4.1.2. Class 17
3.1.4.1.3. Instance 18
3.1.4.1.4. Group 18
3.1.4.1.5. Measurements 18
3.1.4.1.5.1. Measurement Keys 19
3.1.4.1.5.2. Measurement Values 19
3.1.4.1.5.3. Version 20
3.1.4.1.5.4. Security Version Number 21
3.1.4.1.5.5. Flags 21
3.1.4.1.5.6. Raw Values Types 22
3.1.4.1.5.7. Address Types 22
3.1.4.1.6. Crypto Keys 22
3.1.4.1.7. Domain Types 23
3.1.4.2. Reference Values Triple 23
3.1.4.3. Endorsed Values Triple 24
3.1.4.4. Device Identity Triple 24
3.1.4.5. Attestation Keys Triple 24
3.1.4.6. Domain Dependency Triple 24
3.1.4.7. Domain Membership Triple 25
3.1.4.8. CoMID-CoSWID Linking Triple 25
3.2. Extensibility 25
4. Implementation Status 26
4.1. Veraison 26
5. Security and Privacy Considerations 27
6. IANA Considerations 27
6.1. New COSE Header Parameters 27
6.2. New CBOR Tags 27
6.3. New CoRIM Registries 27
6.4. New CoMID Registries 27
6.5. New Media Types 27
6.5.1. corim-signed+cbor 28
6.5.2. corim-unsigned+cbor 28
6.6. CoAP Content-Formats Registration 29
7. References 29
7.1. Normative References 29

7.2. Informative References 31
Appendix A. Full CoRIM CDDL 32
Acknowledgments 32
Authors' Addresses 32

1. Introduction

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/86

1.1. Terminology and Requirements Language

This document uses terms and concepts defined by the RATS architecture. For a complete glossary see Section 4 of [I-D.ietf-rats-architecture].

The terminology from CBOR [STD94], CDDL [RFC8610] and COSE [RFC8152] applies; in particular, CBOR diagnostic notation is defined in Section 8 of [STD94] and Appendix G of [RFC8610].

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

1.2. CDDL Typographical Conventions

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

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

Table 1: Type Traits & Typographical Conventions

1.3. Common Types

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

1.3.1. Non-Empty

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

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

1.3.2. Entity

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

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

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

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

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

The following describes each member of the entity-map.

* entity-name (index 0): The name of entity which is responsible for the action(s) as defined by the role. $entity-name-type-choice can only be Other specifications can extend the $entity-name-type-choice (see Section 6.4).

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

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

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

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

1.3.3. Validity

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

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

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

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

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

1.3.4. UUID

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

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

1.3.5. UEID

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

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

1.3.6. OID

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

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

1.3.7. Tagged Integer Type

```
// Content missing.  Tracked at: https://github.com/ietf-rats/draft-
birkholz-rats-corim/issues/87

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

1.3.8. Hash Entry

A hash entry represents the value of a hashing operation together with the hash algorithm used. Defined in Section 2.9.1 of [I-D.ietf-sacm-coswid]. The CDDL is copied below for convenience.

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

2. CoRIM

```
// Content missing.  Tracked at: https://github.com/ietf-rats/draft-
birkholz-rats-corim/issues/98
```

At the top-level, a CoRIM can either be a CBOR-tagged corim-map (Section 2.1) or a COSE signed corim-map (Section 2.2).

```
corim = #6.500(concise-rim-type-choice)

$concise-rim-type-choice /= #6.501(corim-map)
$concise-rim-type-choice /= #6.502(signed-corim)
```

2.1. CoRIM Map

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

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

The following describes each child item of this map.

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

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

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

* profile (index 3): One or more unique identifiers for the profiles of the tags contained in this CoRIM. All the listed profiles MUST be understood. Failure to recognize a profile identifier MUST result in the rejection of the entire processing. Described in Section 2.1.4

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

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

* $$corim-map-extension: This CDDL socket is used to add new information structures to the corim-map. See Section 6.3.

2.1.1. Identity

A CoRIM id can be either a text string or a UUID type that uniquely identifies a CoRIM.

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

2.1.2. Tags

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

```
$concise-tag-type-choice /= #6.505(bytes .cbor concise-swid-tag)
$concise-tag-type-choice /= #6.506(bytes .cbor concise-mid-tag)
```

2.1.3. Locator Map

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

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

The following describes each child element of this type.

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

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

2.1.4. Profile Types

A profile specifies which of the optional parts of a CoRIM are required, which are prohibited and which extension points are exercised and how.

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

2.1.5. Entities

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

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

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

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

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

2.2. Signed CoRIM

Signing a CoRIM follows the procedures defined in CBOR Object Signing and Encryption [RFC8152]. A CoRIM tag MUST be wrapped in a COSE_Sign1 structure. The CoRIM MUST be signed by the CoRIM creator.

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

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

The following describes each child element of this type.

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

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

* payload: A CBOR encoded tagged CoRIM.

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

2.2.1. Protected Header Map

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

The following describes each child item of this map.

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

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

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

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

Additional data can be included in the COSE header map as per Section 3 of [RFC8152].

2.2.2. Meta Map

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

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

The following describes each child item of this group.

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

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

2.2.2.1. Signer Map

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

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

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

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

3. CoMID

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/88

3.1. Structure

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

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

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

* lang (index 0): A textual language tag that conforms with IANA "Language Subtag Registry" [IANA.language-subtag-registry]. The context of the specified language applies to all sibling and

descendant textual values, unless a descendant object has defined a different language tag. Thus, a new context is established when a descendant object redefines a new language tag. All textual values within a given context MUST be considered expressed in the specified language.

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

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

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

* triples (index 4): One or more triples providing information specific to the described module, e.g.: reference or endorsed values, cryptographic material, or structural relationship between the described module and other modules. Described in (Section 3.1.4).

3.1.1. Tag Identity

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

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

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

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

3.1.1.1. Tag ID

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

A Tag ID is either a 16-byte binary string, or a textual identifier, uniquely referencing the CoMID. The tag identifier MUST be globally unique. Failure to ensure global uniqueness can create ambiguity in tag use since the tag-id serves as the global key for matching,

lookups and linking. If represented as a 16-byte binary string, the identifier MUST be a valid universally unique identifier as defined by [RFC4122]. There are no strict guidelines on how the identifier is structured, but examples include a 16-byte GUID (e.g., class 4 UUID) [RFC4122], or a URI [STD66].

3.1.1.2. Tag Version

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

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

3.1.2. Entities

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

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

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

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

The roles defined for a CoMID entity are:

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

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

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

3.1.3. Linked Tag

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

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

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

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

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

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

The relations defined in this specification are:

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

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

3.1.4. Triples

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

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

The following describes each member of the triples-map:

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

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

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

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

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

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

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

3.1.4.1. Common Types

3.1.4.1.1. Environment

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

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

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

The following describes each member of the environment-map:

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

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

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

3.1.4.1.2. Class

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

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

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

The following describes each member of the class-map:

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

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

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

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

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

3.1.4.1.3. Instance

An instance carries a unique identifier that is reliably bound to an instance of the attester.

The types defined for an instance identifier are UEID or UUID.

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

3.1.4.1.4. Group

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

The type defined for a group identified is UUID.

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

3.1.4.1.5. Measurements

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

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

Class measurements apply generally to all the attesters in the given class. Instance measurements apply to a specific attester instances. 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.

```
measurement-map = {
  ? &(mkey: 0) => $measured-element-type-choice
  &(mval: 1) => measurement-values-map
}
```

The following describes each member of the measurement-map:

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

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

3.1.4.1.5.1. Measurement Keys

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

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

3.1.4.1.5.2. Measurement Values

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.1.4.1.5.3. Version

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

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

The following describes each member of the version-map:

* version (index 0): the version string

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

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

3.1.4.1.5.4. Security Version Number

```
// Content missing. Tracked at: https://github.com/ietf-rats/draft-
birkholz-rats-corim/issues/89

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

3.1.4.1.5.5. Flags

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

```
flags-map = {
  ? &(configured: 0) => bool
  ? &(secure: 1) => bool
  ? &(recovery: 2) => bool
  ? &(debug: 3) => bool
  ? &(replay-protected: 4) => bool
  ? &(integrity-protected: 5) => bool
  * $$flags-map-extension
}
```

The following describes each member of the flags-map:

* configured (index 0): The measured environment is fully configured for normal operation if the flag is true.

* secure (index 1): The measured environment's configurable security settings are fully enabled if the flag is true.

* recovery (index 2): The measured environment is NOT in a recovery state if the flag is true.

* debug (index 3): The measured environment is in a debug enabled state if the flag is true.

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

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

3.1.4.1.5.6. Raw Values Types

```
// Content missing.  Tracked at: https://github.com/ietf-rats/draft-
birkholz-rats-corim/issues/90

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

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

3.1.4.1.5.7. Address Types

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

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

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

3.1.4.1.6. Crypto Keys

A cryptographic key can be one of the following formats:

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

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

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

```
$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

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)
```

3.1.4.1.7. Domain Types

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

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

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

3.1.4.2. Reference Values Triple

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

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

3.1.4.3. Endorsed Values Triple

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

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

3.1.4.4. Device Identity Triple

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

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

3.1.4.5. Attestation Keys Triple

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

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

3.1.4.6. Domain Dependency Triple

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

trustworthiness properties of the subject domain exists.

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

3.1.4.7. Domain Membership Triple

A Domain Membership triple assigns domain membership to environments. The subject identifies a domain (Section 3.1.4.1.7) that has a predicate relationship to the object containing one or more environments. Endorsed environments (Section 3.1.4.3) membership is conditional upon successful matching of Reference Values (Section 3.1.4.2) to Evidence.

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

3.1.4.8. CoMID-CoSWID Linking Triple

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

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

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

3.2. Extensibility

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/91

4. 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 catalog 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".

4.1. 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/README.md)

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

* Implementation's level of maturity: alpha.

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

* Version compatibility: Version -02 of the draft

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

5. Security and Privacy Considerations

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/92

6. IANA Considerations

6.1. New COSE Header Parameters

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/96

6.2. New CBOR Tags

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/93

6.3. New CoRIM Registries

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/94

6.4. New CoMID Registries

// Content missing. Tracked at: https://github.com/ietf-rats/draft-birkholz-rats-corim/issues/95

6.5. New Media Types

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

Name	Template	Reference
corim-signed+cbor	application/corim-signed+cbor	RFCthis, Section 6.5.1
corim-unsigned+cbor	application/corim-unsigned+cbor	RFCthis, Section 6.5.2

Table 2: New Media Types

6.5.1. corim-signed+cbor

Type name: application
Subtype name: corim-signed+cbor
Required parameters: n/a
Optional parameters: "profile" (CoRIM profile in string format. OIDs MUST use the dotted-decimal notation.)
Encoding considerations: binary
Security considerations: Section 5 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Attestation Verifiers, Endorsers and Reference-Value providers that need to transfer COSE Sign1 wrapped CoRIM payloads over HTTP(S), CoAP(S), and other transports.
Fragment identifier considerations: n/a
Magic number(s): D9 01 F6 D2, D9 01 F4 D9 01 F6 D2
File extension(s): n/a
Macintosh file type code(s): n/a
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration? Maybe

6.5.2. corim-unsigned+cbor

Type name: application
Subtype name: corim-unsigned+cbor
Required parameters: n/a
Optional parameters: "profile" (CoRIM profile in string format. OIDs MUST use the dotted-decimal notation.)
Encoding considerations: binary
Security considerations: Section 5 of RFCthis
Interoperability considerations: n/a

Published specification: RFCthis
Applications that use this media type: Attestation Verifiers, Endorsers and Reference-Value providers that need to transfer unprotected CoRIM payloads over HTTP(S), CoAP(S), and other transports.
Fragment identifier considerations: n/a
Magic number(s): D9 01 F5, D9 01 F4 D9 01 F5
File extension(s): n/a
Macintosh file type code(s): n/a
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration? Maybe

6.6. CoAP Content-Formats Registration

IANA is requested to register the two following Content-Format numbers in the "CoAP Content-Formats" sub-registry, within the "Constrained RESTful Environments (CoRE) Parameters" Registry [IANA.core-parameters]:

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

Table 3: New Content-Formats

7. References

7.1. Normative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-18, 14 June 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-18>.

[I-D.ietf-rats-eat]
Lundblade, L., Mandyam, G., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-14, 10 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-14>.

[I-D.ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-21, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-sacm-coswid-21>.

[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>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/rfc/rfc4122>.

[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>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/rfc/rfc8152>.

[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>.

[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>.

[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>.

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

7.2. Informative References

[I-D.fdb-rats-psa-endorsements] Fossati, T., Deshpande, Y., and H. Birkholz, "Arm's Platform Security Architecture (PSA) Attestation Verifier Endorsements", Work in Progress, Internet-Draft, draft-fdb-rats-psa-endorsements-01, 11 May 2022, <https://datatracker.ietf.org/doc/html/draft-fdb-rats-psa-endorsements-01>.

[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>.

Appendix A. Full CoRIM CDDL

```
// Content missing.  Tracked at: https://github.com/ietf-rats/draft-
birkholz-rats-corim/issues/80

corim = []
```

Acknowledgments

Carl Wallace for review and comments on this document.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@sit.fraunhofer.de

Thomas Fossati
arm
Email: Thomas.Fossati@arm.com

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

Ned Smith
Intel
Email: ned.smith@intel.com

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

Remote ATtestation ProcedureS
Internet-Draft
Intended status: Standards Track
Expires: 11 December 2023

S. Frost
Arm

9 June 2023

Entity Attestation Token (EAT) Collection Type
draft-frost-rats-eat-collection-03

Abstract

The default top-level definitions for an EAT [I-D.ietf-rats-eat] assume a hierarchy involving a leading signer within the Attester. Some token use cases do not match that model. This specification defines an extension to EAT allowing the top-level of the token to consist of a collection of otherwise defined tokens.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-frost-rats-eat-collection/.

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 11 December 2023.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 2
2. Terminology and Requirements Language 3
3. Design Considerations / Use Cases 3
4. Token Collection 4
4.1. Binder Definition 5
5. Security Considerations 7
6. IANA Considerations 7
6.1. Media Type Registration 8
7. References 9
7.1. Normative References 9
7.2. Informative References 10
Appendix A. CDDL 11
Acknowledgments 12
Contributors 12
Author's Address 12

1. Introduction

An Attestation Token conforming to EAT [I-D.ietf-rats-eat] has a default top level definition for a token to be constructed principally as a claim set within a CBOR Web Token (CWT) [RFC8392] with the associated COSE envelope [STD96] providing at least integrity and authentication. An equivalent JSON encoding for a JWT [RFC7519] in a JWS envelope [RFC7515] is supported as an alternative at the top-level definition. The top level token can be augmented with related claims in a Detached EAT Bundle.

For the use case of transmitting a claim set through a secure channel, the top-level definition can be extended to use an Unprotected CWT Claim Set (UCCS) [I-D.ietf-rats-uccs].

This document outlines an additional top-level extension for which neither of the above top level definitions match exactly: the attestation token consists of a collection of objects, each with their own integrity and some internally defined relationship through which the integrity of the whole collection can be determined. i.e. there is no top-level signer for the set. The objects may all share the same logical hierarchy in a device or have a hierarchy which is internally defined within the object set.

2. Terminology and Requirements Language

Readers are also expected to be familiar with the terminology from [I-D.ietf-rats-eat] and [RFC9334].

In this document, the structure of data is specified in CDDL [RFC8610] [RFC9165].

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.

3. Design Considerations / Use Cases

Take a device with an attestation system consisting of a platform claim set and a workload claim set, each controlled by different components and with an underlying hardware Root of Trust. The two claim sets are delivered together to make up the overall attestation token. Depending upon the implementation and deployment use case, the signing system can either be entirely centric to the platform RoT or can have separate signers for the two claim sets. In either case, a cryptographic binding is established between the two parts of the token.

A specific manifestation of such a device is one incorporating the Arm Confidential Compute Architecture (CCA) attestation token [Arm-CCA]. In trying to prepare the attestation token using EAT, there were no issues constructing the claim sets or incorporating them into individual CWTs where appropriate. However, in trying to design an 'envelope structure' to convey the two parts as a single report it was found that maintaining EAT compatibility would require very different shaped compound tokens for different models, for example one based on a submod arrangement and another based on a Detached EAT Bundle, though with different leading objects. This would create extra code and explanation in areas where keeping things simple is desirable. There was an alternative approach considered, which stays close to existing thinking on EAT, which would be to

create the wrapper from the UCCS EAT extension containing only submods for the respective components. This however stretches the current use case for UCCS beyond its existing description. The RATS WG approach of separating UCCS from the core EAT specification to be an extension also encourages proposing this further extension.

To support the CCA use case, it is also relevant to consider current attestation technologies which are based on certificate chains (e.g. SPDM, DICE, several key attestation systems). Here also are multiple objects with their own integrity and an internally defined relationship. If attempting to move such a technology to the EAT world, the same challenges apply.

An additional use case beyond the production of tokens from an Attester occurs when using EAT to convey Attestation Results from a Verifier. Attestation results may be separated into different sections depending upon what aspects of Appraisal Policy are applied by the Verifier. For example, the set of validated evidence claims may form one section, while claims reflecting semantic conclusions drawn by an Appraisal Policy could form another section. Given the role of different authorities in concluding result sections, each could have a different signer rather than all results being under a single signature from the Verifier. In this case, a collection can be used to collate all result sections into a single response message. Using a collection simplifies operations if individual sections from the collated result sections need to be later dispersed to different Relying Parties.

A further Attestation Result use case can be seen in the "Below Zero Trust" system described in [I-D.ietf-rats-ar4si] where the AR-augmented Evidence credential has compound form.

4. Token Collection

The proposed extension for the top-level definition is to add a 'Token Collection' type. The contents of the type are a map of CWTs (JWTs). The Detached EAT Bundle top-level entry for EAT is included for completeness, and the UCCS extension can also be embraced, though the use cases for these have not been explored. The identification of collection members and the intra collection integrity mechanism is considered usage specific. A verifier will be expected to extract each of the members of the collection and check their validity both individually and as a set. In addition to entries which have their own integrity, it is also supported to include an unsigned Claims Set, provided that the integrity for that Claims Set is provided within another entry that uses one of the signed forms.

A map was chosen rather than an unbounded array to give the opportunity to add identifying map tags to each entry. The interpretation of the tags will be usage specific, but may correspond to registered identities of specific token types. To assist a verifier correlate the expected contents a profile entry can be added as the profile-label identity in the map.

See Appendix A for a CDDL description of the proposed extension.

While most of the use cases for collections are for scenarios where there will be at least two entries in a collection, the CDDL allows for >= 1 entries in a collection to allow for the scenario where only one entry is currently available even though the normal set is larger.

4.1. Binder Definition

This specification includes a proposal for a Collection Binder claim (see Figure 1). This claim would be included within any collection entry as a definiton of the integrity mechanism that binds that collection entry to another collection entry. There may be more than one binder claims in a given collection entry. If a binder is present, a verifier MUST use the information within this claim to drive inter-collection entry integrity checking. The binders in a collection describe a digraph structure. The verifier MUST reject a collection that contains loops, i.e., it must make sure that the described structure is a simple directed graph. This claim would not be mandatory within a collection entry as a verifier may implement the integrity checking based upon information in the profile alone.

```
; The Collection Binder is a formal declaration of the inter entry
; binding mechanism. It would be included within the body of one or
; more of the collection entries.
Collection-Binder = [
    binder-function,
    [*binder-source-label],
    destination-collection-entry,
    destination-claim
]

; binder-function is the name/id of a hash algorithm
binder-function = JC<text,int>

; By definition, the binder-function is applied to a concatenation
; of the ordered list of source claims.
; If the array is empty, the function is applied to the whole
; contents of the token.
binder-source-label = Claim-Label

destination-collection-entry = collection-entry-label
destination-claim = Claim-Label

Claim-Label = JC<"text", int>
collection-entry-label = JC<text, int>
JC<J,C> = J .feature "json" / C .feature "cbor"
```

Figure 1: EAT collection binder

The attributes within the binder claim are:

* binder-function: the identity of the binding cryptographic function, usually a hash function, applied to the values identified by the binder-source-label array.

* binder-source-label: an array defining the set of claims providing the binding information within the collection entry. It is assumed that the values corresponding to this (ordered) list will be concatenated and have the binder-function applied to produce a binder value. If the array is empty, the entire source collection entry is used as input to the binder-function. This latter condition would normally be applied to a collection entry consisting of a Claim Set.

* destination-collection-entry: this defines the collection entry that will hold (receive) the binding for this (source) collection entry.

* destination-claim: this defines the claim label within destination-collection-entry which will store the binder value.

A verifier can check the binding between two collection entries by computing the binder value for one entry and comparing the result stored within the value of the destination claim (in the destination collection entry).

5. Security Considerations

A verifier for an attestation token must apply a verification process for the full set of entries contained within the Token Collection. This process will be custom to the relevant profile for the Token Collection and take into account any individual verification per entry and/or verification for the objects considered collectively, including the intra token integrity scheme. As there is no overall signature for the Collection, protection against malicious modification MUST be contained within the entries. In general, a secure channel for the conveyance of EAT collections MUST provide at least end-to-end cryptographic integrity. It is expected that there exists a cryptographic binding between entries, this can for example be one to many or one to one in a (chain) series. The implementation of creating these bindings may involve passing data across ABIs. This provides an attack vector on the integrity of the collection which must be considered within any threat model. With respect to binder claims, these require integrity protection. This protection can either be provided by the signature on the token entry which contains the binder or, in the case where the entry does not have a signature, by including the binder claim with any other claims when preparing input into the cryptographic binding function. Depending upon the use case and associated threat model, the freshness of entries may need extra consideration.

6. IANA Considerations

In the registry [IANA.cbor-tags], IANA is requested to allocate the tag in Table 1 from the FCFS space, with the present document as the specification reference.

Tag	Data Item	Semantics
TBD399	map	EAT Collection RFCthis

Table 1: EAT Collection

6.1. Media Type Registration

IANA is requested to register the "application/eat-collection" media type [RFC2046] in the "Media Types" registry [IANA.media-types] in the manner described in [RFC6838], which can be used to indicate that the content is an EAT collection.

The new media type extends the set of EAT media types introduced in [I-D.ietf-rats-eat-media-type].

* Type name: application

* Subtype name: eat-collection

* Required parameters: n/a

* Optional parameters: n/a

* Encoding considerations: binary

* Security considerations: See the Security Considerations section of RFCthis

* Interoperability considerations: n/a

* Published specification: RFCthis

* Applications that use this media type: Attesters, Relying Parties and Verifiers sending EAT collection messages over HTTP(S) and other transports.

* Fragment identifier considerations: n/a

* Additional information:

  - Magic number(s): n/a

  - File extension(s): n/a

  - Macintosh file type code(s): n/a

* Person & email address to contact for further information: Simon Frost, Simon.Frost@arm.com

* Intended usage: COMMON

* Restrictions on usage: none

* Author: Simon Frost, Simon.Frost@arm.com

* Change controller: IESG

* Provisional registration? No

7. References

7.1. Normative References

[I-D.ietf-rats-eat]
Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-19, 19 December 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-19>.

[I-D.ietf-rats-eat-media-type]
Lundblade, L., Birkholz, H., and T. Fossati, "EAT Media Types", Work in Progress, Internet-Draft, draft-ietf-rats-eat-media-type-02, 10 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-media-type-02>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/rfc/rfc2046>.

[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>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[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>.

[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>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/rfc/rfc9165>.

7.2. Informative References

[Arm-CCA] Arm Ltd, "Confidential Compute Architecture", 2022, <https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture>.

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

[I-D.ietf-rats-uccs] Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-ietf-rats-uccs-05, 1 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-uccs-05>.

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

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

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[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>.

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

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

[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>.

Appendix A. CDDL

```
$$EAT-CBOR-Tagged-Token /= Tagged-Collection
$$EAT-CBOR-Untagged-Token /= TL-Collection

Tagged-Collection =  #6.TBD399(TL-Collection)

TL-Collection = {
    ? eat-collection-identifier,
    + all-collection-types
}

eat-collection-identifier = (
    profile-label => general-uri / general-oid
)

all-collection-types = (
    cwt-collection-entries //
    jwt-collection-entries //
    claim-set-collection-entries //
    detatched-eat-bundle-collection-entries
)

cwt-collection-entries = (
    collection-entry-label => CWT-Messages
)

jwt-collection-entries = (
    collection-entry-label => JWT-Messages
)

claim-set-collection-entries = (
    collection-entry-label => JC<json-wrapped-claims-set,
                                 cbor-wrapped-claims-set>
)

detatched-eat-bundle-collection-entries = (
    collection-entry-label => BUNDLE-Messages
```

```
)

collection-entry-label = JC<text, int>

; The Collection Binder is a formal declaration of the inter entry
; binding mechanism. It would be included within the body of one or
; more of the collection entries.
Tagged-Collection-Binder =  #6.TBD99(Collection-Binder)
Collection-Binder = [
    binder-function,
    [*binder-source-label],
    destination-collection-entry,
    destination-claim
]
; binder function is normally the name/id of a hash algorithm
binder-function = JC<text,int>

; by definition, the binder-function is applied to a concatenation
; of the ordered list of source claims
; If the array is empty, the function is applied to the whole
; contents of the token
binder-source-label = Claim-Label

destination-collection-entry = collection-entry-label

destination-claim = Claim-Label
```

Acknowledgments

Thomas Fossati proposed the inclusion of the Binder definiton and collaborated on the CDDL. Yogesh Deshpande provided insightful comments and review for this proposal.

Contributors

Thomas Fossati
Arm Limited
Email: thomas.fossati@arm.com

Author’s Address

Simon Frost
Arm
Email: Simon.Frost@arm.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2024

E. Voit
Cisco
H. Birkholz
Fraunhofer SIT
T. Hardjono
MIT
T. Fossati
Linaro
V. Scarlata
Intel
1 March 2024

Attestation Results for Secure Interactions
draft-ietf-rats-ar4si-06

Abstract

This document defines reusable Attestation Result information elements. When these elements are offered to Relying Parties as Evidence, different aspects of Attester trustworthiness can be evaluated. Additionally, where the Relying Party is interfacing with a heterogeneous mix of Attesting Environment and Verifier types, consistent policies can be applied to subsequent information exchange between each Attester and the Relying Party.

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-ar4si.

Status of This Memo

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

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

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

This Internet-Draft will expire on 2 September 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 3
  1.1. Requirements Notation 4
  1.2. Terminology 4
2. Attestation Results for Secure Interactions 6
  2.1. Information driving a Relying Party Action 6
  2.2. Non-repudiable Identity 6
    2.2.1. Attester and Attesting Environment 8
    2.2.2. Verifier 10
    2.2.3. Communicating Identity 11
  2.3. Trustworthiness Claims 11
    2.3.1. Design Principles 11
    2.3.2. Enumeration Encoding 13
    2.3.3. Assigning a Trustworthiness Claim value 14
    2.3.4. Specific Claims 14
    2.3.5. Trustworthiness Vector 19
    2.3.6. Trustworthiness Vector for a type of Attesting Environment 19
  2.4. Freshness 20
3. Secure Interactions Models 20
  3.1. Background-Check 20
    3.1.1. Verifier Retrieval 21
    3.1.2. Co-resident Verifier 21
  3.2. Below Zero Trust 22
  3.3. Mutual Attestation 26
  3.4. Transport Protocol Integration 26
4. Privacy Considerations 27
5. Security Considerations 27
6. IANA Considerations 27
7. References 27
  7.1. Normative References 27

7.2. Informative References 27
Appendix A. Implementation Guidance 29
A.1. Supplementing Trustworthiness Claims 29
Appendix B. Supportable Trustworthiness Claims 29
B.1. Supportable Trustworthiness Claims for HSM-based CC . . . 29
B.2. Supportable Trustworthiness Claims for process-based CC 31
B.3. Supportable Trustworthiness Claims for VM-based CC . . . 33
Appendix C. Some issues being worked 34
Appendix D. Contributors 34
Authors' Addresses 35

1. Introduction

The first paragraph of the May 2021 US Presidential Executive Order on Improving the Nation's Cybersecurity [US-Executive-Order] ends with the statement "the trust we place in our digital infrastructure should be proportional to how trustworthy and transparent that infrastructure is." Later this order explores aspects of trustworthiness such as an auditable trust relationship, which it defines as an "agreed-upon relationship between two or more system elements that is governed by criteria for secure interaction, behavior, and outcomes."

The Remote ATtestation procedureS (RATS) architecture [RFC9334] provides a useful context for programmatically establishing and maintaining such auditable trust relationships. Specifically, the architecture defines conceptual messages conveyed between architectural subsystems to support trustworthiness appraisal. The RATS conceptual message used to convey evidence of trustworthiness is the Attestation Results. The Attestation Results includes Verifier generated appraisals of an Attester including such information as the identity of the Attester, the security mechanisms employed on this Attester, and the Attester's current state of trustworthiness.

Generated Attestation Results are ultimately conveyed to one or more Relying Parties. Reception of an Attestation Result enables a Relying Party to determine what action to take with regards to an Attester. Frequently, this action will be to choose whether to allow the Attester to securely interact with the Relying Party over some connection between the two.

When determining whether to allow secure interactions with an Attester, a Relying Party is challenged with a number of difficult problems which it must be able to handle successfully. These problems include:

* What Attestation Results (AR) might a Relying Party be willing to trust from a specific Verifier?

* What information does a Relying Party need before allowing interactions or choosing policies to apply to a connection?

* What are the operating/environmental realities of the Attesting Environment where a Relying Party should only be able to associate a certain confidence regarding Attestation Results out of the Verifier? (In other words, different types of Trusted Execution Environments (TEE) need not be treated as equivalent.)

* How to make direct comparisons where there is a heterogeneous mix of Attesting Environments and Verifier types.

To address these problems, it is important that specific Attestation Result information elements are framed independently of Attesting Environment specific constraints. If they are not, a Relying Party would be forced to adapt to the syntax and semantics of many vendor specific environments. This is not a reasonable ask as there can be many types of Attesters interacting with or connecting to a Relying Party.

The business need therefore is for common Attestation Result information element definitions. With these definitions, consistent interaction or connectivity decisions can be made by a Relying Party where there is a heterogenous mix of Attesting Environment types and Verifier types.

This document defines information elements for Attestation Results in a way which normalizes the trustworthiness assertions that can be made from a diverse set of Attesters.

1.1. Requirements Notation

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

1.2. Terminology

The following terms are imported from [RFC9334]: Appraisal Policy for Attestation Results, Attester, Attesting Environment, Claims, Evidence, Relying Party, Target Environment and Verifier.

[RFC9334] also describes topological patterns that illustrate the need for interoperable conceptual messages. The two patterns called "background-check model" and "passport model" are imported from the RATS architecture and used in this document as a reference to the architectural concepts: Background-Check Model and Passport Model.

Newly defined terms for this document:

AR-augmented Evidence: a bundle of Evidence which includes at least the following:

1. Verifier signed Attestation Results. These Attestation Results must include Identity Evidence for the Attester, a Trustworthiness Vector describing a Verifier's most recent appraisal of an Attester, and some Verifier Proof-of-Freshness (PoF).

2. A Relying Party PoF which is bound to the Attestation Results of (1) by the Attester's Attesting Environment signature.

3. Sufficient information to determine the elapsed interval between the Verifier PoF and Relying Party PoF.

Identity Evidence: Evidence which unambiguously identifies an identity. Identity Evidence could take different forms, such as a certificate, or a signature which can be appraised to have only been generated by a specific private/public key pair.

Trustworthiness Claim: a specific quanta of trustworthiness which can be assigned by a Verifier based on its appraisal policy.

Trustworthiness Tier: a categorization of the levels of trustworthiness which may be assigned by a Verifier to a specific Trustworthiness Claim. These enumerated categories are: Affirming, Warning, Contraindicated, and None.

Trustworthiness Vector: a set of zero to many Trustworthiness Claims assigned during a single appraisal procedure by a Verifier using Evidence generated by an Attester. The vector is included within Attestation Results.

2. Attestation Results for Secure Interactions

A Verifier generates the Attestation Results used by a Relying Party. When a Relying Party needs to determine whether to permit communications with an Attester, these Attestation Results must contain a specific set of information elements. This section defines those information elements, and in some cases encodings for information elements.

2.1. Information driving a Relying Party Action

When the action is a communication establishment attempt with an Attester, there is only a limited set of actions which a Relying Party might take. These actions include:

* Allow or deny information exchange with the Attester. When there is a deny, reasons should be returned to the Attester.

* Establish a transport connection between an Attester and a specific context within a Relying Party (e.g., a TEE, or Virtual Routing Function (VRF).)

* Apply policies on this connection (e.g., rate limits).

There are three categories of information which must be conveyed to the Relying Party (which also is integrated with a Verifier) before it determines which of these actions to take.

1. Non-repudiable Identity Evidence Evidence which undoubtably identifies one or more entities involved with a communication.

2. Trustworthiness Claims Specifics a Verifier asserts with regards to its trustworthiness findings about an Attester.

3. Claim Freshness Establishes the time of last update (or refresh) of Trustworthiness Claims.

The following sections detail requirements for these three categories.

2.2. Non-repudiable Identity

Identity Evidence must be conveyed during the establishment of any trust-based relationship. Specific use cases will define the minimum types of identities required by a particular Relying Party as it evaluates Attestation Results, and perhaps additional associated Evidence. At a bare minimum, a Relying Party MUST start with the ability to verify the identity of a Verifier it chooses to trust.

Attester identities may then be acquired through signed or encrypted communications with the Verifier identity and/or the pre-provisioning Attester public keys in the Attester.

During the Remote Attestation process, the Verifier's identity must be established with a Relying Party, often via a Verifier signature across recent Attestation Results. This Verifier identity could only have come from a key pair maintained by a trusted developer or operator of the Verifier.

Additionally, each set of Attestation Results must be provably and non-reputably bound to the identity of the original Attesting Environment which was evaluated by the Verifier. This is accomplished via satisfying two requirements. First the Verifier signed Attestation Results MUST include sufficient Identity Evidence to ensure that this Attesting Environment signature refers to the same Attesting Environment appraised by the Verifier. Second, where the passport model is used as a subsystem, an Attesting Environment signature which spans the Verifier signature MUST also be included. As the Verifier signature already spans the Attester Identity as well as the Attestation Results, this restricts the viability of spoofing attacks.

In a subset of use cases, these two pieces of Identity Evidence may be sufficient for a Relying Party to successfully meet the criteria for its Appraisal Policy for Attestation Results. If the use case is a connection request, a Relying Party may simply then establish a transport session with an Attester after a successful appraisal. However an Appraisal Policy for Attestation Results will often be more nuanced, and the Relying Party may need additional information. Some Identity Evidence related policy questions which the Relying Party may consider include:

* Does the Relying Party only trust this Verifier to make Trustworthiness Claims on behalf a specific type of Attesting Environment? Might a mix of Verifiers be necessary to cover all mandatory Trustworthiness Claims?

* Does the Relying Party only accept connections from a verified-authentic software build from a specific software developer?

* Does the Relying Party only accept connections from specific preconfigured list of Attesters?

For any of these more nuanced appraisals, additional Identity Evidence or other policy related information must be conveyed or pre-provisioned during the formation of a trust context between the Relying Party, the Attester, the Attester's Attesting Environment, and the Verifier.

2.2.1. Attester and Attesting Environment

Per [RFC9334] Figure 2, an Attester and a corresponding Attesting Environment might not share common code or even hardware boundaries. Consequently, an Attester implementation needs to ensure that any Evidence which originates from outside the Attesting Environment MUST have been collected and delivered securely before any Attesting Environment signing may occur. After the Verifier performs its appraisal, it will include sufficient information in the Attestation Results to enable a Relying Party to have confidence that the Attester's trustworthiness is represented via Trustworthiness Claims signed by the appropriate Attesting Environment.

This document recognizes three general categories of Attesters.

1. HSM-based: A Hardware Security Module (HSM) based cryptoprocessor which hashes one or more streams of security measurements from an Attester within the Attesting Environment. Maintenance of this hash enables detection of an Attester which is not reporting the exact set of security measurements (such as log entries) taken within the Attesting Environment. An example of a HSM is a TPM2.0 [TPM2.0].

2. Process-based: An individual process which has its runtime memory encrypted by an Attesting Environment in a way that no other processes can read and decrypt that memory (e.g., [SGX] or [I-D.tschofenig-rats-psa-token].)

3. VM-based: An entire Guest VM (or a set of containers within a host) have been encrypted as a walled-garden unit by an Attesting Environment. The result is that the host operating system cannot read and decrypt what is executing within that VM (e.g., [SEV-SNP] or [TDX].)

Each of these categories of Attesters above will be capable of generating Evidence which is protected using private keys / certificates which are not accessible outside of the corresponding Attesting Environment. The owner of these secrets is the owner of the identity which is bound within the Attesting Environment. Effectively this means that for any Attester identity, there will exist a chain of trust ultimately bound to a hardware-based root of trust in the Attesting Environment. It is upon this root of trust that unique, non-repudiable Attester identities may be founded.

There are several types of Attester identities defined in this document. This list is extensible:

* chip-vendor: the vendor of the hardware chip used for the Attesting Environment (e.g., a primary Endorsement Key from a TPM)

* chip-hardware: specific hardware with specific firmware from an 'chip-vendor'

* target-environment: a unique instance of a software build running in an Attester (e.g., MRENCLAVE [SGX], an Instance ID [I-D.tschofenig-rats-psa-token], an Identity Block [SEV-SNP], or a hash which represents a set of software loaded since boot (e.g., TPM based integrity verification.))

* target-developer: the organizational unit responsible for a particular 'target-environment' (e.g., MRSIGNER [SGX])

* instance: a unique instantiated instance of an Attesting Environment running on 'chip-hardware' (e.g., an LDevID [IEEE802.1AR])

Based on the category of the Attesting Environment, different types of identities might be exposed by an Attester.

Attester Identity type	Process-based	VM-based	HSM-based
chip-vendor	Mandatory	Mandatory	Mandatory
chip-hardware	Mandatory	Mandatory	Mandatory
target-environment	Mandatory	Mandatory	Optional
target-developer	Mandatory	Optional	Optional
instance	Optional	Optional	Optional

Table 1

It is expected that drafts subsequent to this specification will provide the definitions and value domains for specific identities, each of which falling within the Attester identity types listed above. In some cases the actual unique identities might encoded as complex structures. An example complex structure might be a 'target-environment' encoded as a Software Bill of Materials (SBOM).

With the identity definitions and value domains, a Relying Party will have sufficient information to ensure that the Attester identities and Trustworthiness Claims asserted are actually capable of being supported by the underlying type of Attesting Environment. Consequently, the Relying Party SHOULD require Identity Evidence which indicates of the type of Attesting Environment when it considers its Appraisal Policy for Attestation Results.

2.2.2. Verifier

For the Verifier identity, it is critical for a Relying Party to review the certificate and chain of trust for that Verifier. Additionally, the Relying Party must have confidence that the Trustworthiness Claims being relied upon from the Verifier considered the chain of trust for the Attesting Environment.

There are two categorizations of Verifier identities defined in this document:

* verifier build: a unique instance of a software build running as a Verifier.

* verifier developer: the organizational unit responsible for a particular 'verifier build'.

Within each category, communicating the identity can be accomplished via a variety of objects and encodings.

2.2.3. Communicating Identity

Any of the above identities used by the Appraisal Policy for Attestation Results needed to be pre-established by the Relying Party before, or provided during, the exchange of Attestation Results. When provided during this exchange, the identity may be communicated either implicitly or explicitly.

An example of explicit communication would be to include the following Identity Evidence directly within the Attestation Results: a unique identifier for an Attesting Environment, the name of a key which can be provably associated with that unique identifier, and the set of Attestation Results which are signed using that key. As these Attestation Results are signed by the Verifier, it is the Verifier which is explicitly asserting the credentials it believes are trustworthy.

An example of implicit communication would be to include Identity Evidence in the form of a signature which has been placed over the Attestation Results asserted by a Verifier. It would be then up to the Relying Party's Appraisal Policy for Attestation Results to extract this signature and confirm that it only could have been generated by an Attesting Environment having access to a specific private key. This implicit identity communication is only viable if the Attesting Environment's public key is already known by the Relying Party.

One final step in communicating identity is proving the freshness of the Attestation Results to the degree needed by the Relying Party. A typical way to accomplish this is to include an element of freshness be embedded within a signed portion of the Attestation Results. This element of freshness reduces the identity spoofing risks from a replay attack. For more on this, see Section 2.4.

2.3. Trustworthiness Claims

2.3.1. Design Principles

Trust is not absolute. Trust is a belief in some aspect about an entity (in this case an Attester), and that this aspect is something which can be depended upon (in this case by a Relying Party.) Within the context of Remote Attestation, believability of this aspect is facilitated by a Verifier. This facilitation depends on the Verifier's ability to parse detailed Evidence from an Attester and then to assert conclusions about this aspect in a way interpretable

by a Relying Party.

Specific aspects for which a Verifier will assert trustworthiness are defined in this section. These are known as Trustworthiness Claims. These claims have been designed to enable a common understanding between a broad array of Attesters, Verifiers, and Relying Parties. The following set of design principles have been applied in the Trustworthiness Claim definitions:

1. Expose a small number of Trustworthiness Claims.

   Reason: a plethora of similar Trustworthiness Claims will result in divergent choices made on which to support between different Verifiers. This would place a lot of complexity in the Relying Party as it would be up to the Relying Party (and its policy language) to enable normalization across rich but incompatible Verifier object definitions.

2. Each Trustworthiness Claim enumerates only the specific states that could viably result in a different outcome after the Policy for Attestation Results has been applied.

   Reason: by explicitly disallowing the standardization of enumerated states which cannot easily be connected to a use case, we avoid forcing implementers from making incompatible guesses on what these states might mean.

3. Verifier and RP developers need explicit definitions of each state in order to accomplish the goals of (1) and (2).

   Reason: without such guidance, the Verifier will append plenty of raw supporting info. This relieves the Verifier of making the hard decisions. Of course, this raw info will be mostly non-interpretable and therefore non-actionable by the Relying Party.

4. Support standards and non-standard extensibility for (1) and (2).

   Reason: standard types of Verifier generated Trustworthiness Claims should be vetted by the full RATS working group, rather than being maintained in a repository which doesn’t follow the RFC process. This will keep a tight lid on extensions which must be considered by the Relying Party’s policy language. Because this process takes time, non-standard extensions will be needed for implementation speed and flexibility.

These design principles are important to keep the number of Verifier generated claims low, and to retain the complexity in the Verifier rather than the Relying Party.

2.3.2. Enumeration Encoding

Per design principle (2), each Trustworthiness Claim will only expose specific encoded values. To simplify the processing of these enumerations by the Relying Party, the enumeration will be encoded as a single signed 8 bit integer. These value assignments for this integer will be in four Trustworthiness Tiers which follow these guidelines:

None: The Verifier makes no assertions regarding this aspect of trustworthiness.

* Value 0: The Evidence received is insufficient to make a conclusion. Note: this should always be always treated equivalently by the Relying Party as no claim being made. I.e., the RP's Appraisal Policy for Attestation Results SHOULD NOT make any distinction between a Trustworthiness Claim with enumeration '0', and no Trustworthiness Claim being provided.

* Value 1: The Evidence received contains unknown elements which the Verifier is unable to evaluate. An example might be that the wrong type of Evidence has been delivered. Another case is that of Evidence coming from a composite Attester: a Verifier may understand only part of it and leave as "unknown" the Trustworthiness claims related to features it can't appraise.

* Value -1: A verifier malfunction occurred during the Verifier's appraisal processing.

Affirming: The Verifier affirms the Attester support for this aspect of trustworthiness.

* Values 2 to 31: A standards enumerated reason for affirming.

* Values -2 to -32: A non-standard reason for affirming.

Warning: The Verifier warns about this aspect of trustworthiness.

* Values 32 to 95: A standards enumerated reason for the warning.

* Values -33 to -96: A non-standard reason for the warning.

Contraindicated: The Verifier asserts the Attester is explicitly untrustworthy in regard to this aspect.

* Values 96 to 127: A standards enumerated reason for the contraindication.

* Values -97 to -128: A non-standard reason for the contraindication.

This enumerated encoding listed above will simplify the Appraisal Policy for Attestation Results. Such a policies may be as simple as saying that a specific Verifier has recently asserted Trustworthiness Claims, all of which are Affirming.

2.3.3. Assigning a Trustworthiness Claim value

In order to simplify design, only a single encoded value is asserted by a Verifier for any Trustworthiness Claim within a using the following process.

1. If applicable, a Verifier MUST assign a standardized value from the Contraindicated tier.

2. Else if applicable, a Verifier MUST assign a non-standardized value from the Contraindicated tier.

3. Else if applicable, a Verifier MUST assign a standardized value from the Warning tier.

4. Else if applicable, a Verifier MUST assign a non-standardized value from the Warning tier.

5. Else if applicable, a Verifier MUST assign a standardized value from the Affirming tier.

6. Else if applicable, a Verifier MUST assign a non-standardized value from the Affirming tier.

7. Else a Verifier MAY assign a 0 or -1.

2.3.4. Specific Claims

Following are the Trustworthiness Claims and their supported enumerations which may be asserted by a Verifier:

configuration: A Verifier has appraised an Attester's configuration, and is able to make conclusions regarding the exposure of known vulnerabilities

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: The configuration is a known and approved config.

3: The configuration includes or exposes no known vulnerabilities.

32: The configuration includes or exposes known vulnerabilities.

36: Elements of the configuration relevant to security are unavailable to the Verifier.

96: The configuration is unsupportable as it exposes unacceptable security vulnerabilities.

99: Cryptographic validation of the Evidence has failed.

executables: A Verifier has appraised and evaluated relevant runtime files, scripts, and/or other objects which have been loaded into the Target environment's memory.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: Only a recognized genuine set of approved executables, scripts, files, and/or objects have been loaded during and after the boot process.

3: Only a recognized genuine set of approved executables have been loaded during the boot process.

32: Only a recognized genuine set of executables, scripts, files, and/or objects have been loaded. However the Verifier cannot vouch for a subset of these due to known bugs or other known vulnerabilities.

33: Runtime memory includes executables, scripts, files, and/or objects which are not recognized.

96: Runtime memory includes executables, scripts, files, and/or object which are contraindicated.

99: Cryptographic validation of the Evidence has failed.

file-system: A Verifier has evaluated a specific set of directories within the Attester's file system. (Note: the Verifier may or may not indicate what these directory and expected files are via an unspecified management interface.)

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: Only a recognized set of approved files are found.

32: The file system includes unrecognized executables, scripts, or files.

96: The file system includes contraindicated executables, scripts, or files.

99: Cryptographic validation of the Evidence has failed.

hardware: A Verifier has appraised any Attester hardware and firmware which are able to expose fingerprints of their identity and running code.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: An Attester has passed its hardware and/or firmware verifications needed to demonstrate that these are genuine/supported.

32: An Attester contains only genuine/supported hardware and/or firmware, but there are known security vulnerabilities.

96: Attester hardware and/or firmware is recognized, but its trustworthiness is contraindicated.

97: A Verifier does not recognize an Attester's hardware or firmware, but it should be recognized.

99: Cryptographic validation of the Evidence has failed.

instance-identity: A Verifier has appraised an Attesting Environment's unique identity based upon private key signed Evidence which can be correlated to a unique instantiated instance of the Attester. (Note: this Trustworthiness Claim should only be generated if the Verifier actually expects to recognize the unique identity of the Attester.)

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: The Attesting Environment is recognized, and the associated instance of the Attester is not known to be compromised.

96: The Attesting Environment is recognized, and but its unique private key indicates a device which is not trustworthy.

97: The Attesting Environment is not recognized; however the Verifier believes it should be.

99: Cryptographic validation of the Evidence has failed.

runtime-opaque: A Verifier has appraised the visibility of Attester objects in memory from perspectives outside the Attester.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: the Attester's executing Target Environment and Attesting Environments are encrypted and within Trusted Execution Environment(s) opaque to the operating system, virtual machine manager, and peer applications. (Note: This value corresponds to the protections asserted by O.RUNTIME_CONFIDENTIALITY from [GP-TEE-PP])

32: the Attester's executing Target Environment and Attesting Environments inaccessible from any other parallel application or Guest VM running on the Attester's physical device. (Note that unlike "1" these environments are not encrypted in a way which restricts the Attester's root operator visibility. See O.TA_ISOLATION from [GP-TEE-PP].)

96: The Verifier has concluded that in memory objects are unacceptably visible within the physical host that supports the Attester.

99: Cryptographic validation of the Evidence has failed.

sourced-data: A Verifier has evaluated of the integrity of data objects from external systems used by the Attester.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: All essential Attester source data objects have been provided by other Attester(s) whose most recent appraisal(s) had both no Trustworthiness Claims of "0" where the current Trustworthiness Claim is "Affirming", as well as no "Warning" or "Contraindicated" Trustworthiness Claims.

32: Attester source data objects come from unattested sources, or attested sources with "Warning" type Trustworthiness Claims.

96: Attester source data objects come from contraindicated sources.

99: Cryptographic validation of the Evidence has failed.

storage-opaque: A Verifier has appraised that an Attester is capable of encrypting persistent storage. (Note: Protections must meet the capabilities of [OMTP-ATE] Section 5, but need not be hardware tamper resistant.)

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: the Attester encrypts all secrets in persistent storage via using keys which are never visible outside an HSM or the Trusted Execution Environment hardware.

32: the Attester encrypts all persistently stored secrets, but without using hardware backed keys

96: There are persistent secrets which are stored unencrypted in an Attester.

99: Cryptographic validation of the Evidence has failed.

It is possible for additonal Trustworthiness Claims and enumerated values to be defined in subsequent documents. At the same time, the standardized Trustworthiness Claim values listed above have been designed so there is no overlap within a Trustworthiness Tier. As a result, it is possible to imagine a future where overlapping Trustworthiness Claims within a single Trustworthiness Tier may be defined. Wherever possible, the Verifier SHOULD assign the best fitting standardized value.

Where a Relying Party doesn't know how to handle a particular Trustworthiness Claim, it MAY choose an appropriate action based on the Trustworthiness Tier under which the enumerated value fits.

It is up to the Verifier to publish the types of evaluations it performs when determining how Trustworthiness Claims are derived for a type of any particular type of Attester. It is out of the scope of this document for the Verifier to provide proof or specific logic on how a particular Trustworthiness Claim which it is asserting was derived.

2.3.5. Trustworthiness Vector

Multiple Trustworthiness Claims may be asserted about an Attesting Environment at single point in time. The set of Trustworthiness Claims inserted into an instance of Attestation Results by a Verifier is known as a Trustworthiness Vector. The order of Claims in the vector is NOT meaningful. A Trustworthiness Vector with no Trustworthiness Claims (i.e., a null Trustworthiness Vector) is a valid construct. In this case, the Verifier is making no Trustworthiness Claims but is confirming that an appraisal has been made.

2.3.6. Trustworthiness Vector for a type of Attesting Environment

Some Trustworthiness Claims are implicit based on the underlying type of Attesting Environment. For example, a validated MRSIGNER identity can be present where the underlying [SGX] hardware is 'hw-authentic'. Where such implicit Trustworthiness Claims exist, they do not have to be explicitly included in the Trustworthiness Vector. However, these implicit Trustworthiness Claims SHOULD be considered as being present by the Relying Party. Another way of saying this is if a Trustworthiness Claim is automatically supported as a result of coming from a specific type of TEE, that claim need not be

redundantly articulated. Such implicit Trustworthiness Claims can be seen in the tables within Appendix B.2 and Appendix B.3.

Additionally, there are some Trustworthiness Claims which cannot be adequately supported by an Attesting Environment. For example, it would be difficult for an Attester that includes only a TPM (and no other TEE) from ever having a Verifier appraise support for 'runtime-opaque'. As such, a Relying Party would be acting properly if it rejects any non-supportable Trustworthiness Claims asserted from a Verifier.

As a result, the need for the ability to carry a specific Trustworthiness Claim will vary by the type of Attesting Environment. Example mappings can be seen in Appendix B.

2.4. Freshness

A Relying Party will care about the recentness of the Attestation Results, and the specific Trustworthiness Claims which are embedded. All freshness mechanisms of [RFC9334], Section 10 are supportable by this specification.

Additionally, a Relying Party may track when a Verifier expires its confidence for the Trustworthiness Claims or the Trustworthiness Vector as a whole. Mechanisms for such expiry are not defined within this document.

There is a subset of secure interactions where the freshness of Trustworthiness Claims may need to be revisited asynchronously. This subset is when trustworthiness depends on the continuous availability of a transport session between the Attester and Relying Party. With such connectivity dependent Attestation Results, if there is a reboot which resets transport connectivity, all established Trustworthiness Claims should be cleared. Subsequent connection re-establishment will allow fresh new Trustworthiness Claims to be delivered.

3. Secure Interactions Models

There are multiple ways of providing a Trustworthiness Vector to a Relying Party. This section describes two alternatives.

3.1. Background-Check

3.1.1. Verifier Retrieval

It is possible to for a Relying Party to follow the Background-Check Model defined in Section 5.2 of [RFC9334]. In this case, a Relying Party will receive Attestation Results containing the Trustworthiness Vector directly from a Verifier. These Attestation Results can then be used by the Relying Party in determining the appropriate treatment for interactions with the Attester.

While applicable in some cases, the utilization of the Background-Check Model without modification has potential drawbacks in other cases. These include:

* Verifier scale: if the Attester has many Relying Parties, a Verifier appraising that Attester could be frequently be queried based on the same Evidence.

* Information leak: Evidence which the Attester might consider private can be visible to the Relying Party. Hiding that Evidence could devalue any resulting appraisal.

* Latency: a Relying Party will need to wait for the Verifier to return Attestation Results before proceeding with secure interactions with the Attester.

An implementer should examine these potential drawbacks before selecting this alternative.

3.1.2. Co-resident Verifier

A simplified Background-Check Model may exist in a very specific case. This is where the Relying Party and Verifier functions are co-resident. This model is appropriate when:

* Some hardware-based private key is used by an Attester while proving its identity as part of a mutually authenticated secure channel establishment with the Relying Party, and

* this Attester identity is accepted as sufficient proof of Attester integrity.

Effectively this means that detailed forensic capabilities of a robust Verifier are unnecessary because it is accepted that the code and operational behavior of the Attester cannot be manipulated after TEE initialization.

An example of such a scenario may be when an SGX's MRENCLAVE and MRSIGNER values have been associated with a known QUOTE value. And the code running within the TEE is not modifiable after launch.

3.2. Below Zero Trust

Zero Trust Architectures are referenced in [US-Executive-Order] eleven times. However despite this high profile, there is an architectural gap with Zero Trust. The credentials used for authentication and admission control can be manipulated on the endpoint. Attestation can fill this gap through the generation of a compound credential called AR-augmented Evidence. This compound credential is rooted in the hardware based Attesting Environment of an endpoint, plus the trustworthiness of a Verifier. The overall solution is known as "Below Zero Trust" as the compound credential cannot be manipulated or spoofed by an administrator of an endpoint with root access. This solution is not adversely impacted by the potential drawbacks with pure background-check described above.

To kick-off the "Below Zero Trust" compound credential creation sequence, a Verifier evaluates an Attester and returns signed Attestation Results back to this original Attester no less frequently than a well-known interval. This interval may also be asynchronous, based on the changing of certain Evidence as described in [I-D.ietf-rats-network-device-subscription].

When a Relying Party is to receive information about the Attester's trustworthiness, the Attesting Environment assembles the minimal set of Evidence which can be used to confirm or refute whether the Attester remains in the state of trustworthiness represented by the AR. To this Evidence, the Attesting Environment appends the signature from the most recent AR as well as a Relying Party Proof-of-Freshness. The Attesting Environment then signs the combination.

The Attester then assembles AR Augmented Evidence by taking the signed combination and appending the full AR. The assembly now consists of two independent but semantically bound sets of signed Evidence.

The AR Augmented Evidence is then sent to the Relying Party. The Relying Party then can appraise these semantically bound sets of signed Evidence by applying an Appraisal Policy for Attestation Results as described below. This policy will consider both the AR as well as additional information about the Attester within the AR Augmented Evidence the when determining what action to take.

This alternative combines the [RFC9334] Sections 5.1 Passport Model and Section 5.2 Background-Check Model. Figure 1 describes this flow of information. The flows within this combined model are mapped to [RFC9334] in the following way. "Verifier A" below corresponds to the "Verifier" Figure 5 within [RFC9334]. And "Relying Party/ Verifier B" below corresponds to the union of the "Relying Party" and "Verifier" boxes within Figure 6 of [RFC9334]. This union is possible because Verifier B can be implemented as a simple, self-contained process. The resulting combined process can appraise the AR-augmented Evidence to determine whether an Attester qualifies for secure interactions with the Relying Party. The specific steps of this process are defined later in this section.

```
  .----------------.
  | Attester       |
  | .-------------.|
  | | Attesting   ||             .----------.    .---------------.
  | | Environment ||             | Verifier |    | Relying Party |
  | '-------------'|             |     A    |    |  / Verifier B |
  '----------------'             '----------'    '---------------'
        time(VG)                       |                 |
          |<------Verifier PoF-------time(NS)            |
          |                            |                 |
 time(EG)(1)------Evidence------------>|                 |
          |                         time(RG)             |
          |<------Attestation Results-(2)                |
          ~                            ~                 ~
        time(VG')?                     |                 |
          ~                            ~                 ~
          |<------Relying Party PoF-----------------(3)time(NS')
          |                            |                 |
time(EG')(4)------AR-augmented Evidence----------------->|
          |                            |   time(RG',RA')(5)
                                                        (6)
                                                         ~
                                                     time(RX')
```

Figure 1: Below Zero Trust

The interaction model depicted above includes specific time related events from Appendix A of [RFC9334]. With the identification of these time related events, time duration/interval tracking becomes possible. Such duration/interval tracking can become important if the Relying Party cares if too much time has elapsed between the Verifier PoF and Relying Party PoF. If too much time has elapsed, perhaps the Attestation Results themselves are no longer trustworthy.

Note that while time intervals will often be relevant, there is a simplified case that does not require a Relying Party's PoF in step (3). In this simplified case, the Relying Party trusts that the Attester cannot be meaningfully changed from the outside during any reportable interval. Based on that assumption, and when this is the case then the step of the Relying Party PoF can be safely omitted.

In all cases, appraisal policies define the conditions and prerequisites for when an Attester does qualify for secure interactions. To qualify, an Attester has to be able to provide all of the mandatory affirming Trustworthiness Claims and identities needed by a Relying Party's Appraisal Policy for Attestation Results, and none of the disqualifying detracting Trustworthiness Claims.

More details on each interaction step of Below Zero Trust are as follows. The numbers used in this sequence match to the numbered steps in Figure 1:

1. An Attester sends Evidence which is provably fresh to Verifier A at time(EG). Freshness from the perspective of Verifier A MAY be established with Verifier PoF such as a nonce.

2. Verifier A appraises (1), then sends the following items back to that Attester within Attestation Results:

    1. the verified identity of the Attesting Environment,

    2. the Verifier A appraised Trustworthiness Vector of an Attester,

    3. a freshness proof associated with the Attestation Results,

    4. a Verifier signature across (2.1) though (2.3).

3. At time(EG') a Relying Party PoF (such as a nonce) known to the Relying Party is sent to the Attester.

4. The Attester generates and sends AR-augmented Evidence to the Relying Party/Verifier B. This AR-augmented Evidence includes:

    1. The Attestation Results from (2)

    2. Any (optionally) new incremental Evidence from the Attesting Environment

3. Attestation Environment signature which spans a hash of the Attestation Results (such as the signature of (2.4)), the proof-of-freshness from (3), and (4.2). Note: this construct allows the delta of time between (2.3) and (3) to be definitively calculated by the Relying Party.

5. On receipt of (4), the Relying Party applies its Appraisal Policy for Attestation Results. At minimum, this appraisal policy process must include the following:

    1. Verify that (4.3) includes the nonce from (3).

    2. Use a local certificate to validate the signature (4.1).

    3. Verify that the hash from (4.3) matches (4.1)

    4. Use the identity of (2.1) to validate the signature of (4.3).

    5. Failure of any steps (5.1) through (5.4) means the link does not meet minimum validation criteria, therefore appraise the link as having a null Verifier B Trustworthiness Vector. Jump to step (6.1).

    6. When there is large or uncertain time gap between time(EG) and time(EG'), the link should be assigned a null Verifier B Trustworthiness Vector. Jump to step (6.1).

    7. Assemble the Verifier B Trustworthiness Vector

        1. Copy Verifier A Trustworthiness Vector to Verifier B Trustworthiness Vector

        2. Add implicit Trustworthiness Claims inherent to the type of TEE.

        3. Prune any Trustworthiness Claims unsupportable by the Attesting Environment.

        4. Prune any Trustworthiness Claims the Relying Party doesn't accept from this Verifier.

6. The Relying Party takes action based on Verifier B's appraised Trustworthiness Vector, and applies the Appraisal Policy for Attestation Results. Following is a reasonable process for such evaluation:

1. Prune any Trustworthiness Claims from the Trustworthiness Vector not used in the Appraisal Policy for Attestation Results.

2. Allow the information exchange from the Attester into a Relying Party context in the Appraisal Policy for Attestation Results where the Verifier B appraised Trustworthiness Vector includes all the mandatory Trustworthiness Claims are in the "Affirming" value range, and none of the disqualifying Trustworthiness Claims are in the "Contraindicated" value range.

3. Disallow any information exchange into a Relying Party context for which that Verifier B appraised Trustworthiness Vector is not qualified.

As link layer protocols re-authenticate, steps (1) to (2) and steps (3) to (6) will independently refresh. This allows the Trustworthiness of Attester to be continuously re-appraised. There are only specific event triggers which will drive the refresh of Evidence generation (1), Attestation Result generation (2), or AR-augmented Evidence generation (4):

* life-cycle events, e.g. a change to an Authentication Secret of the Attester or an update of a software component.

* uptime-cycle events, e.g. a hard reset or a re-initialization of an Attester.

* authentication-cycle events, e.g. a link-layer interface reset could result in a new (4).

3.3. Mutual Attestation

In the interaction models described above, each device on either side of a secure interaction may require remote attestation of its peer. This process is known as mutual-attestation. To support mutual-attestation, the interaction models listed above may be run independently on either side of the connection.

3.4. Transport Protocol Integration

Either unidirectional attestation or mutual attestation may be supported within the protocol interactions needed for the establishment of a single transport session. While this document does not mandate specific transport protocols, messages containing the Attestation Results and AR Augmented Evidence can be passed within an authentication framework such the EAP protocol [RFC5247]

over TLS [RFC8446].

4. Privacy Considerations

Privacy Considerations Text

5. Security Considerations

Security Considerations Text

6. IANA Considerations

See Body.

7. References

7.1. Normative References

[GP-TEE-PP] "Global Platform TEE Protection Profile v1.3", September 2020, <https://globalplatform.org/specs-library/tee-protection-profile-v1-3/>.

[OMTP-ATE] "Open Mobile Terminal Platform - Advanced Trusted Environment", May 2009, <https://www.gsma.com/newsroom/wp-content/uploads/2012/03/omtpadvancedtrustedenvironmentomtptr1v11.pdf>.

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

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

[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>.

7.2. Informative References

[I-D.ietf-rats-network-device-subscription]
Birkholz, H., Voit, E., and W. Pan, "Attestation Event Stream Subscription", Work in Progress, Internet-Draft, draft-ietf-rats-network-device-subscription-04, 10 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-network-device-subscription-04>.

[I-D.tschofenig-rats-psa-token]
Tschofenig, H., Frost, S., Brossard, M., Shaw, A. L., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", Work in Progress, Internet-Draft, draft-tschofenig-rats-psa-token-22, 21 February 2024, <https://datatracker.ietf.org/doc/html/draft-tschofenig-rats-psa-token-22>.

[IEEE802.1AR]
"802.1AR: Secure Device Identity", 2 August 2018, <https://ieeexplore.ieee.org/document/8423794>.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <https://www.rfc-editor.org/rfc/rfc5247>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[SEV-SNP] "AMD SEV-SNP: Stregthening VM Isolation with Integrity Protection and More", 2020, <https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf>.

[SGX] "Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives", 2017, <https://software.intel.com/content/dam/develop/external/us/en/documents/intel-sgx-support-for-third-party-attestation-801017.pdf>.

[TDX] "Intel Trust Domain Extensions", 2020, <https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf>.

[TPM-ID] "TPM Keys for Platform Identity for TPM 1.2", August 2015, <https://www.trustedcomputinggroup.org/wp-content/uploads/TPM_Keys_for_Platform_Identity_v1_0_r3_Final.pdf>.

[TPM2.0] "Trusted Platform Module Library - Part 1: Architecture", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.07-2014-03-13.pdf>.

[US-Executive-Order] "Executive Order on Improving the Nation's Cybersecurity", 12 May 2021, <https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>.

Appendix A. Implementation Guidance

A.1. Supplementing Trustworthiness Claims

What has been encoded into each Trustworthiness Claim is the domain of integer values which is likely to drive a different programmatic decision in the Relying Party's Appraisal Policy for Attestation Results. This will not be the only thing a Relying Party's Operations team might care to track for measurement or debugging purposes.

There is also the opportunity for the Verifier to include supplementary Evidence beyond a set of asserted Trustworthiness Claims. It is recommended that if supplementary Evidence is provided by the Verifier within the Attestation Results, that this supplementary Evidence includes a reference to a specific Trustworthiness Claim. This will allow a deeper understanding of some of the reasoning behind the integer value assigned.

Appendix B. Supportable Trustworthiness Claims

The following is a table which shows what Claims are supportable by different Attesting Environment types. Note that claims MAY BE implicit to an Attesting Environment type, and therefore do not have to be included in the Trustworthiness Vector to be considered as set by the Relying Party.

B.1. Supportable Trustworthiness Claims for HSM-based CC

Following are Trustworthiness Claims which MAY be set for a HSM-based Confidential Computing Attester. (Such as a TPM [TPM-ID].)

Trustworthiness Claim	Required?	Appraisal Method
configuration	Optional	Verifier evaluation of Attester reveals no configuration lines which expose the Attester to known security vulnerabilities. This may be done with or without the involvement of a TPM PCR.
executables	Yes	Checks the TPM PCRs for the static operating system, and for any tracked files subsequently loaded
file-system	No	Can be supported, but TPM tracking is unlikely
hardware	Yes	If TPM PCR check ok from BIOS checks, through Master Boot Record configuration
instance-identity	Optional	Check IDevID
runtime-opaque	n/a	TPMs are not recommended to provide a sufficient technology base for this Trustworthiness Claim.
sourced-data	n/a	TPMs are not recommended to provide a sufficient technology base for this Trustworthiness Claim.
storage-opaque	Minimal	With a TPM, secure storage space exists and is writeable by external applications. But the space is so limited that it often is used just be used to store keys.

Table 2

Setting the Trustworthiness Claims may follow the following logic at the Verifier A within (2) of Figure 1:

```
Start: Evidence received starts the generation of a new
Trustworthiness Vector.  (e.g.,  TPM Quote Received, log received,
or appraisal timer expired)

Step 0: set Trustworthiness Vector = Null

Step 1: Is there sufficient fresh signed evidence to appraise?
  (yes) - No Action
  (no) -  Goto Step 6

Step 2: Appraise Hardware Integrity PCRs
   if (hardware NOT "0") - push onto vector
   if (hardware NOT affirming or warning), go to Step 6

Step 3: Appraise Attesting Environment identity
   if (instance-identity <> "0") - push onto vector

Step 4: Appraise executable loaded and filesystem integrity
   if (executables NOT "0") - push onto vector
   if (executables NOT affirming or warning), go to Step 6

Step 5: Appraise all remaining Trustworthiness Claims
        Independently and set as appropriate.

Step 6: Assemble Attestation Results, and push to Attester

End
```

B.2. Supportable Trustworthiness Claims for process-based CC

Following are Trustworthiness Claims which MAY be set for a process-based Confidential Computing based Attester. (Such as a SGX Enclaves and TrustZone.)

Trustworthiness Claim	Required?	Appraisal Method
instance-identity	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
configuration	Optional	If done, this is at the Application Layer. Plus each process needs it own protection mechanism as the protection is limited to the process itself.
executables	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
file-system	Optional	Can be supported by application, but process-based CC is not a sufficient technology base for this Trustworthiness Claim.
hardware	Implicit in signature	At least the TEE is protected here. Other elements of the system outside of the TEE might need additional protections is used by the application process.
runtime-opaque	Implicit in signature	From the TEE
storage-opaque	Implicit in signature	Although the application must assert that this function is used by the code itself.
sourced-data	Optional	Will need to be supported by application code

Table 3

B.3. Supportable Trustworthiness Claims for VM-based CC

Following are Trustworthiness Claims which MAY be set for a VM-based Confidential Computing based Attester. (Such as SEV, TDX, ACCA, SEV-SNP.)

Trustworthiness Claim	Required?	Appraisal Method
instance-identity	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
configuration	Optional	Requires application integration. Easier than with process-based solution, as the whole protected machine can be evaluated.
executables	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
file-system	Optional	Can be supported by application
hardware	Chip dependent	At least the TEE is protected here. Other elements of the system outside of the TEE might need additional protections is used by the application process.
runtime-opaque	Implicit in signature	From the TEE
storage-opaque	Chip dependent	Although the application must assert that this function is used by the code itself.
sourced-data	Optional	Will need to be supported by application code

Table 4

Appendix C. Some issues being worked

It is possible for a cluster/hierarchy of Verifiers to have aggregate AR which are perhaps signed/endorsed by a lead Verifier. What should be the Proof-of-Freshness or Verifier associated with any of the aggregate set of Trustworthiness Claims?

There will need to be a subsequent document which documents how these objects which will be translated into a protocol on a wire (e.g. EAP on TLS). Some breakpoint between what is in this draft, and what is in specific drafts for wire encoding will need to be determined. Questions like architecting the cluster/hierarchy of Verifiers fall into this breakdown.

For some Trustworthiness Claims, there could be value in identifying a specific Appraisal Policy for Attestation Results applied within the Attester. One way this could be done would be a URI which identifies the policy used at Verifier A, and this URI would reference a specific Trustworthiness Claim. As the URI also could encode the version of the software, it might also act as a mechanism to signal the Relying Party to refresh/re-evaluate its view of Verifier A. Do we need this type of structure to be included here? Should it be in subsequent documents?

Expand the variant of Figure 1 which requires no Relying Party PoF into its own picture.

In what document (if any) do we attempt normalization of the identity claims between different types of TEE. E.g., does MRSIGNER plus extra loaded software = the sum of TrustZone Signer IDs for loaded components?

Appendix D. Contributors

Guy Fedorkow

Email: gfedorkow@juniper.net

Dave Thaler

Email: dthaler@microsoft.com

Ned Smith

Email: ned.smith@intel.com

Lawrence Lundblade

Email: lgl@island-resort.com

Authors' Addresses

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Thomas Hardjono
MIT
Email: hardjono@mit.edu

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

Vincent Scarlata
Intel
Email: vincent.r.scarlata@intel.com

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 1 April 2023

H. Birkholz
Fraunhofer SIT
D. Thaler
Microsoft
M. Richardson
Sandelman Software Works
N. Smith
Intel
W. Pan
Huawei Technologies
28 September 2022

Remote Attestation Procedures Architecture
draft-ietf-rats-architecture-22

Abstract

In network protocol exchanges it is often useful for one end of a communication to know whether the other end is in an intended operating state. This document provides an architectural overview of the entities involved that make such tests possible through the process of generating, conveying, and evaluating evidentiary claims. An attempt is made to provide for a model that is neutral toward processor architectures, the content of claims, and protocols.

Note to Readers

Discussion of this document takes place on the RATS Working Group mailing list (rats@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/rats/ (https://mailarchive.ietf.org/arch/browse/rats/).

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 1 April 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 3
2. Reference Use Cases 5
  2.1. Network Endpoint Assessment 5
  2.2. Confidential Machine Learning Model Protection 6
  2.3. Confidential Data Protection 6
  2.4. Critical Infrastructure Control 6
  2.5. Trusted Execution Environment Provisioning 7
  2.6. Hardware Watchdog 7
  2.7. FIDO Biometric Authentication 8
3. Architectural Overview 8
  3.1. Two Types of Environments of an Attester 10
  3.2. Layered Attestation Environments 12
  3.3. Composite Device 14
  3.4. Implementation Considerations 16
4. Terminology 17
  4.1. Roles 17
  4.2. Artifacts 18
5. Topological Patterns 20
  5.1. Passport Model 20
  5.2. Background-Check Model 21
  5.3. Combinations 22
6. Roles and Entities 24
7. Trust Model 24
  7.1. Relying Party 24
  7.2. Attester 26

7.3. Relying Party Owner 26
7.4. Verifier 26
7.5. Endorser, Reference Value Provider, and Verifier Owner . 28
8. Conceptual Messages 29
8.1. Evidence 29
8.2. Endorsements 29
8.3. Reference Values 30
8.4. Attestation Results 30
8.5. Appraisal Policies 31
9. Claims Encoding Formats 32
10. Freshness 33
10.1. Explicit Timekeeping using Synchronized Clocks 34
10.2. Implicit Timekeeping using Nonces 35
10.3. Implicit Timekeeping using Epoch IDs 35
10.4. Discussion 36
11. Privacy Considerations 37
12. Security Considerations 38
12.1. Attester and Attestation Key Protection 38
12.1.1. On-Device Attester and Key Protection 39
12.1.2. Attestation Key Provisioning Processes 39
12.2. Conceptual Message Protection 41
12.3. Epoch ID-based Attestation 42
12.4. Trust Anchor Protection 42
13. IANA Considerations 43
14. Acknowledgments 43
15. Notable Contributions 43
16. References 43
16.1. Normative References 43
16.2. Informative References 44
Appendix A. Time Considerations 46
A.1. Example 1: Timestamp-based Passport Model Example 48
A.2. Example 2: Nonce-based Passport Model Example 50
A.3. Example 3: Epoch ID-based Passport Model Example 52
A.4. Example 4: Timestamp-based Background-Check Model Example 53
A.5. Example 5: Nonce-based Background-Check Model Example . . 54
Contributors 55
Authors' Addresses 57

1. Introduction

The question of how one system can know that another system can be trusted has found new interest and relevance in a world where trusted computing elements are maturing in processor architectures.

Systems that have been attested and verified to be in a good state (for some value of "good") can improve overall system posture. Conversely, systems that cannot be attested and verified to be in a good state can be given reduced access or privileges, taken out of service, or otherwise flagged for repair.

For example:

* A bank backend system might refuse to transact with another system that is not known to be in a good state.

* A healthcare system might refuse to transmit electronic healthcare records to a system that is not known to be in a good state.

In Remote Attestation Procedures (RATS), one peer (the "Attester") produces believable information about itself - Evidence - to enable a remote peer (the "Relying Party") to decide whether to consider that Attester a trustworthy peer or not. RATS are facilitated by an additional vital party, the Verifier.

The Verifier appraises Evidence via appraisal policies and creates the Attestation Results to support Relying Parties in their decision process. This document defines a flexible architecture consisting of attestation roles and their interactions via conceptual messages. Additionally, this document defines a universal set of terms that can be mapped to various existing and emerging Remote Attestation Procedures. Common topological patterns and the sequence of data flows associated with them, such as the "Passport Model" and the "Background-Check Model", are illustrated. The purpose is to define useful terminology for remote attestation and enable readers to map their solution architecture to the canonical attestation architecture provided here. Having a common terminology that provides well-understood meanings for common themes such as roles, device composition, topological patterns, and appraisal procedures is vital for semantic interoperability across solutions and platforms involving multiple vendors and providers.

Amongst other things, this document is about trust and trustworthiness. Trust is a choice one makes about another system. Trustworthiness is a quality about the other system that can be used in making one’s decision to trust it or not. This is a subtle difference and being familiar with the difference is crucial for using this document. Additionally, the concepts of freshness and trust relationships with respect to RATS are elaborated on to enable implementers to choose appropriate solutions to compose their Remote Attestation Procedures.

2. Reference Use Cases

This section covers a number of representative and generic use cases for remote attestation, independent of specific solutions. The purpose is to provide motivation for various aspects of the architecture presented in this document. Many other use cases exist, and this document does not intend to have a complete list, only to illustrate a set of use cases that collectively cover all the functionality required in the architecture.

Each use case includes a description followed by an additional summary of the Attester and Relying Party roles derived from the use case.

2.1. Network Endpoint Assessment

Network operators want trustworthy reports that include identity and version information about the hardware and software on the machines attached to their network. Examples of reports include purposes, such as inventory summaries, audit results, anomaly notifications, typically including the maintenance of log records or trend reports. The network operator may also want a policy by which full access is only granted to devices that meet some definition of hygiene, and so wants to get Claims about such information and verify its validity. Remote attestation is desired to prevent vulnerable or compromised devices from getting access to the network and potentially harming others.

Typically, a solution starts with a specific component (sometimes referred to as a root of trust) that often provides trustworthy device identity, and performs a series of operations that enables trustworthiness appraisals for other components. Such components perform operations that help determine the trustworthiness of yet other components, by collecting, protecting or signing measurements. Measurements that have been signed by such components are comprised of Evidence that when evaluated either supports or refutes a claim of trustworthiness. Measurements can describe a variety of attributes of system components, such as hardware, firmware, BIOS, software, etc.

Attester: A device desiring access to a network.

Relying Party: Network equipment such as a router, switch, or access point, responsible for admission of the device into the network.

2.2. Confidential Machine Learning Model Protection

A device manufacturer wants to protect its intellectual property. The intellectual property's scope primarily encompasses the machine learning (ML) model that is deployed in the devices purchased by its customers. The protection goals include preventing attackers, potentially the customer themselves, from seeing the details of the model.

This typically works by having some protected environment in the device go through a remote attestation with some manufacturer service that can assess its trustworthiness. If remote attestation succeeds, then the manufacturer service releases either the model, or a key to decrypt a model already deployed on the Attester in encrypted form, to the requester.

Attester: A device desiring to run an ML model.

Relying Party: A server or service holding ML models it desires to protect.

2.3. Confidential Data Protection

This is a generalization of the ML model use case above, where the data can be any highly confidential data, such as health data about customers, payroll data about employees, future business plans, etc. As part of the attestation procedure, an assessment is made against a set of policies to evaluate the state of the system that is requesting the confidential data. Attestation is desired to prevent leaking data via compromised devices.

Attester: An entity desiring to retrieve confidential data.

Relying Party: An entity that holds confidential data for release to authorized entities.

2.4. Critical Infrastructure Control

Potentially harmful physical equipment (e.g., power grid, traffic control, hazardous chemical processing, etc.) is connected to a network in support of critical infrastructure. The organization managing such infrastructure needs to ensure that only authorized code and users can control corresponding critical processes, and that these processes are protected from unauthorized manipulation or other threats. When a protocol operation can affect a critical system component of the infrastructure, devices attached to that critical component require some assurances depending on the security context, including that: a requesting device or application has not been

compromised, and the requesters and actors act on applicable policies. As such, remote attestation can be used to only accept commands from requesters that are within policy.

Attester: A device or application wishing to control physical equipment.

Relying Party: A device or application connected to potentially dangerous physical equipment (hazardous chemical processing, traffic control, power grid, etc.).

2.5. Trusted Execution Environment Provisioning

A Trusted Application Manager (TAM) server is responsible for managing the applications running in a Trusted Execution Environment (TEE) of a client device, as described in [I-D.ietf-teep-architecture]. To achieve its purpose, the TAM needs to assess the state of a TEE, or of applications in the TEE, of a client device. The TEE conducts Remote Attestation Procedures with the TAM, which can then decide whether the TEE is already in compliance with the TAM's latest policy. If not, the TAM has to uninstall, update, or install approved applications in the TEE to bring it back into compliance with the TAM's policy.

Attester: A device with a TEE capable of running trusted applications that can be updated.

Relying Party: A TAM.

2.6. Hardware Watchdog

There is a class of malware that holds a device hostage and does not allow it to reboot to prevent updates from being applied. This can be a significant problem, because it allows a fleet of devices to be held hostage for ransom.

A solution to this problem is a watchdog timer implemented in a protected environment such as a Trusted Platform Module (TPM), as described in [TCGarch] section 43.3. If the watchdog does not receive regular, and fresh, Attestation Results as to the system's health, then it forces a reboot.

Attester: The device that should be protected from being held hostage for a long period of time.

Relying Party: A watchdog capable of triggering a procedure that resets a device into a known, good operational state.

2.7. FIDO Biometric Authentication

In the Fast IDentity Online (FIDO) protocol [WebAuthN], [CTAP], the device in the user's hand authenticates the human user, whether by biometrics (such as fingerprints), or by PIN and password. FIDO authentication puts a large amount of trust in the device compared to typical password authentication because it is the device that verifies the biometric, PIN and password inputs from the user, not the server. For the Relying Party to know that the authentication is trustworthy, the Relying Party needs to know that the Authenticator part of the device is trustworthy. The FIDO protocol employs remote attestation for this.

The FIDO protocol supports several remote attestation protocols and a mechanism by which new ones can be registered and added. Remote attestation defined by RATS is thus a candidate for use in the FIDO protocol.

Attester: FIDO Authenticator.

Relying Party: Any web site, mobile application backend, or service that relies on authentication data based on biometric information.

3. Architectural Overview

Figure 1 depicts the data that flows between different roles, independent of protocol or use case.

```
 .--------.     .---------.       .--------.       .-------------.
| Endorser |   | Reference |     | Verifier |     | Relying Party |
 '-+------'    | Value     |     | Owner    |     | Owner         |
   |           | Provider  |      '---+----'       '----+--------'
   |            '-----+---'           |                 |
   |                  |               |                 |
   | Endorsements     | Reference     | Appraisal       | Appraisal
   |                  | Values        | Policy for      | Policy for
   |                  |               | Evidence        | Attestation
    '-----------.     |               |                 | Results
                |     |               |                 |
                v     v               v                 |
               .-------------------------.              |
       .------>|        Verifier         +-----.        |
      |        '-------------------------'      |       |
      |                                         |       |
      | Evidence                  Attestation   |       |
      |                           Results       |       |
      |                                         |       |
      |                                         v       v
 .-----+----.                             .---------------.
| Attester |                             | Relying Party |
 '----------'                             '---------------'
```

Figure 1: Conceptual Data Flow

The text below summarizes the activities conducted by the roles illustrated in Figure 1. Roles are assigned to entities. Entities are often system components [RFC4949], such as devices. As the term device is typically more intuitive than the term entity or system component, device is often used as an illustrative synonym throughout this document.

The Attester role is assigned to entities that create Evidence that is conveyed to a Verifier.

The Verifier role is assigned to entities that use the Evidence, any Reference Values from Reference Value Providers, and any Endorsements from Endorsers, by applying an Appraisal Policy for Evidence to assess the trustworthiness of the Attester. This procedure is called the appraisal of Evidence.

Subsequently, the Verifier role generates Attestation Results for use by Relying Parties.

The Appraisal Policy for Evidence might be obtained from the Verifier Owner via some protocol mechanism, or might be configured into the Verifier by the Verifier Owner, or might be programmed into the Verifier, or might be obtained via some other mechanism.

The Relying Party role is assigned to an entity that uses Attestation Results by applying its own appraisal policy to make application-specific decisions, such as authorization decisions. This procedure is called the appraisal of Attestation Results.

The Appraisal Policy for Attestation Results might be obtained from the Relying Party Owner via some protocol mechanism, or might be configured into the Relying Party by the Relying Party Owner, or might be programmed into the Relying Party, or might be obtained via some other mechanism.

See Section 8 for further discussion of the conceptual messages shown in Figure 1. Section 4 provides a more complete definition of all RATS roles.

3.1. Two Types of Environments of an Attester

As shown in Figure 2, an Attester consists of at least one Attesting Environment and at least one Target Environment co-located in one entity. In some implementations, the Attesting and Target Environments might be combined into one environment. Other implementations might have multiple Attesting and Target Environments, such as in the examples described in more detail in Section 3.2 and Section 3.3. Other examples may exist. All compositions of Attesting and Target Environments discussed in this architecture can be combined into more complex implementations.

```
                 .-------------------------------.
                 |                               |
                 |            Verifier           |
                 |                               |
                 '-------------------------------'
                                ^
                                |
             .------------------|----------.
             |                  |          |
             |    .---------------.        |
             |    | Target        |        |
             |    | Environment   |        |
             |    |               |        | Evidence
             |    '--------------+-'        |
             |                   |          |
             |                   |          |
             |          Collect  |          |
             |           Claims  |          |
             |                   |          |
             |                   v          |
             |                .-------+-----.|
             |                | Attesting   |
             |                | Environment |
             |                |             |
             |                '-------------'
             |              Attester        |
             '--------------------------------'
```

Figure 2: Two Types of Environments

Claims are collected from Target Environments. That is, Attesting Environments collect the values and the information to be represented in Claims, by reading system registers and variables, calling into subsystems, taking measurements on code, memory, or other security related assets of the Target Environment. Attesting Environments then format the Claims appropriately, and typically use key material and cryptographic functions, such as signing or cipher algorithms, to generate Evidence. There is no limit to or requirement on the types of hardware or software environments that can be used to implement an Attesting Environment, for example: Trusted Execution Environments (TEEs), embedded Secure Elements (eSEs), Trusted Platform Modules (TPMs) [TCGarch], or BIOS firmware.

An arbitrary execution environment may not, by default, be capable of Claims collection for a given Target Environment. Execution environments that are designed specifically to be capable of Claims collection are referred to in this document as Attesting Environments. For example, a TPM doesn't actively collect Claims

itself, it instead requires another component to feed various values to the TPM. Thus, an Attesting Environment in such a case would be the combination of the TPM together with whatever component is feeding it the measurements.

3.2. Layered Attestation Environments

By definition, the Attester role generates Evidence. An Attester may consist of one or more nested environments (layers). The bottom layer of an Attester has an Attesting Environment that is typically designed to be immutable or difficult to modify by malicious code. In order to appraise Evidence generated by an Attester, the Verifier needs to trust various layers, including the bottom Attesting Environment. Trust in the Attester's layers, including the bottom layer, can be established in various ways as discussed in Section 7.4.

In layered attestation, Claims can be collected from or about each layer beginning with an initial layer. The corresponding Claims can be structured in a nested fashion that reflects the nesting of the Attester's layers. Normally, Claims are not self-asserted, rather a previous layer acts as the Attesting Environment for the next layer. Claims about an initial layer typically are asserted by an Endorser.

The example device illustrated in Figure 3 includes (A) a BIOS stored in read-only memory, (B) a bootloader, and (C) an operating system kernel.

```
                .-------------.   Endorsement for ROM
                |  Endorser   +-----------------------.
                '-------------'                       |
                                                      v
                .-------------.   Reference      .----------.
                | Reference   |   Values for     |          |
                | Value       +----------------->| Verifier |
                | Provider(s) | ROM, bootloader, |          |
                '-------------'    and kernel    '----------'
                                                      ^
            .-----------------------------------.     |
            |                                   |     |
            |   .---------------------------.   |     |
            |   | Kernel                    |   |     |
            |   |                           |   |     | Layered
            |   |   Target                  |   |     | Evidence
            |   | Environment               |   |     |   for
            |   '---------------+-----------'   |     | bootloader
            |           Collect |               |     |   and
            |           Claims  |               |     | kernel
            |   .---------------|-----------.   |     |
            |   | Bootloader    v           |   |     |
            |   |               .-----------. |   |     |
            |   |   Target      | Attesting | |   |     |
            |   | Environment   |Environment+-----------'
            |   |               |           | |   |
            |   |               '-----------' |   |
            |   |                   ^         |   |
            |   '---------------+---|---------'   |
            |           Collect |   | Evidence for |
            |           Claims  v   | bootloader   |
            |   .-----------------+---------.     |
            |   | ROM                       |     |
            |   |                           |     |
            |   |                Attesting  |     |
            |   |               Environment |     |
            |   '---------------------------'     |
            |                                     |
            '-------------------------------------'
```

Figure 3: Layered Attester

The first Attesting Environment, the ROM in this example, has to ensure the integrity of the bootloader (the first Target Environment). There are potentially multiple kernels to boot, and the decision is up to the bootloader. Only a bootloader with intact integrity will make an appropriate decision. Therefore, the Claims relating to the integrity of the bootloader have to be measured securely. At this stage of the boot-cycle of the device, the Claims collected typically cannot be composed into Evidence.

After the boot sequence is started, the BIOS conducts the most important and defining feature of layered attestation, which is that the successfully measured bootloader now becomes (or contains) an Attesting Environment for the next layer. This procedure in layered attestation is sometimes called "staging". It is important that the bootloader not be able to alter any Claims about itself that were collected by the BIOS. This can be ensured having those Claims be either signed by the BIOS or stored in a tamper-proof manner by the BIOS.

Continuing with this example, the bootloader’s Attesting Environment is now in charge of collecting Claims about the next Target Environment, which in this example is the kernel to be booted. The final Evidence thus contains two sets of Claims: one set about the bootloader as measured and signed by the BIOS, plus a set of Claims about the kernel as measured and signed by the bootloader.

This example could be extended further by making the kernel become another Attesting Environment for an application as another Target Environment. This would result in a third set of Claims in the Evidence pertaining to that application.

The essence of this example is a cascade of staged environments. Each environment has the responsibility of measuring the next environment before the next environment is started. In general, the number of layers may vary by device or implementation, and an Attesting Environment might even have multiple Target Environments that it measures, rather than only one as shown by example in Figure 3.

3.3. Composite Device

A composite device is an entity composed of multiple sub-entities such that its trustworthiness has to be determined by the appraisal of all these sub-entities.

Each sub-entity has at least one Attesting Environment collecting the Claims from at least one Target Environment, then this sub-entity generates Evidence about its trustworthiness. Therefore, each sub-

entity can be called an Attester. Among all the Attesters, there may be only some which have the ability to communicate with the Verifier while others do not.

For example, a carrier-grade router consists of a chassis and multiple slots. The trustworthiness of the router depends on all its slots' trustworthiness. Each slot has an Attesting Environment, such as a TEE, collecting the Claims of its boot process, after which it generates Evidence from the Claims.

Among these slots, only a "main" slot can communicate with the Verifier while other slots cannot. But other slots can communicate with the main slot by the links between them inside the router. So the main slot collects the Evidence of other slots, produces the final Evidence of the whole router and conveys the final Evidence to the Verifier. Therefore, the router is a composite device, each slot is an Attester, and the main slot is the lead Attester.

Another example is a multi-chassis router composed of multiple single carrier-grade routers. Multi-chassis router setups create redundancy groups that provide higher throughput by interconnecting multiple routers in these groups, which can be treated as one logical router for simpler management. A multi-chassis router setup provides a management point that connects to the Verifier. Typically, one router in the group is designated as the main router. Other routers in the multi-chassis setup are connected to the main router only via physical network links and are therefore managed and appraised via the main router's help. Consequently, a multi-chassis router setup is a composite device, each router is an Attester, and the main router is the lead Attester.

Figure 4 depicts the conceptual data flow for a composite device.

```
                      .----------------------------.
                      |          Verifier          |
                      '----------------------------'
                                     ^
                                     |
                                     | Evidence of
                                     | Composite Device
                                     |
   .---------------------------------|-------------------------------.
   | .-------------------------------|-----.      .------------.     |
   | |  Collect             .---------+--.  |      |            |     |
   | |  Claims   .--------->|  Attesting |<--------+ Attester B +-.   |
   | |           |          |Environment |  |      '------------' |   |
   | |  .--------+-------.  |            |<----------+ Attester C +-. |
   | |  |     Target     |  |            |  |        '------------' | |
   | |  | Environment(s) |  |            |<------------+ ...        |
   | |  |                |  '------------'  | Evidence '------------'
   | |  '----------------'                   |    of
   | |                                       | Attesters
   | | lead Attester A                       | (via Internal Links or
   | '---------------------------------------' Network Connections)
   |                                                                 |
   |                       Composite Device                          |
   '-----------------------------------------------------------------'
```

Figure 4: Composite Device

In a composite device, each Attester generates its own Evidence by its Attesting Environment(s) collecting the Claims from its Target Environment(s). The lead Attester collects Evidence from other Attesters and conveys it to a Verifier. Collection of Evidence from sub-entities may itself be a form of Claims collection that results in Evidence asserted by the lead Attester. The lead Attester generates Evidence about the layout of the whole composite device, while sub-Attesters generate Evidence about their respective (sub-)modules.

In this scenario, the trust model described in Section 7 can also be applied to an inside Verifier.

3.4. Implementation Considerations

An entity can take on multiple RATS roles (e.g., Attester, Verifier, Relying Party, etc.) at the same time. Multiple entities can cooperate to implement a single RATS role as well. In essence, the combination of roles and entities can be arbitrary. For example, in the composite device scenario, the entity inside the lead Attester can also take on the role of a Verifier, and the outer entity of

Verifier can take on the role of a Relying Party. After collecting the Evidence of other Attesters, this inside Verifier uses Endorsements and appraisal policies (obtained the same way as by any other Verifier) as part of the appraisal procedures that generate Attestation Results. The inside Verifier then conveys the Attestation Results of other Attesters to the outside Verifier, whether in the same conveyance protocol as part of the Evidence or not.

As explained in Section 4, there are a variety of roles in the RATS architecture and they are defined by a unique combination of artifacts they produce and consume. Conversely, artifacts are also defined by the roles that produce or consume them. To produce an artifact means that a given role introduces it into the RATS architecture. To consume an artifact means that a given role has responsibility for processing it in the RATS architecture. Roles also have the ability to perform additional actions such as caching or forwarding artifacts as opaque data. As depicted in Section 5, these additional actions can be performed by several roles.

4. Terminology

[RFC4949] has defined a number of terms that are also used in this document. Some of the terms are close to, but not exactly the same. Where the terms are similar, they are noted below with references. As explained in [RFC4949], Section 2.6 when this document says "Compare:", the terminology used in this document differs significantly from the definition in the reference.

This document uses the following terms.

4.1. Roles

Attester: A role performed by an entity (typically a device) whose Evidence must be appraised in order to infer the extent to which the Attester is considered trustworthy, such as when deciding whether it is authorized to perform some operation.

Produces: Evidence

Relying Party: A role performed by an entity that depends on the validity of information about an Attester, for purposes of reliably applying application specific actions. Compare: /relying party/ in [RFC4949].

Consumes: Attestation Results, Appraisal Policy for Attestation Results

Verifier: A role performed by an entity that appraises the validity of Evidence about an Attester and produces Attestation Results to be used by a Relying Party.

Consumes: Evidence, Reference Values, Endorsements, Appraisal Policy for Evidence

Produces: Attestation Results

Relying Party Owner: A role performed by an entity (typically an administrator), that is authorized to configure Appraisal Policy for Attestation Results in a Relying Party.

Produces: Appraisal Policy for Attestation Results

Verifier Owner: A role performed by an entity (typically an administrator), that is authorized to configure Appraisal Policy for Evidence in a Verifier.

Produces: Appraisal Policy for Evidence

Endorser: A role performed by an entity (typically a manufacturer) whose Endorsements may help Verifiers appraise the authenticity of Evidence and infer further capabilities of the Attester.

Produces: Endorsements

Reference Value Provider: A role performed by an entity (typically a manufacturer) whose Reference Values help Verifiers appraise Evidence to determine if acceptable known Claims have been recorded by the Attester.

Produces: Reference Values

4.2. Artifacts

Claim: A piece of asserted information, often in the form of a name/value pair. Claims make up the usual structure of Evidence and other RATS artifacts. Compare: /claim/ in [RFC7519].

Endorsement: A secure statement that an Endorser vouches for the integrity of an Attester's various capabilities such as Claims collection and Evidence signing.

Consumed By: Verifier

Produced By: Endorser

Evidence: A set of Claims generated by an Attester to be appraised by a Verifier. Evidence may include configuration data, measurements, telemetry, or inferences.

Consumed By: Verifier

Produced By: Attester

Attestation Result: The output generated by a Verifier, typically including information about an Attester, where the Verifier vouches for the validity of the results.

Consumed By: Relying Party

Produced By: Verifier

Appraisal Policy for Evidence: A set of rules that informs how a Verifier evaluates the validity of information about an Attester. Compare: /security policy/ in [RFC4949].

Consumed By: Verifier

Produced By: Verifier Owner

Appraisal Policy for Attestation Results: A set of rules that direct how a Relying Party uses the Attestation Results regarding an Attester generated by the Verifiers. Compare: /security policy/ in [RFC4949].

Consumed by: Relying Party

Produced by: Relying Party Owner

Reference Values: A set of values against which values of Claims can be compared as part of applying an Appraisal Policy for Evidence. Reference Values are sometimes referred to in other documents as known-good values, golden measurements, or nominal values, although those terms typically assume comparison for equality, whereas here Reference Values might be more general and be used in any sort of comparison.

Consumed By: Verifier

Produced By: Reference Value Provider

5. Topological Patterns

Figure 1 shows a data-flow diagram for communication between an Attester, a Verifier, and a Relying Party. The Attester conveys its Evidence to the Verifier for appraisal, and the Relying Party receives the Attestation Result from the Verifier. This section refines the data-flow diagram by describing two reference models, as well as one example composition thereof. The discussion that follows is for illustrative purposes only and does not constrain the interactions between RATS roles to the presented patterns.

5.1. Passport Model

The passport model is so named because of its resemblance to how nations issue passports to their citizens. The nature of the Evidence that an individual needs to provide to its local authority is specific to the country involved. The citizen retains control of the resulting passport document and presents it to other entities when it needs to assert a citizenship or identity Claim, such as an airport immigration desk. The passport is considered sufficient because it vouches for the citizenship and identity Claims, and it is issued by a trusted authority.

Thus, in this immigration desk analogy, the citizen is the Attester, the passport issuing agency is a Verifier, and the passport application and identifying information (e.g., birth certificate) is the the Evidence. The passport is an Attestation Result, and the immigration desk is a Relying Party.

In this model, an Attester conveys Evidence to a Verifier, which compares the Evidence against its appraisal policy. The Verifier then gives back an Attestation Result which the Attester treats as opaque data.

The Attester does not consume the Attestation Result, but might cache it. The Attester can then present the Attestation Result (and possibly additional Claims) to a Relying Party, which then compares this information against its own appraisal policy. The Attester may also present the same Attestation Result to other Relying Parties.

Three ways in which the process may fail include:

* First, the Verifier may not issue a positive Attestation Result due to the Evidence not passing the Appraisal Policy for Evidence.

* The second way in which the process may fail is when the Attestation Result is examined by the Relying Party, and based upon the Appraisal Policy for Attestation Results, the result does not pass the policy.

* The third way is when the Verifier is unreachable or unavailable.

As with any other information needed by the Relying Party to make an authorization decision, an Attestation Result can be carried in a resource access protocol between the Attester and Relying Party. In this model the details of the resource access protocol constrain the serialization format of the Attestation Result. The format of the Evidence on the other hand is only constrained by the Attester-Verifier remote attestation protocol. This implies that interoperability and standardization is more relevant for Attestation Results than it is for Evidence.

```
      .------------.
      |            | Compare Evidence
      |  Verifier  | against appraisal policy
      |            |
      '--------+---'
          ^    |
 Evidence |    | Attestation
          |    | Result
          |    v
      .---+--------.              .-------------.
      |            +------------->|             | Compare Attestation
      |  Attester  | Attestation  |   Relying   | Result against
      |            | Result       |    Party    | appraisal policy
      '------------'              '-------------'
```

Figure 5: Passport Model

5.2. Background-Check Model

The background-check model is so named because of the resemblance of how employers and volunteer organizations perform background checks. When a prospective employee provides Claims about education or previous experience, the employer will contact the respective institutions or former employers to validate the Claim. Volunteer organizations often perform police background checks on volunteers in order to determine the volunteer's trustworthiness. Thus, in this analogy, a prospective volunteer is an Attester, the organization is the Relying Party, and the organization that issues a report is a Verifier.

In this model, an Attester conveys Evidence to a Relying Party, which treats it as opaque and simply forwards it on to a Verifier. The Verifier compares the Evidence against its appraisal policy, and returns an Attestation Result to the Relying Party. The Relying Party then compares the Attestation Result against its own appraisal policy.

The resource access protocol between the Attester and Relying Party includes Evidence rather than an Attestation Result, but that Evidence is not processed by the Relying Party.

Since the Evidence is merely forwarded on to a trusted Verifier, any serialization format can be used for Evidence because the Relying Party does not need a parser for it. The only requirement is that the Evidence can be _encapsulated in_ the format required by the resource access protocol between the Attester and Relying Party.

However, like in the Passport model, an Attestation Result is still consumed by the Relying Party. Code footprint and attack surface area can be minimized by using a serialization format for which the Relying Party already needs a parser to support the protocol between the Attester and Relying Party, which may be an existing standard or widely deployed resource access protocol. Such minimization is especially important if the Relying Party is a constrained node.

```
                                  .-------------.
                                  |             | Compare Evidence
                                  |   Verifier  | against appraisal
                                  |             | policy
                                  '--------+----'
                                       ^   |
                              Evidence |   | Attestation
                                       |   | Result
                                       |   v
      .------------.              .----|--------.
      |            +-------------->|---'         | Compare Attestation
      |  Attester  |   Evidence   |     Relying | Result against
      |            |              |      Party  | appraisal policy
      '------------'              '-------------'
```

Figure 6: Background-Check Model

5.3. Combinations

One variation of the background-check model is where the Relying Party and the Verifier are on the same machine, performing both functions together. In this case, there is no need for a protocol between the two.

It is also worth pointing out that the choice of model depends on the use case, and that different Relying Parties may use different topological patterns.

The same device may need to create Evidence for different Relying Parties and/or different use cases. For instance, it would use one model to provide Evidence to a network infrastructure device to gain access to the network, and the other model to provide Evidence to a server holding confidential data to gain access to that data. As such, both models may simultaneously be in use by the same device.

Figure 7 shows another example of a combination where Relying Party 1 uses the passport model, whereas Relying Party 2 uses an extension of the background-check model. Specifically, in addition to the basic functionality shown in Figure 6, Relying Party 2 actually provides the Attestation Result back to the Attester, allowing the Attester to use it with other Relying Parties. This is the model that the Trusted Application Manager plans to support in the TEEP architecture [I-D.ietf-teep-architecture].

```
        .-------------.
        |             | Compare Evidence
        |   Verifier  | against appraisal policy
        |             |
        '--------+----'
             ^   |
   Evidence  |   | Attestation
             |   | Result
             |   v
        .----+--------.
        |             | Compare
        |   Relying   | Attestation Result
        |   Party 2   | against appraisal policy
        '--------+----'
             ^   |
   Evidence  |   | Attestation
             |   | Result
             |   v
        .----+--------.               .-------------.
        |            +-------------->|             | Compare Attestation
        |   Attester |  Attestation  |   Relying   | Result against
        |            |     Result    |   Party 1   | appraisal policy
        '-------------'               '-------------'
```

Figure 7: Example Combination

6. Roles and Entities

An entity in the RATS architecture includes at least one of the roles defined in this document.

An entity can aggregate more than one role into itself, such as being both a Verifier and a Relying Party, or being both a Reference Value Provider and an Endorser. As such, any conceptual messages (see Section 8 for more discussion) originating from such roles might also be combined. For example, Reference Values might be conveyed as part of an appraisal policy if the Verifier Owner and Reference Value Provider roles are combined. Similarly, Reference Values might be conveyed as part of an Endorsement if the Endorser and Reference Value Provider roles are combined.

Interactions between roles aggregated into the same entity do not necessarily use the Internet Protocol. Such interactions might use a loopback device or other IP-based communication between separate environments, but they do not have to. Alternative channels to convey conceptual messages include function calls, sockets, GPIO interfaces, local busses, or hypervisor calls. This type of conveyance is typically found in composite devices. Most importantly, these conveyance methods are out-of-scope of RATS, but they are presumed to exist in order to convey conceptual messages appropriately between roles.

In essence, an entity that combines more than one role creates and consumes the corresponding conceptual messages as defined in this document.

7. Trust Model

7.1. Relying Party

This document covers scenarios for which a Relying Party trusts a Verifier that can appraise the trustworthiness of information about an Attester. Such trust is expressed by storing one or more "trust anchors" in a secure location known as a trust anchor store.

As defined in [RFC6024], "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative." The trust anchor may be a certificate or it may be a raw public key along with additional data if necessary such as its public key algorithm and parameters. In the context of this document, a trust anchor may also be a symmetric key, as in [TCG-DICE-SIBDA] or the symmetric mode described in [I-D.tschofenig-rats-psa-token].

Thus, trusting a Verifier might be expressed by having the Relying Party store the Verifier's key or certificate in its trust anchor store, or might be expressed by storing the public key or certificate of an entity (e.g., a Certificate Authority) that is in the Verifier's certificate path. For example, the Relying Party can verify that the Verifier is an expected one by out-of-band establishment of key material, combined with a protocol like TLS to communicate. There is an assumption that between the establishment of the trusted key material and the creation of the Evidence, that the Verifier has not been compromised.

For a stronger level of security, the Relying Party might require that the Verifier first provide information about itself that the Relying Party can use to assess the trustworthiness of the Verifier before accepting its Attestation Results. Such process would provide a stronger level of confidence in the correctness of the information provided, such as a belief that the authentic Verifier has not been compromised by malware.

For example, one explicit way for a Relying Party "A" to establish such confidence in the correctness of a Verifier "B", would be for B to first act as an Attester where A acts as a combined Verifier/ Relying Party. If A then accepts B as trustworthy, it can choose to accept B as a Verifier for other Attesters.

Similarly, the Relying Party also needs to trust the Relying Party Owner for providing its Appraisal Policy for Attestation Results, and in some scenarios the Relying Party might even require that the Relying Party Owner go through a remote attestation procedure with it before the Relying Party will accept an updated policy. This can be done similarly to how a Relying Party could establish trust in a Verifier as discussed above, i.e., verifying credentials against a trust anchor store and optionally requiring Attestation Results from the Relying Party Owner.

7.2. Attester

In some scenarios, Evidence might contain sensitive information such as Personally Identifiable Information (PII) or system identifiable information. Thus, an Attester must trust entities to which it conveys Evidence, to not reveal sensitive data to unauthorized parties. The Verifier might share this information with other authorized parties, according to a governing policy that address the handling of sensitive information (potentially included in Appraisal Policies for Evidence). In the background-check model, this Evidence may also be revealed to Relying Party(s).

When Evidence contains sensitive information, an Attester typically requires that a Verifier authenticates itself (e.g., at TLS session establishment) and might even request a remote attestation before the Attester sends the sensitive Evidence. This can be done by having the Attester first act as a Verifier/Relying Party, and the Verifier act as its own Attester, as discussed above.

7.3. Relying Party Owner

The Relying Party Owner might also require that the Relying Party first act as an Attester, providing Evidence that the Owner can appraise, before the Owner would give the Relying Party an updated policy that might contain sensitive information. In such a case, authentication or attestation in both directions might be needed, in which case typically one side’s Evidence must be considered safe to share with an untrusted entity, in order to bootstrap the sequence. See Section 11 for more discussion.

7.4. Verifier

The Verifier trusts (or more specifically, the Verifier’s security policy is written in a way that configures the Verifier to trust) a manufacturer, or the manufacturer’s hardware, so as to be able to appraise the trustworthiness of that manufacturer’s devices. Such trust is expressed by storing one or more trust anchors in the Verifier’s trust anchor store.

In a typical solution, a Verifier comes to trust an Attester indirectly by having an Endorser (such as a manufacturer) vouch for the Attester’s ability to securely generate Evidence through Endorsements (see Section 8.2). Endorsements might describe the ways in which the Attester resists attack, protects secrets and measures Target Environments. Consequently, the Endorser’s key material is stored in the Verifier’s trust anchor store so that Endorsements can be authenticated and used in the Verifier’s appraisal process.

In some solutions, a Verifier might be configured to directly trust an Attester by having the Verifier have the Attester's key material (rather than the Endorser's) in its trust anchor store.

Such direct trust must first be established at the time of trust anchor store configuration either by checking with an Endorser at that time, or by conducting a security analysis of the specific device. Having the Attester directly in the trust anchor store narrows the Verifier's trust to only specific devices rather than all devices the Endorser might vouch for, such as all devices manufactured by the same manufacturer in the case that the Endorser is a manufacturer.

Such narrowing is often important since physical possession of a device can also be used to conduct a number of attacks, and so a device in a physically secure environment (such as one's own premises) may be considered trusted whereas devices owned by others would not be. This often results in a desire to either have the owner run their own Endorser that would only endorse devices one owns, or to use Attesters directly in the trust anchor store. When there are many Attesters owned, the use of an Endorser enables better scalability.

That is, a Verifier might appraise the trustworthiness of an application component, operating system component, or service under the assumption that information provided about it by the lower-layer firmware or software is true. A stronger level of assurance of security comes when information can be vouched for by hardware or by ROM code, especially if such hardware is physically resistant to hardware tampering. In most cases, components that have to be vouched for via Endorsements because no Evidence is generated about them are referred to as roots of trust.

The manufacturer having arranged for an Attesting Environment to be provisioned with key material with which to sign Evidence, the Verifier is then provided with some way of verifying the signature on the Evidence. This may be in the form of an appropriate trust anchor, or the Verifier may be provided with a database of public keys (rather than certificates) or even carefully curated and secured lists of symmetric keys.

The nature of how the Verifier manages to validate the signatures produced by the Attester is critical to the secure operation of a remote attestation system, but is not the subject of standardization within this architecture.

A conveyance protocol that provides authentication and integrity protection can be used to convey Evidence that is otherwise unprotected (e.g., not signed). Appropriate conveyance of unprotected Evidence (e.g., [I-D.birkholz-rats-uccs]) relies on the following conveyance protocol's protection capabilities:

1. The key material used to authenticate and integrity protect the conveyance channel is trusted by the Verifier to speak for the Attesting Environment(s) that collected Claims about the Target Environment(s).

2. All unprotected Evidence that is conveyed is supplied exclusively by the Attesting Environment that has the key material that protects the conveyance channel

3. A trusted environment protects the conveyance channel's key material which may depend on other Attesting Environments with equivalent strength protections.

As illustrated in [I-D.birkholz-rats-uccs], an entity that receives unprotected Evidence via a trusted conveyance channel always takes on the responsibility of vouching for the Evidence's authenticity and freshness. If protected Evidence is generated, the Attester's Attesting Environments take on that responsibility. In cases where unprotected Evidence is processed by a Verifier, Relying Parties have to trust that the Verifier is capable of handling Evidence in a manner that preserves the Evidence's authenticity and freshness. Generating and conveying unprotected Evidence always creates significant risk and the benefits of that approach have to be carefully weighed against potential drawbacks.

See Section 12 for discussion on security strength.

7.5. Endorser, Reference Value Provider, and Verifier Owner

In some scenarios, the Endorser, Reference Value Provider, and Verifier Owner may need to trust the Verifier before giving the Endorsement, Reference Values, or appraisal policy to it. This can be done similarly to how a Relying Party might establish trust in a Verifier.

As discussed in Section 7.3, authentication or attestation in both directions might be needed, in which case typically one side's identity or Evidence must be considered safe to share with an untrusted entity, in order to bootstrap the sequence. See Section 11 for more discussion.

8. Conceptual Messages

Figure 1 illustrates the flow of conceptual messages between various roles. This section provides additional elaboration and implementation considerations. It is the responsibility of protocol specifications to define the actual data format and semantics of any relevant conceptual messages.

8.1. Evidence

Evidence is a set of Claims about the target environment that reveal operational status, health, configuration or construction that have security relevance. Evidence is appraised by a Verifier to establish its relevance, compliance, and timeliness. Claims need to be collected in a manner that is reliable such that a Target Environment cannot lie to the Attesting Environment about its trustworthiness properties. Evidence needs to be securely associated with the target environment so that the Verifier cannot be tricked into accepting Claims originating from a different environment (that may be more trustworthy). Evidence also must be protected from an active on-path attacker who may observe, change or misdirect Evidence as it travels from Attester to Verifier. The timeliness of Evidence can be captured using Claims that pinpoint the time or interval when changes in operational status, health, and so forth occur.

8.2. Endorsements

An Endorsement is a secure statement that some entity (e.g., a manufacturer) vouches for the integrity of the device's various capabilities such as claims collection, signing, launching code, transitioning to other environments, storing secrets, and more. For example, if the device's signing capability is in hardware, then an Endorsement might be a manufacturer certificate that signs a public key whose corresponding private key is only known inside the device's hardware. Thus, when Evidence and such an Endorsement are used together, an appraisal procedure can be conducted based on appraisal policies that may not be specific to the device instance, but merely specific to the manufacturer providing the Endorsement. For example, an appraisal policy might simply check that devices from a given manufacturer have information matching a set of Reference Values, or an appraisal policy might have a set of more complex logic on how to appraise the validity of information.

However, while an appraisal policy that treats all devices from a given manufacturer the same may be appropriate for some use cases, it would be inappropriate to use such an appraisal policy as the sole means of authorization for use cases that wish to constrain _which_ compliant devices are considered authorized for some purpose. For

example, an enterprise using remote attestation for Network Endpoint Assessment [RFC5209] may not wish to let every healthy laptop from the same manufacturer onto the network, but instead only want to let devices that it legally owns onto the network. Thus, an Endorsement may be helpful information in authenticating information about a device, but is not necessarily sufficient to authorize access to resources which may need device-specific information such as a public key for the device or component or user on the device.

8.3. Reference Values

Reference Values used in appraisal procedures come from a Reference Value Provider and are then used by the Verifier to compare to Evidence. Reference Values with matching Evidence produces acceptable Claims. Additionally, appraisal policy may play a role in determining the acceptance of Claims.

8.4. Attestation Results

Attestation Results are the input used by the Relying Party to decide the extent to which it will trust a particular Attester, and allow it to access some data or perform some operation.

Attestation Results may carry a boolean value indicating compliance or non-compliance with a Verifier's appraisal policy, or may carry a richer set of Claims about the Attester, against which the Relying Party applies its Appraisal Policy for Attestation Results.

The quality of the Attestation Results depends upon the ability of the Verifier to evaluate the Attester. Different Attesters have a different _Strength of Function_ [strengthoffunction], which results in the Attestation Results being qualitatively different in strength.

An Attestation Result that indicates non-compliance can be used by an Attester (in the passport model) or a Relying Party (in the background-check model) to indicate that the Attester should not be treated as authorized and may be in need of remediation. In some cases, it may even indicate that the Evidence itself cannot be authenticated as being correct.

By default, the Relying Party does not believe the Attester to be compliant. Upon receipt of an authentic Attestation Result and given the Appraisal Policy for Attestation Results is satisfied, the Attester is allowed to perform the prescribed actions or access. The simplest such appraisal policy might authorize granting the Attester full access or control over the resources guarded by the Relying Party. A more complex appraisal policy might involve using the information provided in the Attestation Result to compare against expected values, or to apply complex analysis of other information contained in the Attestation Result.

Thus, Attestation Results can contain detailed information about an Attester, which can include privacy sensitive information as discussed in Section 11. Unlike Evidence, which is often very device- and vendor-specific, Attestation Results can be vendor-neutral, if the Verifier has a way to generate vendor-agnostic information based on the appraisal of vendor-specific information in Evidence. This allows a Relying Party's appraisal policy to be simpler, potentially based on standard ways of expressing the information, while still allowing interoperability with heterogeneous devices.

Finally, whereas Evidence is signed by the device (or indirectly by a manufacturer, if Endorsements are used), Attestation Results are signed by a Verifier, allowing a Relying Party to only need a trust relationship with one entity, rather than a larger set of entities, for purposes of its appraisal policy.

8.5. Appraisal Policies

The Verifier, when appraising Evidence, or the Relying Party, when appraising Attestation Results, checks the values of matched Claims against constraints specified in its appraisal policy. Examples of such constraints checking include:

* comparison for equality against a Reference Value, or

* a check for being in a range bounded by Reference Values, or

* membership in a set of Reference Values, or

* a check against values in other Claims.

Upon completing all appraisal policy constraints, the remaining Claims are accepted as input toward determining Attestation Results, when appraising Evidence, or as input to a Relying Party, when appraising Attestation Results.

9. Claims Encoding Formats

The following diagram illustrates a relationship to which remote attestation is desired to be added:

```
.-------------.               .------------. Evaluate
|             +-------------->|            | request
|  Attester   |  Access some  |   Relying  | against
|             |    resource   |   Party    | security
'-------------'               '------------' policy
```

Figure 8: Typical Resource Access

In this diagram, the protocol between Attester and a Relying Party can be any new or existing protocol (e.g., HTTP(S), COAP(S), ROLIE [RFC8322], 802.1x, OPC UA [OPCUA], etc.), depending on the use case.

Typically, such protocols already have mechanisms for passing security information for authentication and authorization purposes. Common formats include JWTs [RFC7519], CWTs [RFC8392], and X.509 certificates.

Retrofitting already deployed protocols with remote attestation requires adding RATS conceptual messages to the existing data flows. This must be done in a way that does not degrade the security properties of the systems involved and should use native extension mechanisms provided by the underlying protocol. For example, if a TLS handshake is to be extended with remote attestation capabilities, attestation Evidence may be embedded in an ad-hoc X.509 certificate extension (e.g., [TCG-DICE]), or into a new TLS Certificate Type (e.g., [I-D.tschofenig-tls-cwt]).

Especially for constrained nodes there is a desire to minimize the amount of parsing code needed in a Relying Party, in order to both minimize footprint and to minimize the attack surface. While it would be possible to embed a CWT inside a JWT, or a JWT inside an X.509 extension, etc., there is a desire to encode the information natively in a format that is already supported by the Relying Party.

This motivates having a common "information model" that describes the set of remote attestation related information in an encoding-agnostic way, and allowing multiple encoding formats (CWT, JWT, X.509, etc.) that encode the same information into the Claims format needed by the Relying Party.

The following diagram illustrates that Evidence and Attestation Results might be expressed via multiple potential encoding formats, so that they can be conveyed by various existing protocols. It also

motivates why the Verifier might also be responsible for accepting Evidence that encodes Claims in one format, while issuing Attestation Results that encode Claims in a different format.

```
                 Evidence           Attestation Results
.--------------.   CWT                   CWT   .-------------------.
|  Attester-A  +-----------.        .---------->|  Relying Party V  |
'--------------'           |        |           `-------------------'
                           v        |
.--------------.   JWT   .---------+--.   JWT   .-------------------.
|  Attester-B  +-------->|            +-------->|  Relying Party W  |
'--------------'         |            |         `-------------------'
                         |            |
.--------------.  X.509  |            |  X.509  .-------------------.
|  Attester-C  +-------->|  Verifier  +-------->|  Relying Party X  |
'--------------'         |            |         `-------------------'
                         |            |
.--------------.   TPM   |            |   TPM   .-------------------.
|  Attester-D  +-------->|            +-------->|  Relying Party Y  |
'--------------'         '---------+--'         `-------------------'
                                   ^        |
.--------------.  other            |        |     other  .-------------------.
|  Attester-E  +-----------'        '---------->|  Relying Party Z  |
'--------------'                                `-------------------'
```

Figure 9: Multiple Attesters and Relying Parties with Different Formats

10. Freshness

A Verifier or Relying Party might need to learn the point in time (i.e., the "epoch") an Evidence or Attestation Result has been produced. This is essential in deciding whether the included Claims can be considered fresh, meaning they still reflect the latest state of the Attester, and that any Attestation Result was generated using the latest Appraisal Policy for Evidence.

This section provides a number of details. It does not however define any protocol formats, the interactions shown are abstract. This section is intended for those creating protocols and solutions to understand the options available to ensure freshness. The way in which freshness is provisioned in a protocol is an architectural decision. Provisioning of freshness has an impact on the number of needed round trips in a protocol, and therefore must be made very early in the design. Different decisions will have significant impacts on resulting interoperability, which is why this section goes into sufficient detail such that choices in freshness will be compatible across interacting protocols, such as depicted in Figure 9.

Freshness is assessed based on the Appraisal Policy for Evidence or Attestation Results that compares the estimated epoch against an "expiry" threshold defined locally to that policy. There is, however, always a race condition possible in that the state of the Attester, and the appraisal policies might change immediately after the Evidence or Attestation Result was generated. The goal is merely to narrow their recentness to something the Verifier (for Evidence) or Relying Party (for Attestation Result) is willing to accept. Some flexibility on the freshness requirement is a key component for enabling caching and reuse of both Evidence and Attestation Results, which is especially valuable in cases where their computation uses a substantial part of the resource budget (e.g., energy in constrained devices).

There are three common approaches for determining the epoch of Evidence or an Attestation Result.

10.1. Explicit Timekeeping using Synchronized Clocks

The first approach is to rely on synchronized and trustworthy clocks, and include a signed timestamp (see [I-D.birkholz-rats-tuda]) along with the Claims in the Evidence or Attestation Result. Timestamps can also be added on a per-Claim basis to distinguish the time of generation of Evidence or Attestation Result from the time that a specific Claim was generated. The clock’s trustworthiness can generally be established via Endorsements and typically requires additional Claims about the signer’s time synchronization mechanism.

In some use cases, however, a trustworthy clock might not be available. For example, in many Trusted Execution Environments (TEEs) today, a clock is only available outside the TEE and so cannot be trusted by the TEE.

10.2. Implicit Timekeeping using Nonces

A second approach places the onus of timekeeping solely on the Verifier (for Evidence) or the Relying Party (for Attestation Results), and might be suitable, for example, in case the Attester does not have a trustworthy clock or time synchronization is otherwise impaired. In this approach, a non-predictable nonce is sent by the appraising entity, and the nonce is then signed and included along with the Claims in the Evidence or Attestation Result. After checking that the sent and received nonces are the same, the appraising entity knows that the Claims were signed after the nonce was generated. This allows associating a "rough" epoch to the Evidence or Attestation Result. In this case the epoch is said to be rough because:

* The epoch applies to the entire Claim set instead of a more granular association, and

* The time between the creation of Claims and the collection of Claims is indistinguishable.

10.3. Implicit Timekeeping using Epoch IDs

A third approach relies on having epoch identifiers (or "IDs") periodically sent to both the sender and receiver of Evidence or Attestation Results by some "Epoch ID Distributor".

Epoch IDs are different from nonces as they can be used more than once and can even be used by more than one entity at the same time. Epoch IDs are different from timestamps as they do not have to convey information about a point in time, i.e., they are not necessarily monotonically increasing integers.

Like the nonce approach, this allows associating a "rough" epoch without requiring a trustworthy clock or time synchronization in order to generate or appraise the freshness of Evidence or Attestation Results. Only the Epoch ID Distributor requires access to a clock so it can periodically send new epoch IDs.

The most recent epoch ID is included in the produced Evidence or Attestation Results, and the appraising entity can compare the epoch ID in received Evidence or Attestation Results against the latest epoch ID it received from the Epoch ID Distributor to determine if it is within the current epoch. An actual solution also needs to take into account race conditions when transitioning to a new epoch, such as by using a counter signed by the Epoch ID Distributor as the epoch ID, or by including both the current and previous epoch IDs in messages and/or checks, by requiring retries in case of mismatching epoch IDs, or by buffering incoming messages that might be associated with an epoch ID that the receiver has not yet obtained.

More generally, in order to prevent an appraising entity from generating false negatives (e.g., discarding Evidence that is deemed stale even if it is not), the appraising entity should keep an "epoch window" consisting of the most recently received epoch IDs. The depth of such epoch window is directly proportional to the maximum network propagation delay between the first to receive the epoch ID and the last to receive the epoch ID, and it is inversely proportional to the epoch duration. The appraising entity shall compare the epoch ID carried in the received Evidence or Attestation Result with the epoch IDs in its epoch window to find a suitable match.

Whereas the nonce approach typically requires the appraising entity to keep state for each nonce generated, the epoch ID approach minimizes the state kept to be independent of the number of Attesters or Verifiers from which it expects to receive Evidence or Attestation Results, as long as all use the same Epoch ID Distributor.

10.4. Discussion

Implicit and explicit timekeeping can be combined into hybrid mechanisms. For example, if clocks exist within the Attesting Environment and are considered trustworthy (tamper-proof) but are not synchronized, a nonce-based exchange may be used to determine the (relative) time offset between the involved peers, followed by any number of timestamp based exchanges.

It is important to note that the actual values in Claims might have been generated long before the Claims are signed. If so, it is the signer’s responsibility to ensure that the values are still correct when they are signed. For example, values generated at boot time might have been saved to secure storage until network connectivity is established to the remote Verifier and a nonce is obtained.

A more detailed discussion with examples appears in Appendix A.

For a discussion on the security of epoch IDs see Section 12.3.

11. Privacy Considerations

The conveyance of Evidence and the resulting Attestation Results reveal a great deal of information about the internal state of a device as well as potentially any users of the device.

In many cases, the whole point of attestation procedures is to provide reliable information about the type of the device and the firmware/software that the device is running.

This information might be particularly interesting to many attackers. For example, knowing that a device is running a weak version of firmware provides a way to aim attacks better.

In some circumstances, if an attacker can become aware of Endorsements, Reference Values, or appraisal policies, it could potentially provide an attacker with insight into defensive mitigations. It is recommended that attention be paid to confidentiality of such information.

Additionally, many Claims in Evidence, many Claims in Attestation Results, and appraisal policies potentially contain Personally Identifying Information (PII) depending on the end-to-end use case of the remote attestation procedure. Remote attestation that includes containers and applications, e.g., a blood pressure monitor, may further reveal details about specific systems or users.

In some cases, an attacker may be able to make inferences about the contents of Evidence from the resulting effects or timing of the processing. For example, an attacker might be able to infer the value of specific Claims if it knew that only certain values were accepted by the Relying Party.

Conceptual messages (see Section 8) carrying sensitive or confidential information are expected to be integrity protected (i.e., either via signing or a secure channel) and optionally might be confidentiality protected via encryption. If there isn’t confidentiality protection of conceptual messages themselves, the underlying conveyance protocol should provide these protections.

As Evidence might contain sensitive or confidential information, Attesters are responsible for only sending such Evidence to trusted Verifiers. Some Attesters might want a stronger level of assurance of the trustworthiness of a Verifier before sending Evidence to it. In such cases, an Attester can first act as a Relying Party and ask for the Verifier's own Attestation Result, and appraising it just as a Relying Party would appraise an Attestation Result for any other purpose.

Another approach to deal with Evidence is to remove PII from the Evidence while still being able to verify that the Attester is one of a large set. This approach is often called "Direct Anonymous Attestation". See [CCC-DeepDive] section 6.2 and [I-D.ietf-rats-daa] for more discussion.

12. Security Considerations

This document provides an architecture for doing remote attestation. No specific wire protocol is documented here. Without a specific proposal to compare against, it is impossible to know if the security threats listed below have been mitigated well.

The security considerations below should be read as being essentially requirements against realizations of the RATS Architecture. Some threats apply to protocols, some are against implementations (code), and some threats are against physical infrastructure (such as factories).

The fundamental purpose of the RATS architecture is to allow a Relying Party to establish a basis for trusting the Attester.

12.1. Attester and Attestation Key Protection

Implementers need to pay close attention to the protection of the Attester and the manufacturing processes for provisioning attestation key material. If either of these are compromised, intended levels of assurance for RATS are compromised because attackers can forge Evidence or manipulate the Attesting Environment. For example, a Target Environment should not be able to tamper with the Attesting Environment that measures it, by isolating the two environments from each other in some way.

Remote attestation applies to use cases with a range of security requirements, so the protections discussed here range from low to high security where low security may be limited to application or process isolation by the device's operating system, and high security may involve specialized hardware to defend against physical attacks on a chip.

12.1.1. On-Device Attester and Key Protection

It is assumed that an Attesting Environment is sufficiently isolated from the Target Environment it collects Claims about and that it signs the resulting Claims set with an attestation key, so that the Target Environment cannot forge Evidence about itself. Such an isolated environment might be provided by a process, a dedicated chip, a TEE, a virtual machine, or another secure mode of operation. The Attesting Environment must be protected from unauthorized modification to ensure it behaves correctly. Confidentiality protection of the Attesting Environment's signing key is vital so it cannot be misused to forge Evidence.

In many cases the user or owner of a device that includes the role of Attester must not be able to modify or extract keys from the Attesting Environments, to prevent creating forged Evidence. Some common examples include the user of a mobile phone or FIDO authenticator.

Measures for a minimally protected system might include process or application isolation provided by a high-level operating system, and restricted access to root or system privileges. In contrast, For really simple single-use devices that don't use a protected mode operating system, like a Bluetooth speaker, the only factual isolation might be the sturdy housing of the device.

Measures for a moderately protected system could include a special restricted operating environment, such as a TEE. In this case, only security-oriented software has access to the Attester and key material.

Measures for a highly protected system could include specialized hardware that is used to provide protection against chip decapping attacks, power supply and clock glitching, faulting injection and RF and power side channel attacks.

12.1.2. Attestation Key Provisioning Processes

Attestation key provisioning is the process that occurs in the factory or elsewhere to establish signing key material on the device and the validation key material off the device. Sometimes this procedure is referred to as personalization or customization.

When generating keys off-device in the factory or in the device, the use of a Cryptographically Strong Sequence ([RFC4086], Section 6.2) needs consideration.

12.1.2.1. Off-Device Key Generation

One way to provision key material is to first generate it external to the device and then copy the key onto the device. In this case, confidentiality protection of the generator, as well as for the path over which the key is provisioned, is necessary. The manufacturer needs to take care to protect corresponding key material with measures appropriate for its value.

The degree of protection afforded to this key material can vary by the intended function of the device and the specific practices of the device manufacturer or integrator. The confidentiality protection is fundamentally based upon some amount of physical protection: while encryption is often used to provide confidentiality when a key is conveyed across a factory, where the attestation key is created or applied, it must be available in an unencrypted form. The physical protection can therefore vary from situations where the key is unencrypted only within carefully controlled secure enclaves within silicon, to situations where an entire facility is considered secure, by the simple means of locked doors and limited access.

The cryptography that is used to enable confidentiality protection of the attestation key comes with its own requirements to be secured. This results in recursive problems, as the key material used to provision attestation keys must again somehow have been provisioned securely beforehand (requiring an additional level of protection, and so on).

Commonly, a combination of some physical security measures and some cryptographic measures are used to establish confidentiality protection.

12.1.2.2. On-Device Key Generation

When key material is generated within a device and the secret part of it never leaves the device, then the problem may lessen. For public-key cryptography, it is, by definition, not necessary to maintain confidentiality of the public key: however integrity of the chain of custody of the public key is necessary in order to avoid attacks where an attacker is able to get a key they control endorsed.

To summarize: attestation key provisioning must ensure that only valid attestation key material is established in Attesters.

12.2. Conceptual Message Protection

Any solution that conveys information in any conceptual message (see Section 8) must support end-to-end integrity protection and replay attack prevention, and often also needs to support additional security properties, including:

* end-to-end encryption,

* denial of service protection,

* authentication,

* auditing,

* fine grained access controls, and

* logging.

Section 10 discusses ways in which freshness can be used in this architecture to protect against replay attacks.

To assess the security provided by a particular appraisal policy, it is important to understand the strength of the root of trust, e.g., whether it is mutable software, or firmware that is read-only after boot, or immutable hardware/ROM.

It is also important that the appraisal policy was itself obtained securely. If an attacker can configure or modify appraisal policies, Endorsements or Reference Values for a Relying Party or for a Verifier, then integrity of the process is compromised.

Security protections in RATS may be applied at different layers, whether by a conveyance protocol, or an information encoding format. This architecture expects conceptual messages to be end-to-end protected based on the role interaction context. For example, if an Attester produces Evidence that is relayed through some other entity that doesn’t implement the Attester or the intended Verifier roles, then the relaying entity should not expect to have access to the Evidence.

The RATS architecture allows for an entity to function in multiple roles (Section 6) and for composite devices (Section 3.3). Implementers need to evaluate their designs to ensure that the assumed security properties of the individual components and roles still hold despite the lack of separation, and that emergent risk is not introduced. The specifics of this evaluation will depend on the implementation and the use case and hence is out of scope for this

document. Isolation mechanisms in software or hardware that separate Attesting Environments and Target Environments Section 3.1 can support an implementer's evaluation and resulting design decisions.

12.3. Epoch ID-based Attestation

Epoch IDs, described in Section 10.3, can be tampered with, replayed, dropped, delayed, and reordered by an attacker.

An attacker could be either external or belong to the distribution group, for example, if one of the Attester entities have been compromised.

An attacker who is able to tamper with epoch IDs can potentially lock all the participants in a certain epoch of choice forever, effectively freezing time. This is problematic since it destroys the ability to ascertain freshness of Evidence and Attestation Results.

To mitigate this threat, the transport should be at least integrity protected and provide origin authentication.

Selective dropping of epoch IDs is equivalent to pinning the victim node to a past epoch. An attacker could drop epoch IDs to only some entities and not others, which will typically result in a denial of service due to the permanent staleness of the Attestation Result or Evidence.

Delaying or reordering epoch IDs is equivalent to manipulating the victim's timeline at will. This ability could be used by a malicious actor (e.g., a compromised router) to mount a confusion attack where, for example, a Verifier is tricked into accepting Evidence coming from a past epoch as fresh, while in the meantime the Attester has been compromised.

Reordering and dropping attacks are mitigated if the transport provides the ability to detect reordering and drop. However, the delay attack described above can't be thwarted in this manner.

12.4. Trust Anchor Protection

As noted in Section 7, Verifiers and Relying Parties have trust anchor stores that must be secured. [RFC6024] contains more discussion of trust anchor store requirements for protecting public keys. Section 6 of [NIST-800-57-p1] contains a comprehensive treatment of the topic, including the protection of symmetric key material. Specifically, a trust anchor store must resist modification against unauthorized insertion, deletion, and modification. Additionally, if the trust anchor is a symmetric key,

the trust anchor store must not allow unauthorized read.

If certificates are used as trust anchors, Verifiers and Relying Parties are also responsible for validating the entire certificate path up to the trust anchor, which includes checking for certificate revocation. For an example of such a proceedure see Section 6 of [RFC5280].

13. IANA Considerations

This document does not require any actions by IANA.

14. Acknowledgments

Special thanks go to Joerg Borchert, Nancy Cam-Winget, Jessica Fitzgerald-McKay, Diego Lopez, Laurence Lundblade, Paul Rowe, Hannes Tschofenig, Frank Xia, and David Wooten.

15. Notable Contributions

Thomas Hardjono created initial versions of the terminology section in collaboration with Ned Smith. Eric Voit provided the conceptual separation between Attestation Provision Flows and Attestation Evidence Flows. Monty Wisemen created the content structure of the first three architecture drafts. Carsten Bormann provided many of the motivational building blocks with respect to the Internet Threat Model.

Peter Loscocco contributed critical review feedback as part of the weekly design team meetings that added precision and depth to several sections.

16. References

16.1. Normative References

[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>.

[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>.

[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>.

16.2. Informative References

[CCC-DeepDive]
Confidential Computing Consortium, "Confidential Computing Deep Dive", n.d., <https://confidentialcomputing.io/whitepaper-02-latest>.

[CTAP] FIDO Alliance, "Client to Authenticator Protocol", n.d., <https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html>.

[I-D.birkholz-rats-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-07, 10 July 2022, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-tuda-07>.

[I-D.birkholz-rats-uccs]
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-birkholz-rats-uccs-03, 8 March 2021, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-uccs-03>.

[I-D.ietf-rats-daa]
Birkholz, H., Newton, C., Chen, L., and D. Thaler, "Direct Anonymous Attestation for the Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-daa-02, 7 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-daa-02>.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. M. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-18, 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-teep-architecture-18>.

[I-D.tschofenig-rats-psa-token]
Tschofenig, H., Frost, S., Brossard, M., Shaw, A. L., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", Work in Progress, Internet-Draft, draft-tschofenig-rats-psa-token-10, 6 September 2022, <https://datatracker.ietf.org/doc/html/draft-tschofenig-rats-psa-token-10>.

[I-D.tschofenig-tls-cwt]
Tschofenig, H. and M. Brossard, "Using CBOR Web Tokens (CWTs) in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-tschofenig-tls-cwt-02, 13 July 2020, <https://datatracker.ietf.org/doc/html/draft-tschofenig-tls-cwt-02>.

[NIST-800-57-p1]
Barker, E., "Recommendation for Key Managemement: Part 1 - General", May 2020, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>.

[OPCUA] OPC Foundation, "OPC Unified Architecture Specification, Part 2: Security Model, Release 1.03", OPC 10000-2 , 25 November 2015, <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/rfc/rfc4949>.

[RFC5209] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <https://www.rfc-editor.org/rfc/rfc5209>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC8322] Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, <https://www.rfc-editor.org/rfc/rfc8322>.

[strengthoffunction] NISC, "Strength of Function", n.d., <https://csrc.nist.gov/glossary/term/strength_of_function>.

[TCG-DICE] Trusted Computing Group, "DICE Certificate Profiles", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Certificate-Profiles-r01_3june2020-1.pdf>.

[TCG-DICE-SIBDA] Trusted Computing Group, "Symmetric Identity Based Device Attestation for DICE", 24 July 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_DICE_SymIDAttest_v1_r0p94_pubrev.pdf>.

[TCGarch] Trusted Computing Group, "Trusted Platform Module Library - Part 1: Architecture", 8 November 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf>.

[WebAuthN] W3C, "Web Authentication: An API for accessing Public Key Credentials", n.d., <https://www.w3.org/TR/webauthn-1/>.

Appendix A. Time Considerations

Section 10 discussed various issues and requirements around freshness of evidence, and summarized three approaches that might be used by different solutions to address them. This appendix provides more details with examples to help illustrate potential approaches, to inform those creating specific solutions.

The table below defines a number of relevant events, with an ID that is used in subsequent diagrams. The times of said events might be defined in terms of an absolute clock time, such as the Coordinated Universal Time timescale, or might be defined relative to some other timestamp or timeticks counter, such as a clock resetting its epoch each time it is powered on.

ID	Event	Explanation of event
VG	Value generated	A value to appear in a Claim was created. In some cases, a value may have technically existed before an Attester became aware of it but the Attester might have no idea how long it has had that value. In such a case, the Value created time is the time at which the Claim containing the copy of the value was created.
NS	Nonce sent	A nonce not predictable to an Attester (recentness & uniqueness) is sent to an Attester.
NR	Nonce relayed	A nonce is relayed to an Attester by another entity.
IR	Epoch ID received	An epoch ID is successfully received and processed by an entity.
EG	Evidence generation	An Attester creates Evidence from collected Claims.
ER	Evidence relayed	A Relying Party relays Evidence to a Verifier.
RG	Result generation	A Verifier appraises Evidence and generates an Attestation Result.
RR	Result relayed	A Relying Party relays an Attestation Result to a Relying Party.
RA	Result appraised	The Relying Party appraises Attestation Results.
OP	Operation performed	The Relying Party performs some operation requested by the Attester via a resource access protocol as depicted in Figure 8, e.g., across a session created earlier at time(RA).
RX	Result expiry	An Attestation Result should no longer be accepted, according to the Verifier that generated it.

Table 1

Using the table above, a number of hypothetical examples of how a solution might be built are illustrated below. This list is not intended to be complete, but is just representative enough to highlight various timing considerations.

All times are relative to the local clocks, indicated by an "_a" (Attester), "_v" (Verifier), or "_r" (Relying Party) suffix.

Times with an appended Prime (') indicate a second instance of the same event.

How and if clocks are synchronized depends upon the model.

In the figures below, curly braces indicate containment. For example, the notation Evidence{foo} indicates that 'foo' is contained in the Evidence and is thus covered by its signature.

A.1. Example 1: Timestamp-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party, which depends on using a secure clock synchronization mechanism. As a result, the receiver of a conceptual message containing a timestamp can directly compare it to its own clock and timestamps.

```
      .----------.                      .----------.  .---------------.
      | Attester |                      | Verifier |  | Relying Party |
      '----+-----'                      '-----+----'  '-------+-------'
           |                                  |                |
        time(VG_a)                            |                |
           |                                  |                |
           ~                                  ~                ~
           |                                  |                |
        time(EG_a)                            |                |
           |                                  |                |
           +------Evidence{time(EG_a)}------>|                |
           |                                  |                |
           |                              time(RG_v)           |
           |                                  |                |
           |<-----Attestation Result---------+                |
           |      {time(RG_v),time(RX_v)}     |                |
           ~                                                   ~
           |                                                   |
           +--Attestation Result{time(RG_v),time(RX_v)}--> time(RA_r)
           |                                                   |
           ~                                                   ~
           |                                                   |
           |                                              time(OP_r)
```

The Verifier can check whether the Evidence is fresh when appraising it at time(RG_v) by checking time(RG_v) - time(EG_a) < Threshold, where the Verifier's threshold is large enough to account for the maximum permitted clock skew between the Verifier and the Attester.

If time(VG_a) is also included in the Evidence along with the Claim value generated at that time, and the Verifier decides that it can trust the time(VG_a) value, the Verifier can also determine whether the Claim value is recent by checking time(RG_v) - time(VG_a) < Threshold. The threshold is decided by the Appraisal Policy for Evidence, and again needs to take into account the maximum permitted clock skew between the Verifier and the Attester.

The Attester does not consume the Attestation Result, but might cache it.

The Relying Party can check whether the Attestation Result is fresh when appraising it at time(RA_r) by checking time(RA_r) - time(RG_v) < Threshold, where the Relying Party's threshold is large enough to account for the maximum permitted clock skew between the Relying Party and the Verifier. The result might then be used for some time (e.g., throughout the lifetime of a connection established at time(RA_r)). The Relying Party must be careful, however, to not allow continued use beyond the period for which it deems the

Attestation Result to remain fresh enough. Thus, it might allow use (at time(OP_r)) as long as time(OP_r) - time(RG_v) < Threshold. However, if the Attestation Result contains an expiry time time(RX_v) then it could explicitly check time(OP_r) < time(RX_v).

A.2. Example 2: Nonce-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses nonces instead of timestamps. Compared to the timestamp-based example, it requires an extra round trip to retrieve a nonce, and requires that the Verifier and Relying Party track state to remember the nonce for some period of time.

The advantage is that it does not require that any clocks are synchronized. As a result, the receiver of a conceptual message containing a timestamp cannot directly compare it to its own clock or timestamps. Thus, we use a suffix ("a" for Attester, "v" for Verifier, and "r" for Relying Party) on the IDs below indicating which clock generated them, since times from different clocks cannot be compared. Only the delta between two events from the sender can be used by the receiver.

```
      .----------.                      .----------.  .---------------.
      | Attester |                      | Verifier |  | Relying Party |
      '----+-----'                      '-----+----'  '-------+-------'
           |                                  |                |
        time(VG_a)                            |                |
           |                                  |                |
           ~                                  ~                ~
           |                                  |                |
           |<--Nonce1---------------------time(NS_v)          |
           |                                  |                |
        time(EG_a)                            |                |
           |                                  |                |
           +---Evidence--------------------->|                |
           | {Nonce1, time(EG_a)-time(VG_a)} |                |
           |                                  |                |
           |                              time(RG_v)          |
           |                                  |                |
           |<--Attestation Result------------+                |
           |   {time(RX_v)-time(RG_v)}        |                |
           ~                                                   ~
           |                                                   |
           |<--Nonce2-----------------------------------time(NS_r)
           |                                                   |
        time(RR_a)                                             |
           |                                                   |
           +--[Attestation Result{time(RX_v)-time(RG_v)}, -->|time(RA_r)
           |        Nonce2, time(RR_a)-time(EG_a)]             |
           |                                                   |
           ~                                                   ~
           |                                                   |
           |                                             time(OP_r)
```

In this example solution, the Verifier can check whether the Evidence is fresh at time(RG_v) by verifying that time(RG_v)-time(NS_v) < Threshold.

The Verifier cannot, however, simply rely on a Nonce to determine whether the value of a Claim is recent, since the Claim value might have been generated long before the nonce was sent by the Verifier. However, if the Verifier decides that the Attester can be trusted to correctly provide the delta time(EG_a)-time(VG_a), then it can determine recency by checking time(RG_v)-time(NS_v) + time(EG_a)-time(VG_a) < Threshold.

Similarly if, based on an Attestation Result from a Verifier it trusts, the Relying Party decides that the Attester can be trusted to correctly provide time deltas, then it can determine whether the Attestation Result is fresh by checking time(OP_r)-time(NS_r) +

time(RR_a)-time(EG_a) < Threshold. Although the Nonce2 and time(RR_a)-time(EG_a) values cannot be inside the Attestation Result, they might be signed by the Attester such that the Attestation Result vouches for the Attester's signing capability.

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of time(RX_v)-time(RG_v), then the Relying Party can check time(OP_r)-time(NS_r) < time(RX_v)-time(RG_v).

A.3. Example 3: Epoch ID-based Passport Model Example

The example in Figure 10 illustrates a hypothetical Passport Model solution that uses epoch IDs instead of nonces or timestamps.

The Epoch ID Distributor broadcasts epoch ID I which starts a new epoch E for a protocol participant upon reception at time(IR).

The Attester generates Evidence incorporating epoch ID I and conveys it to the Verifier.

The Verifier appraises that the received epoch ID I is "fresh" according to the definition provided in Section 10.3 whereby retries are required in the case of mismatching epoch IDs, and generates an Attestation Result. The Attestation Result is conveyed to the Attester.

After the transmission of epoch ID I' a new epoch E' is established when I' is received by each protocol participant. The Attester relays the Attestation Result obtained during epoch E (associated with epoch ID I) to the Relying Party using the epoch ID for the current epoch I'. If the Relying Party had not yet received I', then the Attestation Result would be rejected, but in this example, it is received.

In the illustrated scenario, the epoch ID for relaying an Attestation Result to the Relying Party is current, while a previous epoch ID was used to generate Verifier evaluated evidence. This indicates that at least one epoch transition has occurred, and the Attestation Results may only be as fresh as the previous epoch. If the Relying Party remembers the previous epoch ID I during an epoch window as discussed in Section 10.3, and the message is received during that window, the Attestation Result is accepted as fresh, and otherwise it is rejected as stale.

```
                  .-------------.
.----------.      | Epoch ID    |    .----------.  .---------------.
| Attester |      | Distributor |    | Verifier |  | Relying Party |
'----+-----'      '------+------'    '-----+----'  '-------+-------'
     |                   |                 |               |
  time(VG_a)             |                 |               |
     |                   |                 |               |
     ~                   |                 ~               ~
     |                   |                 |               |
  time(IR_a) <-----I--o--I------> time(IR_v) ---> time(IR_r)
     |                   |                 |               |
  time(EG_a)             |                 |               |
     |                   |                 |               |
     +---Evidence--------------------->|               |
     |   {I,time(EG_a)-time(VG_a)}     |               |
     |                   |             |               |
     |                   |           time(RG_v)          |
     |                   |             |               |
     |<--Attestation Result------------+               |
     |   {I,time(RX_v)-time(RG_v)}     |               |
     |                   |             |               |
  time(IR'_a) <----I'-o--I' ----> time(IR'_v) --> time(IR'_r)
     |                   |             |               |
     +---[Attestation Result--------------------> time(RA_r)
     |   {I,time(RX_v)-time(RG_v)},I'] |               |
     |                   |             |               |
     ~                   |             ~               ~
     |                   |             |               |
     |                   |             |          time(OP_r)
```

Figure 10: Epoch ID-based Passport Model

A.4. Example 4: Timestamp-based Background-Check Model Example

The following example illustrates a hypothetical Background-Check Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party. The Attester conveys Evidence to the Relying Party, which treats it as opaque and simply forwards it on to the Verifier.

```
  .----------.         .---------------.                    .----------.
  | Attester |         | Relying Party |                    | Verifier |
  '-------+--'         '-------+-------'                    '----+-----'
          |                    |                                 |
    time(VG_a)                 |                                 |
          |                    |                                 |
          ~                    ~                                 ~
          |                    |                                 |
    time(EG_a)                 |                                 |
          |                    |                                 |
          +----Evidence------->|                                 |
          |   {time(EG_a)}     |                                 |
          |               time(ER_r) ---Evidence{time(EG_a)}---->|
          |                    |                                 |
          |                    |                            time(RG_v)
          |                    |                                 |
          |               time(RA_r) <---Attestation Result------+
          |                    |           {time(RX_v)}          |
          ~                    ~                                 ~
          |                    |                                 |
          |                  time(OP_r)                          |
```

The time considerations in this example are equivalent to those discussed under Example 1 above.

A.5. Example 5: Nonce-based Background-Check Model Example

The following example illustrates a hypothetical Background-Check Model solution that uses nonces and thus does not require that any clocks are synchronized. In this example solution, a nonce is generated by a Verifier at the request of a Relying Party, when the Relying Party needs to send one to an Attester.

```
   .----------.         .---------------.                .----------.
   | Attester |         | Relying Party |                | Verifier |
   '----+-----'         '-------+-------'                '----+-----'
        |                       |                              |
     time(VG_a)                 |                              |
        |                       |                              |
        ~                       ~                              ~
        |                       |                              |
        |                       |<-------Nonce-----------time(NS_v)
        |                       |                              |
        |<---Nonce-----------time(NR_r)                        |
        |                       |                              |
     time(EG_a)                 |                              |
        |                       |                              |
        +----Evidence{Nonce}--->|                              |
        |                       |                              |
        |                    time(ER_r) ---Evidence{Nonce}--->|
        |                       |                              |
        |                       |                           time(RG_v)
        |                       |                              |
        |                   ime(RA_r) <---Attestation Result--+
        |                       |    {time(RX_v)-time(RG_v)}   |
        ~                       ~                              ~
        |                       |                              |
        |                    time(OP_r)                        |
```

The Verifier can check whether the Evidence is fresh, and whether a Claim value is recent, the same as in Example 2 above.

However, unlike in Example 2, the Relying Party can use the Nonce to determine whether the Attestation Result is fresh, by verifying that time(OP_r)-time(NR_r) < Threshold.

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of time(RX_v)-time(RG_v), then the Relying Party can check time(OP_r)-time(ER_r) < time(RX_v)-time(RG_v).

Contributors

Monty Wiseman
Email: montywiseman32@gmail.com

Liang Xia
Email: frank.xialiang@huawei.com

Laurence Lundblade
Email: lgl@island-resort.com

Eliot Lear
Email: elear@cisco.com

Jessica Fitzgerald-McKay

Sarah C. Helbe

Andrew Guinn

Peter Loscocco
Email: pete.loscocco@gmail.com

Eric Voit

Thomas Fossati
Email: thomas.fossati@arm.com

Paul Rowe

Carsten Bormann
Email: cabo@tzi.org

Giri Mandyam
Email: mandyam@qti.qualcomm.com

Kathleen Moriarty
Email: kathleen.moriarty.ietf@gmail.com

Guy Fedorkow
Email: gfedorkow@juniper.net

Simon Frost

Email: Simon.Frost@arm.com

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Dave Thaler
Microsoft
United States of America
Email: dthaler@microsoft.com

Michael Richardson
Sandelman Software Works
Canada
Email: mcr+ietf@sandelman.ca

Ned Smith
Intel Corporation
United States of America
Email: ned.smith@intel.com

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

RATS
Internet-Draft
Intended status: Standards Track
Expires: 18 July 2024

L. Lundblade
Security Theory LLC
G. Mandyam

J. O'Donoghue
Qualcomm Technologies Inc.
C. Wallace
Red Hound Software, Inc.
15 January 2024

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-25

Abstract

An Entity Attestation Token (EAT) provides an attested claims set that describes state and characteristics of an entity, a device like a smartphone, IoT device, network equipment or such. This claims set is used by a relying party, server or service to determine the type and degree of trust placed in the entity.

An EAT is either a CBOR Web Token (CWT) or JSON Web Token (JWT) with attestation-oriented claims.

Status of This Memo

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

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

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

This Internet-Draft will expire on 18 July 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 5
1.1. Entity Overview 7
1.2. EAT as a Framework 8
1.3. Operating Model and RATS Architecture 9
1.3.1. Relationship between Evidence and Attestation Results 9
2. Terminology 10
3. Top-Level Token Definition 12
4. The Claims 13
4.1. eat_nonce (EAT Nonce) Claim 14
4.2. Claims Describing the Entity 14
4.2.1. ueid (Universal Entity ID) Claim 15
4.2.1.1. Rules for Creating UEIDs 15
4.2.1.2. Rules for Consuming UEIDs 18
4.2.2. sueids (Semi-permanent UEIDs) Claim (SUEIDs) 18
4.2.3. oemid (Hardware OEM Identification) Claim 19
4.2.3.1. Random Number Based OEM ID 19
4.2.3.2. IEEE Based OEM ID 20
4.2.3.3. IANA Private Enterprise Number Based OEM ID . . . 20
4.2.4. hwmodel (Hardware Model) Claim 21
4.2.5. hwversion (Hardware Version) Claim 22
4.2.6. swname (Software Name) Claim 22
4.2.7. swversion (Software Version) Claim 22
4.2.8. oemboot (OEM Authorized Boot) Claim 23
4.2.9. dbgstat (Debug Status) Claim 23
4.2.9.1. Enabled 24
4.2.9.2. Disabled 24
4.2.9.3. Disabled Since Boot 24
4.2.9.4. Disabled Permanently 24
4.2.9.5. Disabled Fully and Permanently 25
4.2.10. location (Location) Claim 25
4.2.11. uptime (Uptime) Claim 26
4.2.12. bootcount (Boot Count) Claim 26
4.2.13. bootseed (Boot Seed) Claim 26
4.2.14. dloas (Digital Letters of Approval) Claim 27
4.2.15. manifests (Software Manifests) Claim 28
4.2.16. measurements (Measurements) Claim 29

4.2.17. measres (Software Measurement Results) Claim 30
4.2.18. submods (Submodules) 32
4.2.18.1. Submodule Claims-Set 35
4.2.18.2. Detached Submodule Digest 36
4.2.18.3. Nested Tokens 36
4.3. Claims Describing the Token 36
4.3.1. iat (Timestamp) Claim 37
4.3.2. eat_profile (EAT Profile) Claim 37
4.3.3. intuse (Intended Use) Claim 38
5. Detached EAT Bundles 39
6. Profiles 40
6.1. Format of a Profile Document 41
6.2. Full and Partial Profiles 41
6.3. List of Profile Issues 42
6.3.1. Use of JSON, CBOR or both 42
6.3.2. CBOR Map and Array Encoding 42
6.3.3. CBOR String Encoding 43
6.3.4. CBOR Preferred Serialization 43
6.3.5. CBOR Tags 43
6.3.6. COSE/JOSE Protection 43
6.3.7. COSE/JOSE Algorithms 44
6.3.8. Detached EAT Bundle Support 44
6.3.9. Key Identification 44
6.3.10. Endorsement Identification 45
6.3.11. Freshness 45
6.3.12. Claims Requirements 45
6.4. The Constrained Device Standard Profile 46
7. Encoding and Collected CDDL 48
7.1. Claims-Set and CDDL for CWT and JWT 48
7.2. Encoding Data Types 48
7.2.1. Common Data Types 49
7.2.2. JSON Interoperability 49
7.2.3. Labels 50
7.2.4. CBOR Interoperability 50
7.3. Collected CDDL 50
7.3.1. Payload CDDL 50
7.3.2. CBOR-Specific CDDL 55
7.3.3. JSON-Specific CDDL 56
8. Privacy Considerations 57
8.1. UEID and SUEID Privacy Considerations 57
8.2. Location Privacy Considerations 58
8.3. Boot Seed Privacy Considerations 58
8.4. Replay Protection and Privacy 58
9. Security Considerations 58
9.1. Claim Trustworthiness 58
9.2. Key Provisioning 59
9.2.1. Transmission of Key Material 59
9.3. Freshness 60

9.4. Multiple EAT Consumers 60
9.5. Detached EAT Bundle Digest Security Considerations . . . 60
9.6. Verification Keys 61
10. IANA Considerations 61
10.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries 61
10.2. CWT and JWT Claims Registered by This Document 61
10.3. UEID URN Registered by this Document 68
10.4. CBOR Tag for Detached EAT Bundle Registered by this Document 69
11. References 69
11.1. Normative References 69
11.2. Informative References 72
Appendix A. Examples 74
A.1. Claims Set Examples 74
A.1.1. Simple TEE Attestation 74
A.1.2. Submodules for Board and Device 76
A.1.3. EAT Produced by Attestation Hardware Block 77
A.1.4. Key / Key Store Attestation 78
A.1.5. Software Measurements of an IoT Device 80
A.1.6. Attestation Results in JSON 82
A.1.7. JSON-encoded Token with Submodules 83
A.2. Signed Token Examples 84
A.2.1. Basic CWT Example 84
A.2.2. CBOR-encoded Detached EAT Bundle 85
A.2.3. JSON-encoded Detached EAT Bundle 87
Appendix B. UEID Design Rationale 88
B.1. Collision Probability 88
B.2. No Use of UUID 91
Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID) 91
C.1. DevID Used With EAT 92
C.2. How EAT Provides an Equivalent Secure Device Identity . . 92
C.3. An X.509 Format EAT 93
C.4. Device Identifier Permanence 93
Appendix D. CDDL for CWT and JWT 94
Appendix E. New Claim Design Considerations 96
E.1. Interoperability and Relying Party Orientation 96
E.2. Operating System and Technology Neutral 96
E.3. Security Level Neutral 97
E.4. Reuse of Extant Data Formats 97
E.5. Proprietary Claims 97
Appendix F. Endorsements and Verification Keys 98
F.1. Identification Methods 99
F.1.1. COSE/JWS Key ID 99
F.1.2. JWS and COSE X.509 Header Parameters 99
F.1.3. CBOR Certificate COSE Header Parameters 99
F.1.4. Claim-Based Key Identification 100

Appendix G. Changes from Previous Drafts 100
G.1. From draft-ietf-rats-eat-24 100
Contributors 100
Authors' Addresses 101

1. Introduction

An Entity Attestation Token (EAT) is a message made up of claims about an entity. An entity may be a device, some hardware or some software. The claims are ultimately used by a relying party who decides if and how it will interact with the entity. The relying party may choose to trust, not trust or partially trust the entity. For example, partial trust may be allowing a monetary transaction only up to a limit.

The security model and goal for attestation are unique and are not the same as for other security standards like those for server authentication, user authentication and secured messaging. To give an example of one aspect of the difference, consider the association and life-cycle of key material. For authentication, keys are associated with a user or service and set up by actions performed by a user or an operator of a service. For attestation, the keys are associated with specific devices and are configured by device manufacturers. The reader is assumed to be familiar with the goals and security model for attestation as described in RATS Architecture [RFC9334] and are not repeated here.

This document defines some common claims that are potentially of broad use. EAT additionally allows proprietary claims and for further claims to be standardized. Here are some examples:

* Make and model of manufactured consumer device

* Make and model of a chip or processor, particularly for a security-oriented chip

* Identification and measurement of the software running on a device

* Configuration and state of a device

* Environmental characteristics of a device like its Global Positioning Sytem (GPS) location

* Formal certifications received

EAT is constructed to support a wide range of use cases.

No single set of claims can accommodate all use cases so EAT is constructed as a framework for defining specific attestation tokens for specific use cases. In particular, EAT provides a profile mechanism to be able to clearly specify the claims needed, the cryptographic algorithms that should be used, and other characteristics for a particular token and use case. Section 6 describes profile contents and provides a profile that is suitable for constrained device use cases.

The entity's EAT implementation generates the claims and typically signs them with an attestation key. It is responsible for protecting the attestation key. Some EAT implementations will use components with very high resistance to attack like Trusted Platform Modules or Secure Elements. Others may rely solely on simple software defenses.

Nesting of tokens and claims sets is accommodated for composite devices that have multiple subsystems.

An EAT may be encoded in either JavaScript Object Notation (JSON) [RFC8259] or Concise Binary Object Representation (CBOR) [RFC8949] as needed for each use case. EAT is built on CBOR Web Token (CWT) [RFC8392] and JSON Web Token (JWT) [RFC7519] and inherits all their characteristics and their security mechanisms. Like CWT and JWT, EAT does not imply any message flow.

Following is a very simple example. It is JSON format for easy reading, but could also be CBOR. Only the Claims-Set, the payload for the JWT, is shown.

```
{
    "eat_nonce": "MIDBNH28iioisjPy",
    "ueid":      "AgAEizrK3Q",
    "oemid":     76543,
    "swname":    "Acme IoT OS",
    "swversion": "3.1.4"
}
```

This example has a nonce for freshness. This nonce is the base64url encoding of a 12 byte random binary byte string. The ueid is effectively a serial number uniquely identifying the device. This ueid is the base64url encoding of a 48-bit MAC address preceded by the type byte 0x02. The oemid identifies the manufacturer using a Private Enterprise Number [PEN]. The software is identified by a simple string name and version. It could be identified by a full manifest, but this is a minimal example.

1.1. Entity Overview

This document uses the term "entity" to refer to the target of an EAT. Most of the claims defined in this document are claims about an entity. An entity is equivalent to a target environment in an attester as defined in [RFC9334].

Layered attestation and composite devices, as described in [RFC9334], are supported by a submodule mechanism (see Section 4.2.18). Submodules allow nesting of EATs and of claims-sets so that such hierarchies can be modeled.

An entity is the same as a "system component", as defined in the Internet Security Glossary [RFC4949].

Note that [RFC4949] defines "entity" and "system entity" as synonyms, and that they may be a person or organization in addition to being a system component. In the EAT context, "entity" never refers to a person or organization. The hardware and software that implement a web site server or service may be an entity in the EAT sense, but the organization that operates, maintains or hosts the web site is not an entity.

Some examples of entities:

* A Secure Element

* A Trusted Execution Environment (TEE)

* A network card in a router

* A router, perhaps with each network card in the router a submodule

* An Internet of Things (IoT) device

* An individual process

* An app on a smartphone

* A smartphone with many submodules for its many subsystems

* A subsystem in a smartphone like the modem or the camera

An entity may have strong security defenses against hardware invasive attacks. It may also have low security, having no special security defenses. There is no minimum security requirement to be an entity.

1.2. EAT as a Framework

EAT is a framework for defining attestation tokens for specific use cases, not a specific token definition. While EAT is based on and compatible with CWT and JWT, it can also be described as:

* An identification and type system for claims in claims-sets

* Definitions of common attestation-oriented claims

* Claims defined in CDDL and serialized using CBOR or JSON

* Security envelopes based on CBOR Object Signing and Encryption (COSE) and Javascript Object Signing and Encryption (JOSE)

* Nesting of claims sets and tokens to represent complex and compound devices

* A profile mechanism for specifying and identifying specific tokens for specific use cases

EAT uses the name/value pairs the same as CWT and JWT to identify individual claims. Section 4 defines common attestation-oriented claims that are added to the CWT and JWT IANA registries. As with CWT and JWT, no claims are mandatory and claims not recognized should be ignored.

Unlike, but compatible with CWT and JWT, EAT defines claims using Concise Data Definition Language (CDDL) [RFC8610]. In most cases the same CDDL definition is used for both the CBOR/CWT serialization and the JSON/JWT serialization.

Like CWT and JWT, EAT uses COSE and JOSE to provide authenticity, integrity and optionally confidentiality. EAT places no new restrictions on cryptographic algorithms, retaining all the cryptographic flexibility of CWT, COSE, JWT and JOSE.

EAT defines a means for nesting tokens and claims sets to accommodate composite devices that have multiple subsystems and multiple attesters. Tokens with security envelopes or bare claims sets may be embedded in an enclosing token. The nested token and the enclosing token do not have to use the same encoding (e.g., a CWT may be enclosed in a JWT).

EAT adds the ability to detach claims sets and send them separately from a security-enveloped EAT that contains a digest of the detached claims set.

This document registers no media or content types for the identification of the type of EAT, its serialization encoding or security envelope. The definition and registration of EAT media types is addressed in [EAT.media-types].

Finally, the notion of an EAT profile is introduced that facilitates the creation of narrowed definitions of EATs for specific use cases in follow-on documents. One basic profile for constrained devices is normatively defined.

1.3. Operating Model and RATS Architecture

EAT follows the operational model described in Figure 1 in RATS Architecture [RFC9334]. To summarize, an attester generates evidence in the form of a claims set describing various characteristics of an entity. Evidence is usually signed by a key that proves the attester and the evidence it produces are authentic. The claims set includes a nonce or some other means to assure freshness.

A verifier confirms an EAT is valid by verifying the signature and may vet some claims using reference values. The verifier then produces attestation results, which may also be represented as an EAT. The attestation results are provided to the relying party, which is the ultimate consumer of the Remote Attestation Procedure. The relying party uses the attestation results as needed for its use case, perhaps allowing an entity to access a network, allowing a financial transaction or such. In some cases, the verifier and relying party are not distinct entities.

1.3.1. Relationship between Evidence and Attestation Results

Any claim defined in this document or in the IANA CWT or JWT registry may be used in evidence or attestation results. The relationship of claims in attestation results to evidence is fundamentally governed by the verifier and the verifier’s policy.

A common use case is for the verifier and its policy to perform checks, calculations and processing with evidence as the input to produce a summary result in attestation results that indicates the overall health and status of the entity. For example, measurements in evidence may be compared to reference values the results of which are represented as a simple pass/fail in attestation results.

It is also possible that some claims in the Evidence will be forwarded unmodified to the relying party in attestation results. This forwarding is subject to the verifier’s implementation and policy. The relying party should be aware of the verifier’s policy to know what checks it has performed on claims it forwards.

The verifier may modify claims it forwards, for example, to implement a privacy preservation functionality. It is also possible the verifier will put claims in the attestation results that give details about the entity that it has computed or looked up in a database. For example, the verifier may be able to put an "oemid" claim in the attestation results by performing a look up based on a "ueid" claim (e.g., serial number) it received in evidence.

This specification does not establish any normative rules for the verifier to follow, as these are a matter of local policy. It is up to each relying party to understand the processing rules of each verifier to know how to interpret claims in attestation results.

2. Terminology

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.

In this document, the structure of data is specified in CDDL [RFC8610] [RFC9165].

The examples in Appendix A use CBOR diagnostic notation defined in Section 8 of [RFC8949] and Appendix G of [RFC8610].

This document reuses terminology from JWT [RFC7519] and CWT [RFC8392]:

base64url-encoded: base64url-encoded is as described in [RFC7515], i.e., using URL- and filename-safe character set [RFC4648] with all trailing '=' characters omitted and without the inclusion of any line breaks, whitespace, or other additional characters.

Claim: A piece of information asserted about a subject. A claim is represented as pair with a value and either a name or key to identify it.

Claim Name: A unique text string that identifies the claim. It is used as the claim name for JSON encoding.

Claim Key: The CBOR map key used to identify a claim. (The term "Claim Key" comes from CWT. This document, like COSE, uses the term "label" to refer to CBOR map keys to avoid confusion with cryptographic keys.)

Claim Value: The value portion of the claim. A claim value can be

any CBOR data item or JSON value.

Claims Set: The CBOR map or JSON object that contains the claims conveyed by the CWT or JWT.

This document reuses terminology from RATS Architecure [RFC9334]:

Attester: A role performed by an entity (typically a device) whose evidence must be appraised in order to infer the extent to which the attester is considered trustworthy, such as when deciding whether it is authorized to perform some operation.

Verifier: A role that appraises the validity of evidence about an attester and produces attestation results to be used by a relying party.

Relying Party: A role that depends on the validity of information about an attester, for purposes of reliably applying application specific actions. Compare /relying party/ in [RFC4949].

Evidence: A set of claims generated by an attester to be appraised by a verifier. Evidence may include configuration data, measurements, telemetry, or inferences.

Attestation Results: The output generated by a verifier, typically including information about an attester, where the verifier vouches for the validity of the results

Reference Values: A set of values against which values of claims can be compared as part of applying an appraisal policy for evidence. Reference Values are sometimes referred to in other documents as known-good values, golden measurements, or nominal values, although those terms typically assume comparison for equality, whereas here reference values might be more general and be used in any sort of comparison.

Endorsement: A secure statement that an Endorser vouches for the integrity of an attester’s various capabilities such as claims collection and evidence signing.

This document reuses terminology from CDDL [RFC8610]:

Group Socket: refers to the mechanism by which a CDDL definition is extended, as described in [RFC8610] and [RFC9165]

3. Top-Level Token Definition

An "EAT" is an encoded (serialized) message the purpose of which is to transfer a Claims-Set between two parties. An EAT MUST always contain a Claims-Set. In this document an EAT is always a CWT or JWT.

An EAT MUST have authenticity and integrity protection. CWT and JWT provide that in this document.

Further documents may define other encodings and security mechanims for EAT.

The identification of a protocol element as an EAT follows the general conventions used for CWTs and JWTs. Identification depends on the protocol carrying the EAT. In some cases it may be by media type (e.g., in a HTTP Content-Type field). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This document also defines another message, the detached EAT bundle (see Section 5), which holds a collection of detached claims sets and an EAT that provides integrity and authenticity protection for them. Detached EAT bundles can be either CBOR or JSON encoded.

The following CDDL defines the top-level $EAT-CBOR-Tagged-Token, $EAT-CBOR-Untagged-Token and $EAT-JSON-Token-Formats sockets (see Section 3.9 of [RFC8610]), enabling future token formats to be defined. Any new format that plugs into one or more of these sockets MUST be defined by an IETF standards action. Of particular use may be a token type that provides no direct authenticity or integrity protection for use with transports mechanisms that do provide the necessary security services [UCCS].

Nesting of EATs is allowed and defined in Section 4.2.18.3. This includes the nesting of an EAT that is a different format than the enclosing EAT, i.e., the nested EAT may be encoded using CBOR and the enclosing EAT encoded using JSON or vice versa. The definition of Nested-Token references the CDDL defined in this section. When new token formats are defined, the means for identification in a nested token MUST also be defined.

The top-level CDDL type for CBOR-encoded EATs is EAT-CBOR-Token and for JSON is EAT-JSON-Token (while CDDL and CDDL tools provide enough support for shared definitions of most items in this document, they dont provide enough support for this sharing at the top level).

```
EAT-CBOR-Token = $EAT-CBOR-Tagged-Token / $EAT-CBOR-Untagged-Token

$EAT-CBOR-Tagged-Token /= CWT-Tagged-Message
$EAT-CBOR-Tagged-Token /= BUNDLE-Tagged-Message

$EAT-CBOR-Untagged-Token /= CWT-Untagged-Message
$EAT-CBOR-Untagged-Token /= BUNDLE-Untagged-Message

EAT-JSON-Token = $EAT-JSON-Token-Formats

$EAT-JSON-Token-Formats /= JWT-Message
$EAT-JSON-Token-Formats /= BUNDLE-Untagged-Message
```

4. The Claims

This section describes new claims defined for attestation that are to be added to the CWT [IANA.CWT.Claims] and JWT [IANA.JWT.Claims] IANA registries.

All definitions, requirements, creation and validation procedures, security considerations, IANA registrations and so on from CWT and JWT carry over to EAT.

This section also describes how several extant CWT and JWT claims apply in EAT.

The set of claims that an EAT must contain to be considered valid is context dependent and is outside the scope of this specification. Specific applications of EATs will require implementations to understand and process some claims in particular ways. However, in the absence of such requirements, all claims that are not understood by implementations MUST be ignored.

CDDL, along with a text description, is used to define each claim independent of encoding. Each claim is defined as a CDDL group. In Section 7 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

Each claim defined in this document is added to the $$Claims-Set-Claims group socket. Claims defined by other specifications MUST also be added to the $$Claims-Set-Claims group socket.

All claims in an EAT MUST use the same encoding except where otherwise explicitly stated (e.g., in a CBOR-encoded token, all claims must be CBOR-encoded).

This specification includes a CDDL definition of most of what is defined in [RFC8392]. Similarly, this specification includes CDDL for most of what is defined in [RFC7519]. These definitions are in Appendix D and are not normative.

Each claim described has a unique text string and integer that identifies it. CBOR-encoded tokens MUST use only the integer for claim keys. JSON-encoded tokens MUST use only the text string for claim names.

4.1. eat_nonce (EAT Nonce) Claim

An EAT nonce is either a byte or text string or an array of byte or text strings. The array option supports multistage EAT verification and consumption.

A claim named "nonce" was defined and registered with IANA for JWT, but MUST NOT be used because it does not support multiple nonces. No previous "nonce" claim was defined for CWT. To distinguish from the previously defined JWT "nonce" claim, this claim is named "eat_nonce" in JSON-encoded EATs. The CWT nonce defined here is intended for general purpose use and retains the "Nonce" claim name instead of an EAT-specific name.

An EAT nonce MUST have at least 64 bits of entropy. A maximum EAT nonce size is set to limit the memory required for an implementation. All receivers MUST be able to accommodate the maximum size.

In CBOR, an EAT nonce is a byte string between 8 and 64 bytes in length. In JSON, an EAT nonce is a text string between 8 and 88 bytes in length.

```
$$Claims-Set-Claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

nonce-type = JC< tstr .size (8..88), bstr .size (8..64)>
```

4.2. Claims Describing the Entity

The claims in this section describe the entity itself. They describe the entity whether they occur in evidence or occur in attestation results. See Section 1.3.1 for discussion on how attestation results relate to evidence.

4.2.1. ueid (Universal Entity ID) Claim

The "ueid" claim conveys a UEID, which identifies an individual manufactured entity like a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire entity or a submodule. It does not identify types, models or classes of entities. It is akin to a serial number, though it does not have to be sequential.

UEIDs MUST be universally and globally unique across manufacturers and countries, as described in Section 4.2.1.1. UEIDs MUST also be unique across protocols and systems, as tokens are intended to be embedded in many different protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries should have the same UEID (if they are not global and universal in this way, then relying parties receiving them will have to track other characteristics of the entity to keep entities distinct between manufacturers).

UEIDs are not designed for direct use by humans (e.g., printing on the case of a device), so no such representation is defined.

There are privacy considerations for UEIDs. See Section 8.1.

A Device Identifier URN is registered for UEIDs. See Section 10.3.

```
$$Claims-Set-Claims //= (ueid-label => ueid-type)

ueid-type = JC<base64-url-text .size (10..44) , bstr .size (7..33)>
```

4.2.1.1. Rules for Creating UEIDs

These rules are solely for the creation of UEIDs. The EAT consumer need not have any awareness of them.

A UEID is constructed of a single type byte followed by the unique bytes for that type. The type byte assures global uniqueness of a UEID even if the unique bytes for different types are accidentally the same.

UEIDS are variable length to accommodate the types defined here and future-defined types.

UEIDs SHOULD NOT be longer than 33 bytes. If they are longer, there is no guarantee that a receiver will be able to accept them. See Appendix B.

A UEID is permanent. It MUST never change for a given entity.

The different types of UEIDs 1) accommodate different manufacturing processes, 2) accommodate small UEIDs, 3) provide an option that doesn't require registration fees and central administration.

In the unlikely event that a new UEID type is needed, it MUST be defined in a standards-track update to this document.

A manufacturer of entities MAY use different types for different products. They MAY also change from one type to another for a given product or use one type for some items of a given produce and another type for other.

Type Byte	Type Name	Specification
0x01	RAND	This is a 128, 192 or 256-bit random number generated once and stored in the entity. This may be constructed by concatenating enough identifiers to make up an equivalent number of random bits and then feeding the concatenation through a cryptographic hash function. It may also be a cryptographic quality random number generated once at the beginning of the life of the entity and stored. It MUST NOT be smaller than 128 bits. See the length analysis in Appendix B.
0x02	IEEE EUI	This makes use of the device identification scheme operated by the IEEE. An EUI is either an EUI-48, EUI-60 or EUI-64 and made up of an OUI, OUI-36 or a CID, different registered company identifiers, and some unique per-entity identifier. EUIs are often the same as or similar to MAC addresses. This type includes MAC-48, an obsolete name for EUI-48. (Note that while entities with multiple network interfaces may have multiple MAC addresses, there is only one UEID for an entity; changeable MAC addresses that don't meet the permanence requirements in this document MUST NOT be used for the UEID or SUEID) [IEEE.802-2001], [OUI.Guide].
0x03	IMEI	This makes use of the International Mobile Equipment Identity (IMEI) scheme operated by the GSMA. This is a 14-digit identifier consisting of an 8-digit Type Allocation Code (TAC) and a 6-digit serial number allocated by the manufacturer, which SHALL be encoded as byte string of length 14 with each byte as the digit's value (not the ASCII encoding of the digit; the digit 3 encodes as 0x03, not 0x33). The IMEI value encoded SHALL NOT include Luhn checksum or SVN information. See [ThreeGPP.IMEI].

Table 1: UEID Composition Types

4.2.1.2. Rules for Consuming UEIDs

For the consumer, a UEID is solely a globally unique opaque identifier. The consumer does not and should not have any awareness of the rules and structure used to achieve global uniqueness.

All implementations MUST be able to receive UEIDs up to 33 bytes long. 33 bytes is the longest defined in this document and gives necessary entropy for probabilistic uniqueness.

The consumer of a UEID MUST treat it as a completely opaque string of bytes and MUST NOT make any use of its internal structure. The reasons for this are:

* UEIDs types vary freely from one manufacturer to the next.

* New types of UEIDs may be defined.

* The manufacturer of an entity is allowed to change from one type of UEID to another anytime they want.

For example, when the consumer receives a type 0x02 UEID, they should not use the OUI part to identify the manufacturer of the device because there is no guarantee all UEIDs will be type 0x02. Different manufacturers may use different types. A manufacturer may make some of their product with one type and others with a different type or even change to a different type for newer versions of their product. Instead, the consumer should use the "oemid" claim.

4.2.2. sueids (Semi-permanent UEIDs) Claim (SUEIDs)

The "sueids" claim conveys one or more semi-permanent UEIDs (SUEIDs). An SUEID has the same format, characteristics and requirements as a UEID, but MAY change to a different value on entity life-cycle events. An entity MAY have both a UEID and SUEIDs, neither, one or the other.

Examples of life-cycle events are change of ownership, factory reset and on-boarding into an IoT device management system. It is beyond the scope of this document to specify particular types of SUEIDs and the life-cycle events that trigger their change. An EAT profile MAY provide this specification.

There MAY be multiple SUEIDs. Each has a text string label the purpose of which is to distinguish it from others. The label MAY name the purpose, application or type of the SUEID. For example, the label for the SUEID used by XYZ Onboarding Protocol could thus be "XYZ". It is beyond the scope of this document to specify any SUEID labeling schemes. They are use case specific and MAY be specified in an EAT profile.

If there is only one SUEID, the claim remains a map and there still MUST be a label.

An SUEID provides functionality similar to an IEEE LDevID [IEEE.802.1AR].

There are privacy considerations for SUEIDs. See Section 8.1.

A Device Identifier URN is registered for SUEIDs. See Section 10.3.

```
$$Claims-Set-Claims //= (sueids-label => sueids-type)

sueids-type = {
    + tstr => ueid-type
}
```

4.2.3. oemid (Hardware OEM Identification) Claim

The "oemid" claim identifies the Original Equipment Manufacturer (OEM) of the hardware. Any of the three forms described below MAY be used at the convenience of the claim sender. The receiver of this claim MUST be able to handle all three forms.

Note that the "hwmodel" claim in Section 4.2.4, the "oemboot" claim in Section 4.2.8 and "dbgstat" claim in Section 4.2.9 depend on this claim.

Sometimes one manufacturer will acquire or merge with another. Depending on the situation and use case newly manfactured devices may continue to use the old OEM ID or switch to a new one. This is left to the discretion of the manufacturers, but they should consider how it affects the above-mentioned claims and the attestation eco-system for their devices. The considerations are the same for all three forms of this claim.

4.2.3.1. Random Number Based OEM ID

The random number based OEM ID MUST always be 16 bytes (128 bits) long.

The OEM may create their own ID by using a cryptographic-quality random number generator. They would perform this only once in the life of the company to generate the single ID for said company. They would use that same ID in every entity they make. This uniquely identifies the OEM on a statistical basis and is large enough should there be ten billion companies.

In JSON-encoded tokens this MUST be base64url-encoded.

4.2.3.2. IEEE Based OEM ID

The IEEE operates a global registry for MAC addresses and company IDs. This claim uses that database to identify OEMs. The contents of the claim may be either an IEEE MA-L, MA-M, MA-S or an IEEE CID [IEEE-RA]. An MA-L, formerly known as an OUI, is a 24-bit value used as the first half of a MAC address. MA-M similarly is a 28-bit value uses as the first part of a MAC address, and MA-S, formerly known as OUI-36, a 36-bit value. Many companies already have purchased one of these. A CID is also a 24-bit value from the same space as an MA-L, but not for use as a MAC address. IEEE has published Guidelines for Use of EUI, OUI, and CID [OUI.Guide] and provides a lookup service [OUI.Lookup].

Companies that have more than one of these IDs or MAC address blocks SHOULD select one and prefer that for all their entities.

Commonly, these are expressed in Hexadecimal Representation as described in [IEEE.802-2001]. It is also called the Canonical format. When this claim is encoded the order of bytes in the bstr are the same as the order in the Hexadecimal Representation. For example, an MA-L like "AC-DE-48" would be encoded in 3 bytes with values 0xAC, 0xDE, 0x48.

This format is always 3 bytes in size in CBOR.

In JSON-encoded tokens, this MUST be base64url-encoded and always 4 bytes.

4.2.3.3. IANA Private Enterprise Number Based OEM ID

IANA maintains a registry for Private Enterprise Numbers (PEN) [PEN]. A PEN is an integer that identifies an enterprise and may be used to construct an object identifier (OID) relative to the following OID arc that is managed by IANA: iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1).

For EAT purposes, only the integer value assigned by IANA as the PEN is relevant, not the full OID value.

In CBOR this value MUST be encoded as a major type 0 integer and is typically 3 bytes. In JSON, this value MUST be encoded as a number.

```
$$Claims-Set-Claims //= (
    oemid-label => oemid-pen / oemid-ieee / oemid-random
)

oemid-pen = int

oemid-ieee = JC<oemid-ieee-json, oemid-ieee-cbor>
oemid-ieee-cbor = bstr .size 3
oemid-ieee-json = base64-url-text .size 4

oemid-random = JC<oemid-random-json, oemid-random-cbor>
oemid-random-cbor = bstr .size 16
oemid-random-json = base64-url-text .size 24
```

4.2.4. hwmodel (Hardware Model) Claim

The "hwmodel" claim differentiates hardware models, products and variants manufactured by a particular OEM, the one identified by OEM ID in Section 4.2.3. It MUST be unique within a given OEM ID. The concatenation of the OEM ID and "hwmodel" give a global identifier of a particular product. The "hwmodel" claim MUST only be present if an "oemid" claim described in Section 4.2.3 is present.

The granularity of the model identification is for each OEM to decide. It may be very granular, perhaps including some version information. It may be very general, perhaps only indicating top-level products.

The "hwmodel" claim is for use in protocols and not for human consumption. The format and encoding of this claim should not be human-readable to discourage use other than in protocols. If this claim is to be derived from an already-in-use human-readable identifier, it can be run through a hash function.

There is no minimum length so that an OEM with a very small number of models can use a one-byte encoding. The maximum length is 32 bytes. All receivers of this claim MUST be able to receive this maximum size.

The receiver of this claim MUST treat it as a completely opaque string of bytes, even if there is some apparent naming or structure. The OEM is free to alter the internal structure of these bytes as long as the claim continues to uniquely identify its models.

```
$$Claims-Set-Claims //= (
    hardware-model-label => hardware-model-type
)

hardware-model-type = JC<base64-url-text .size (4..44),
                         bytes .size (1..32)>
```

4.2.5. hwversion (Hardware Version) Claim

The "hwversion" claim is a text string the format of which is set by each manufacturer. The structure and sorting order of this text string can be specified using the version-scheme item from CoSWID [RFC9393]. It is useful to know how to sort versions so the newer can be distinguished from the older. A "hwversion" claim MUST only be present if a "hwmodel" claim described in Section 4.2.4 is present.

```
$$Claims-Set-Claims //=  (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version:  tstr,
    ? scheme:  $version-scheme
]
```

4.2.6. swname (Software Name) Claim

The "swname" claim contains a very simple free-form text value for naming the software used by the entity. Intentionally, no general rules or structure are set. This will make it unsuitable for use cases that wish precise naming.

If precise and rigourous naming of the software for the entity is needed, the "manifests" claim described in Section 4.2.15 may be used instead.

```
$$Claims-Set-Claims //= ( sw-name-label => tstr )
```

4.2.7. swversion (Software Version) Claim

The "swversion" claim makes use of the CoSWID version-scheme defined in [RFC9393] to give a simple version for the software. A "swversion" claim MUST only be present if a "swname" claim described in Section 4.2.6 is present.

The "manifests" claim Section 4.2.15 may be instead if this is too simple.

```
$$Claims-Set-Claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version:  tstr
    ? scheme:  $version-scheme
]
```

4.2.8. oemboot (OEM Authorized Boot) Claim

An "oemboot" claim with value of true indicates the entity booted with software authorized by the manufacturer of the entity as indicated by the "oemid" claim described in Section 4.2.3. It indicates the firmware and operating system are fully under control of the OEM and may not be replaced by the end user or even the enterprise that owns the device. The means of control may be by cryptographic authentication of the software, by the software being in Read-Only Memory (ROM), a combination of the two or other. If this claim is present the "oemid" claim MUST be present.

```
$$Claims-Set-Claims //= (oem-boot-label => bool)
```

4.2.9. dbgstat (Debug Status) Claim

The "dbgstat" claim applies to entity-wide or submodule-wide debug facilities of the entity like [JTAG] and diagnostic hardware built into chips. It applies to any software debug facilities related to privileged software that allows system-wide memory inspection, tracing or modification of non-system software like user mode applications.

This characterization assumes that debug facilities can be enabled and disabled in a dynamic way or be disabled in some permanent way, such that no enabling is possible. An example of dynamic enabling is one where some authentication is required to enable debugging. An example of permanent disabling is blowing a hardware fuse in a chip. The specific type of the mechanism is not taken into account. For example, it does not matter if authentication is by a global password or by per-entity public keys.

As with all claims, the absence of the "dbgstat" claim means it is not reported.

This claim is not extensible so as to provide a common interoperable description of debug status. If a particular implementation considers this claim to be inadequate, it can define its own proprietary claim. It may consider including both this claim as a coarse indication of debug status and its own proprietary claim as a refined indication.

The higher levels of debug disabling requires that all debug disabling of the levels below it be in effect. Since the lowest level requires that all of the target's debug be currently disabled, all other levels require that too.

There is no inheritance of claims from a submodule to a superior module or vice versa. There is no assumption, requirement or guarantee that the target of a superior module encompasses the targets of submodules. Thus, every submodule must explicitly describe its own debug state. The receiver of an EAT MUST NOT assume that debug is turned off in a submodule because there is a claim indicating it is turned off in a superior module.

An entity may have multiple debug facilities. The use of plural in the description of the states refers to that, not to any aggregation or inheritance.

The architecture of some chips or devices may be such that a debug facility operates for the whole chip or device. If the EAT for such a chip includes submodules, then each submodule should independently report the status of the whole-chip or whole-device debug facility. This is the only way the receiver can know the debug status of the submodules since there is no inheritance.

4.2.9.1. Enabled

If any debug facility, even manufacturer hardware diagnostics, is currently enabled, then this level must be indicated.

4.2.9.2. Disabled

This level indicates all debug facilities are currently disabled. It may be possible to enable them in the future. It may also be that they were enabled in the past, but they are currently disabled.

4.2.9.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the entity booted/started.

4.2.9.4. Disabled Permanently

This level indicates all non-manufacturer facilities are permanently disabled such that no end user or developer can enable them. Only the manufacturer indicated in the "oemid" claim can enable them. This also indicates that all debug facilities are currently disabled and have been so since boot/start. If this debug state is reported, the "oemid" claim MUST be present.

4.2.9.5. Disabled Fully and Permanently

This level indicates that all debug facilities for the entity are permanently disabled.

```
$$Claims-Set-Claims //= ( debug-status-label => debug-status-type )

debug-status-type = ds-enabled /
                    disabled /
                    disabled-since-boot /
                    disabled-permanently /
                    disabled-fully-and-permanently

ds-enabled                      = JC< "enabled", 0 >
disabled                        = JC< "disabled", 1 >
disabled-since-boot             = JC< "disabled-since-boot", 2 >
disabled-permanently            = JC< "disabled-permanently", 3 >
disabled-fully-and-permanently =
                       JC< "disabled-fully-and-permanently", 4 >
```

4.2.10. location (Location) Claim

The "location" claim gives the geographic position of the entity from which the attestation originates. Latitude, longitude, altitude, accuracy, altitude-accuracy, heading and speed MUST be as defined in the W3C Geolocation API [W3C.GeoLoc] (which, in turn, is based on [WGS84]). If the entity is stationary, the heading is NaN (floating-point not-a-number). Latitude and longitude MUST always be provided. If any other of these values are unknown, they are omitted.

The location may have been cached for a period of time before token creation. For example, it might have been minutes or hours or more since the last contact with a GNSS satellite. Either the timestamp or age data item can be used to quantify the cached period. The timestamp data item is preferred as it a non-relative time. If the entity has no clock or the clock is unset but has a means to measure the time interval between the acquisition of the location and the token creation the age may be reported instead. The age is in seconds.

See location-related privacy considerations in Section 8.2.

```
$$Claims-Set-Claims //= (location-label => location-type)

location-type = {
    latitude => number,
    longitude => number,
    ? altitude => number,
    ? accuracy => number,
    ? altitude-accuracy => number,
    ? heading => number,
    ? speed => number,
    ? timestamp => ~time-int,
    ? age => uint
}

latitude          = JC< "latitude",          1 >
longitude         = JC< "longitude",         2 >
altitude          = JC< "altitude",          3 >
accuracy          = JC< "accuracy",          4 >
altitude-accuracy = JC< "altitude-accuracy", 5 >
heading           = JC< "heading",           6 >
speed             = JC< "speed",             7 >
timestamp         = JC< "timestamp",         8 >
age               = JC< "age",               9 >
```

4.2.11. uptime (Uptime) Claim

The "uptime" claim contains the number of seconds that have elapsed since the entity or submodule was last booted.

```
$$Claims-Set-Claims //= (uptime-label => uint)
```

4.2.12. bootcount (Boot Count) Claim

The "bootcount" claim contains a count of the number times the entity or submodule has been booted. Support for this claim requires a persistent storage on the device.

```
$$Claims-Set-Claims //= (boot-count-label => uint)
```

4.2.13. bootseed (Boot Seed) Claim

The "bootseed" claim contains a value created at system boot time that allows differentiation of attestation reports from different boot sessions of a particular entity (e.g., a certain UEID).

This value is usually public. It is not a secret and MUST NOT be used for any purpose that a secret seed is needed, such as seeding a random number generator.

There are privacy considerations for this claim. See Section 8.3.

```
$$Claims-Set-Claims //=  (boot-seed-label => binary-data)
```

4.2.14. dloas (Digital Letters of Approval) Claim

The "dloas" claim conveys one or more Digital Letters of Approval (DLOAs). A DLOA [DLOA] is a document that describes a certification that an entity has received. Examples of certifications represented by a DLOA include those issued by Global Platform and those based on Common Criteria. The DLOA is unspecific to any particular certification type or those issued by any particular organization.

This claim is typically issued by a verifier, not an attester. Verifiers MUST NOT issue this claim unless the entity has received the certification indicated by the DLOA.

This claim MAY contain more than one DLOA. If multiple DLOAs are present, verifiers MUST NOT issue this claim unless the entity has received all of the certifications.

DLOA documents are always fetched from a registrar that stores them. This claim contains several data items used to construct a Uniform Resource Locator (URL) for fetching the DLOA from the particular registrar.

This claim MUST be encoded as an array with either two or three elements. The first element MUST be the URL for the registrar. The second element MUST be a platform label indicating which platform was certified. If the DLOA applies to an application, then the third element is added which MUST be an application label. The method of constructing the registrar URL, platform label and possibly application label is specified in [DLOA].

The retriever of a DLOA MUST follow the recommendation in [DLOA] and use TLS or some other means to be sure the DLOA registrar they are accessing is authentic. The platform and application labels in the claim indicate the correct DLOA for the entity.

```
$$Claims-Set-Claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: general-uri
    dloa_platform_label: text
    ? dloa_application_label: text
]
```

4.2.15. manifests (Software Manifests) Claim

The "manifests" claim contains descriptions of software present on the entity. These manifests are installed on the entity when the software is installed or are created as part of the installation process. Installation is anything that adds software to the entity, possibly factory installation, the user installing elective applications and so on. The defining characteristic of a manifest is that it is created by the software manufacturer. The purpose of this claim is to relay unmodified manifests to the verifier and possibly to the relying party.

Some manifests are signed by their software manufacturer independently, and some are not either because they do not support signing or the manufacturer chose not to sign them. For example, a CoSWID might be signed independently before it is included in an EAT. When signed manifests are put into an EAT, the manufacturer's signature SHOULD be included even though an EAT's signature will also cover the manifest. This allows the receiver to directly verify the manufacturer-originated manifest.

This claim allows multiple manifest formats. For example, the manifest may be a CBOR-encoded CoSWID, an XML-encoded SWID or other. Identification of the type of manifest is always by a Constrained Application Protocol (CoAP) Content-Format integer [RFC7252]. If there is no CoAP identifier registered for a manifest format, one MUST be registered.

This claim MUST be an array of one or more manifests. Each manifest in the claim MUST be an array of two. The first item in the array of two MUST be an integer CoAP Content-Format identifier. The second item is MUST be the actual manifest.

In JSON-encoded tokens the manifest, whatever encoding it is, MUST be placed in a text string. When a non-text encoded manifest like a CBOR-encoded CoSWID is put in a JSON-encoded token, the manifest MUST be base-64 encoded.

This claim allows for multiple manifests in one token since multiple software packages are likely to be present. The multiple manifests MAY be of different encodings. In some cases EAT submodules may be used instead of the array structure in this claim for multiple manifests.

A CoSWID manifest MUST be a payload CoSWID, not an evidence CoSWID. These are defined in [RFC9393].

A Software Updates for Internet of Things (SUIT) Manifest [SUIT.Manifest] may be used.

This claim is extensible for use of manifest formats beyond those mentioned in this document. No particular manifest format is preferred. For manifest interoperability, an EAT profile as defined in Section 6, should be used to specify which manifest format(s) are allowed.

```
$$Claims-Set-Claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ manifest-format]

manifest-format = [
    content-type:   coap-content-format,
    content-format: JC< $manifest-body-json,
                        $manifest-body-cbor >
]

$manifest-body-cbor /= bytes .cbor untagged-coswid
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= bytes .cbor SUIT_Envelope
$manifest-body-json /= base64-url-text
```

4.2.16. measurements (Measurements) Claim

The "measurements" claim contains descriptions, lists, evidence or measurements of the software that exists on the entity or any other measurable subsystem of the entity (e.g. hash of sections of a file system or non-volatile memory). The defining characteristic of this claim is that its contents are created by processes on the entity that inventory, measure or otherwise characterize the software on the entity. The contents of this claim do not originate from the manufacturer of the measurable subsystem (e.g. developer of a software library).

This claim can be a [RFC9393]. When the CoSWID format is used, it MUST be an evidence CoSWID, not a payload CoSWID.

Formats other than CoSWID MAY be used. The identification of format is by CoAP Content Format, the same as the "manifests" claim in Section 4.2.15.

```
$$Claims-Set-Claims //= (
    measurements-label => measurements-type
)

measurements-type = [+ measurements-format]

measurements-format = [
    content-type:   coap-content-format,
    content-format: JC< $measurements-body-json,
                        $measurements-body-cbor >
]

$measurements-body-cbor /= bytes .cbor untagged-coswid
$measurements-body-json /= base64-url-text
```

4.2.17. measres (Software Measurement Results) Claim

The "measres" claim is a general-purpose structure for reporting comparison of measurements to expected reference values. This claim provides a simple standard way to report the result of a comparison as success, failure, fail to run, and absence.

It is the nature of measurement systems that they are specific to the operating system, software and hardware of the entity that is being measured. It is not possible to standardize what is measured and how it is measured across platforms, OS's, software and hardware. The recipient must obtain the information about what was measured and what it indicates for the characterization of the security of the entity from the provider of the measurement system. What this claim provides is a standard way to report basic success or failure of the measurement. In some use cases it is valuable to know if measurements succeeded or failed in a general way even if the details of what was measured is not characterized.

This claim MAY be generated by the verifier and sent to the relying party. For example, it could be the results of the verifier comparing the contents of the "measurements" claim, Section 4.2.16, to reference values.

This claim MAY also be generated on the entity if the entity has the ability for one subsystem to measure and evaluate another subsystem. For example, a TEE might have the ability to measure the software of the rich OS and may have the reference values for the rich OS.

Within an entity, attestation target or submodule, multiple results can be reported. For example, it may be desirable to report the results for measurements of the file system, chip configuration, installed software, running software and so on.

Note that this claim is not for reporting the overall result of a verifier. It is solely for reporting the result of comparison to reference values.

An individual measurement result (individual-result) is an array consisting of two elements, an identifier of the measurement (result-id) and an enumerated type of the result (result). Different measurement systems will measure different things and perhaps measure the same thing in different ways. It is up to each measurement system to define identifiers (result-id) for the measurements it reports.

Each individual measurement result is part of a group that may contain many individual results. Each group has a text string that names it, typically the name of the measurement scheme or system.

The claim itself consists of one or more groups.

The values for the results enumerated type are as follows:

1 -- comparison successful: Indicates successful comparison to reference values.

2 -- comparison fail: The comparison was completed and did not compare correctly to the reference values.

3 -- comparison not run: The comparison was not run. This includes error conditions such as running out of memory.

4 -- measurement absent: The particular measurement was not available for comparison.

```
$$Claims-Set-Claims //= (
    measurement-results-label =>
        [ + measurement-results-group ] )

measurement-results-group = [
    measurement-system: tstr,
    measurement-results: [ + individual-result ]
]

individual-result = [
    result-id:  tstr / binary-data,
    result:     result-type,
]

result-type = comparison-successful /
              comparison-fail /
              comparison-not-run /
              measurement-absent

comparison-successful    = JC< "success",       1 >
comparison-fail          = JC< "fail",          2 >
comparison-not-run       = JC< "not-run",       3 >
measurement-absent       = JC< "absent",        4 >
```

4.2.18. submods (Submodules)

Some devices are complex and have many subsystems. A mobile phone is a good example. It may have subsystems for communications (e.g., Wi-Fi and cellular), low-power audio and video playback, multiple security-oriented subsystems like a TEE and a Secure Element, and etc. The claims for a subsystem can be grouped together in a submodule.

Submodules may be used in either evidence or attestation results.

Because system architecture will vary greatly from use case to use case, there are no set requirements for what a submodule represents either in evidence or in attestation results. Profiles, Section 6, may wish to impose requirements. An attester that outputs evidence with submodules should document the semantics it associates with particular submodules for the verifier. Likewise, a verifier that outputs attestation results with submodules should document the semantics it associates with the submodules for the relying party.

A submodule claim is a map that holds some number of submodules. Each submodule is named by its label in the submodule claim map. The value of each entry in a submodule may be a Claims-Set, nested token or Detached-Submodule-Digest. This allows for the submodule to serve as its own attester or not and allows for claims for each submodule to be represented directly or indirectly, i.e., detached.

A submodule may include a submodule, allowing for arbitrary levels of nesting. However, submodules do not inherit anything from the containing token and must explicitly include all claims. Submodules may contain claims that are present in any surrounding token or submodule. For example, the top-level of the token may have a UEID, a submodule may have a different UEID and a further subordinate submodule may also have a UEID.

The following sub-sections define the three types for representing submodules:

* A submodule Claims-Set

* The digest of a detached Claims-Set

* A nested token, which can be any EAT

The Submodule type definition and Nested-Token type definition vary with the type of encoding. The definitions for CBOR-encoded EATs are as follows:

```
Nested-Token = CBOR-Nested-Token

CBOR-Nested-Token =
    JSON-Token-Inside-CBOR-Token /
    CBOR-Token-Inside-CBOR-Token

CBOR-Token-Inside-CBOR-Token = bstr .cbor $EAT-CBOR-Tagged-Token

JSON-Token-Inside-CBOR-Token = tstr

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / CBOR-Nested-Token /
            Detached-Submodule-Digest
```

The Submodule and Nested-Token definitions for JSON-encoded EATs is as below. This difference in definitions vs. CBOR is necessary because JSON has no tag mechanism and no byte string type to help indicate the nested token is CBOR.

```
Nested-Token = JSON-Selector

JSON-Selector = $JSON-Selector

$JSON-Selector /= [type: "JWT", nested-token: JWT-Message]
$JSON-Selector /= [type: "CBOR", nested-token: CBOR-Token-Inside-JSON-Token]
$JSON-Selector /= [type: "BUNDLE", nested-token: Detached-EAT-Bundle]
$JSON-Selector /= [type: "DIGEST", nested-token: Detached-Submodule-Digest]

CBOR-Token-Inside-JSON-Token = base64-url-text

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / JSON-Selector
```

The Detached-Submodule-Digest type is defined as follows:

```
Detached-Submodule-Digest = [
   hash-algorithm : text / int,
   digest         : binary-data
]
```

Nested tokens can be one of three types as defined in this document or types standardized in follow-on documents (e.g., [UCCS]). Nested tokens are the only mechanism by which JSON can be embedded in CBOR and vice versa.

The addition of further types is accomplished by augmenting the $EAT-CBOR-Tagged-Token socket or the $JSON-Selector socket.

When decoding a JSON-encoded EAT, the type of submodule is determined as follows. A JSON object indicates the submodule is a Claims-Set. In all other cases, it is a JSON-Selector, which is an array of two elements that indicates whether the submodule is a nested token or a Detached-Submodule-Digest.The first element in the array indicates the type present in the second element. If the value is JWT, CBOR, BUNDLE or a future-standardized token types, e.g., [UCCS], the submodule is a nested token of the indicated type, i.e., JWT-Message, CBOR-Token-Inside-JSON-Token, Detached-EAT-Bundle, or a future type. If the value is "DIGEST", the submodule is a Detached-Submodule-Digest. Any other value indicates a standardized extension to this specification.

When decoding a CBOR-encoded EAT, the CBOR item type indicates the type of the submodule as follows. A map indicates a CBOR-encoded submodule Claims-Set. An array indicates a CBOR-encoded Detached-Submodule-Digest. A byte string indicates a CBOR-encoded CBOR-Nested-Token. A text string indicates a JSON-encoded JSON-Selector. Where JSON-Selector is used in a CBOR-encoded EAT, the "DIGEST" type and corresponding Detached-Submodule-Digest type MUST NOT be used.

The type of a CBOR-encoded nested token is always determined by the CBOR tag encountered after the byte string wrapping is removed in a CBOR-encoded enclosing token or after the base64 wrapping is removed in JSON-encoded enclosing token.

The type of a JSON-encoded nested token is always determined by the string name in JSON-Selector and is always JWT, BUNDLE or a new name standardized outside this document for a further type (e.g., UCCS). This string name may also be CBOR to indicate the nested token is CBOR-encoded.

"JWT": The second array item MUST be a JWT formatted according to [RFC7519]

"CBOR": The second array item MUST be some base64url-encoded CBOR that is a tag, typically a CWT or CBOR-encoded detached EAT bundle

"BUNDLE": The second array item MUST be a JSON-encoded Detached EAT Bundle as defined in this document.

"DIGEST": The second array item MUST be a JSON-encoded Detached-Submodule-Digest as defined in this document.

As noted elsewhere, additional EAT types may be defined by a standards action. New type specifications MUST address the integration of the new type into the Submodule claim type for submodules.

4.2.18.1. Submodule Claims-Set

The Claims-Set type provides a means of representing claims from a submodule that does not have its own attesting environment, i.e., it has no keys distinct from the attester producing the surrounding token. Claims are represented as a Claims-Set. Submodule claims represented in this way are secured by the same mechanism as the enclosing token (e.g., it is signed by the same attestation key).

The encoding of a submodule Claims-Set MUST be the same as the encoding as the surrounding EAT, e.g., all submodule Claims-Sets in a CBOR-encoded token must be CBOR-encoded.

4.2.18.2. Detached Submodule Digest

The Detached-Submodule-Digest type is similar to a submodule Claims-Set, except a digest of the Claims-Set is included in the claim with the Claims-Set contents conveyed separately. The separately-conveyed Claims-Set is called a detached claims set. The input to the digest algorithm is directly the CBOR or JSON-encoded Claims-Set for the submodule. There is no byte-string wrapping or base 64 encoding.

The data type for this type of submodule is an array consisting of two data items: an algorithm identifier and a byte string containing the digest. The hash algorithm identifier is always from the COSE Algorithm registry, [IANA.COSE.Algorithms]. Either the integer or string identifier may be used. The hash algorithm identifier is never from the JOSE Algorithm registry.

A detached EAT bundle, described in Section 5, may be used to convey detached claims sets and the EAT containing the corresponding detached digests. EAT, however, doesn’t require use of a detached EAT bundle. Any other protocols may be used to convey detached claims sets and the EAT containing the corresponding detached digests. Detached Claims-Sets must not be modified in transit, else validation will fail.

4.2.18.3. Nested Tokens

The CBOR-Nested-Token and JSON-Selector types provide a means of representing claims from a submodule that has its own attesting environment, i.e., it has keys distinct from the attester producing the surrounding token. Claims are represented in a signed EAT token.

Inclusion of a signed EAT as a claim cryptographically binds the EAT to the surrounding token. If it was conveyed in parallel with the surrounding token, there would be no such binding and attackers could substitute a good attestation from another device for the attestation of an errant subsystem.

A nested token need not use the same encoding as the enclosing token. This enables composite devices to be built without regards to the encoding used by components. Thus, a CBOR-encoded EAT can have a JSON-encoded EAT as a nested token and vice versa.

4.3. Claims Describing the Token

The claims in this section provide meta data about the token they occur in. They do not describe the entity. They may appear in evidence or attestation results.

4.3.1. iat (Timestamp) Claim

The "iat" claim defined in CWT and JWT is used to indicate the date-of-creation of the token, the time at which the claims are collected and the token is composed and signed.

The data for some claims may be held or cached for some period of time before the token is created. This period may be long, even days. Examples are measurements taken at boot or a geographic position fix taken the last time a satellite signal was received. There are individual timestamps associated with these claims to indicate their age is older than the "iat" timestamp.

CWT allows the use of floating-point for this claim. EAT disallows the use of floating-point. An EAT token MUST NOT contain an "iat" claim in floating-point format. Any recipient of a token with a floating-point format "iat" claim MUST consider it an error.

A 64-bit integer representation of the CBOR epoch-based time [RFC8949] used by this claim can represent a range of +/- 500 billion years, so the only point of a floating-point timestamp is to have precession greater than one second. This is not needed for EAT.

4.3.2. eat_profile (EAT Profile) Claim

See Section 6 for the detailed description of an EAT profile.

The "eat_profile" claim identifies an EAT profile by either a Uniform Resource Identifier (URI) or an Object Identifier (OID). Typically, the URI will reference a document describing the profile. An OID is just a unique identifier for the profile. It may exist anywhere in the OID tree. There is no requirement that the named document be publicly accessible. The primary purpose of the "eat_profile" claim is to uniquely identify the profile even if it is a private profile.

The OID is always absolute and never relative.

See Section 7.2.1 for OID and URI encoding.

```
$$Claims-Set-Claims //= (profile-label => general-uri / general-oid)
```

4.3.3. intuse (Intended Use) Claim

EATs may be employed in the context of several different applications. The "intuse" claim provides an indication to an EAT consumer about the intended usage of the token. This claim can be used as a way for an application using EAT to internally distinguish between different ways it utilizes EAT. 5 possible values for "intuse" are currently defined, but an IANA registry can be created in the future to extend these values based on new use cases of EAT.

1 -- Generic: Generic attestation describes an application where the EAT consumer requires the most up-to-date security assessment of the attesting entity. It is expected that this is the most commonly-used application of EAT.

2-- Registration: Entities that are registering for a new service may be expected to provide an attestation as part of the registration process. This "intuse" setting indicates that the attestation is not intended for any use but registration.

3 -- Provisioning: Entities may be provisioned with different values or settings by an EAT consumer. Examples include key material or device management trees. The consumer may require an EAT to assess entity security state of the entity prior to provisioning.

4 -- Certificate Issuance: Certification Authorities (CAs) may require attestation results (which in a background check model might require receiving evidence to be passed to a verifier) to make decisions about the issuance of certificates. An EAT may be used as part of the certificate signing request (CSR).

5 -- Proof-of-Possession: An EAT consumer may require an attestation as part of an accompanying proof-of-possession (PoP) application. More precisely, a PoP transaction is intended to provide to the recipient cryptographically-verifiable proof that the sender has possession of a key. This kind of attestation may be necessary to verify the security state of the entity storing the private key used in a PoP application.

```
$$Claims-Set-Claims //= ( intended-use-label => intended-use-type )

intended-use-type = generic /
                    registration /
                    provisioning /
                    csr /
                    pop

generic      = JC< "generic",      1 >
registration = JC< "registration", 2 >
provisioning = JC< "provisioning", 3 >
csr          = JC< "csr",          4 >
pop          = JC< "pop",          5 >
```

5. Detached EAT Bundles

A detached EAT bundle is a message to convey an EAT plus detached claims sets secured by that EAT. It is a top-level message like a CWT or JWT. It can occur in any place that a CWT or JWT occurs, for example as a submodule nested token as defined in Section 4.2.18.3.

A detached EAT bundle may be either CBOR or JSON-encoded.

A detached EAT bundle consists of two parts.

The first part is an encoded EAT as follows:

* MUST have at least one submodule that is a detached submodule digest as defined in Section 4.2.18.2

* MAY be either CBOR or JSON-encoded and doesn’t have to the the same as the encoding of the bundle

* MAY be a CWT, or JWT or some future-defined token type, but MUST NOT be a detached EAT bundle

* MUST be authenticity and integrity protected

The same mechanism for distinguishing the type for nested token submodules is employed here.

The second part is a map/object as follows:

* MUST be a Claims-Set

* MUST use the same encoding as the bundle

* MUST be wrapped in a byte string when the encoding is CBOR and be base64url-encoded when the encoding is JSON

For CBOR-encoded detached EAT bundles, tag TBD602 can be used to identify it. The standard rules apply for use or non-use of a tag. When it is sent as a submodule, it is always sent as a tag to distinguish it from the other types of nested tokens.

The digests of the detached claims sets are associated with detached Claims-Sets by label/name. It is up to the constructor of the detached EAT bundle to ensure the names uniquely identify the detached claims sets. Since the names are used only in the detached EAT bundle, they can be very short, perhaps one byte.

```
BUNDLE-Messages = BUNDLE-Tagged-Message / BUNDLE-Untagged-Message

BUNDLE-Tagged-Message   = #6.TBD602(BUNDLE-Untagged-Message)
BUNDLE-Untagged-Message = Detached-EAT-Bundle

Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => JC<json-wrapped-claims-set,
                     cbor-wrapped-claims-set>
    }
]

json-wrapped-claims-set = base64-url-text

cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

6. Profiles

EAT makes normative use of CBOR, JSON, COSE, JOSE, CWT and JWT. Most of these have implementation options to accommodate a range of use cases.

For example, COSE doesn’t require a particular set of cryptographic algorithms so as to accommodate different usage scenarios and evolution of algorithms over time. Section 10 of [RFC9052] describes the profiling considerations for COSE.

The use of encryption is optional for both CWT and JWT. Section 8 of [RFC7519] describes implementation requirement and recommendations for JWT.

Similarly, CBOR provides indefinite length encoding, which is not commonly used, but valuable for very constrained devices. For EAT itself, in a particular use case some claims will be used and others will not. Section 4 of [RFC8949] describes serialization considerations for CBOR.

For example a mobile phone use case may require the device make and model, and prohibit UEID and location for privacy reasons. The general EAT standard retains all this flexibility because it too is aimed to accommodate a broad range of use cases.

It is necessary to explicitly narrow these implementation options to guarantee interoperability. EAT chooses one general and explicit mechanism, the profile, to indicate the choices made for these implementation options for all aspects of the token.

Below is a list of the various issues that should be addressed by a profile.

The "eat_profile" claim in Section 4.3.2 provides a unique identifier for the profile a particular token uses.

A profile can apply to evidence or to attestation results or both.

6.1. Format of a Profile Document

A profile document doesn’t have to be in any particular format. It may be simple text, something more formal or a combination.

A profile may define, and possibly register, one or more new claims if needed. A profile may also reuse one or more already defined claims, either as-is or with values constrained to a subset or subrange.

6.2. Full and Partial Profiles

For a "full" profile, the receiver will be able to decode and verify every possible EAT sent when a sender and receiver both adhere to it. For a "partial" profile, there are still some protocol options left undecided.

For example, a profile that allows the use of signing algorithms by the sender that the receiver is not required to support is a partial profile. The sender might choose a signing algorithm that some receivers don’t support.

Full profiles MUST be complete such that a complying receiver can decode, verify and check for freshness every EAT created by a complying sender. A full profile MAY or MAY NOT require the receiver to fully handle every claim in an EAT from a complying sender. Profile specifications may assume the receiver has access to the necessary verification keys or may go into specific detail on the means to access verification keys.

The "eat_profile" claim MUST NOT be used to identify partial profiles.

While fewer profiles are preferrable, sometimes several may be needed for a use case. One approach to handling variation in devices might be to define several full profiles that are variants of each other. It is relatively easy and inexpensive to define profiles as they don't have to be standards track and don't have to be registered anywhere. For example, flexibility for post-quantum algorithms can be handled as follows. First, define a full profile for a set of non-post-quantum algorithms for current use. Then, when post-quantum algorithms are settled, define another full profile derived from the first.

6.3. List of Profile Issues

The following is a list of EAT, CWT, JWT, COSE, JOSE and CBOR options that a profile should address.

6.3.1. Use of JSON, CBOR or both

A profile should specify whether CBOR, JSON or both may be sent. A profile should specify that the receiver can accept all encodings that the sender is allowed to send.

This should be specified for the top-level and all nested tokens. For example, a profile might require all nested tokens to be of the same encoding of the top level token.

6.3.2. CBOR Map and Array Encoding

A profile should specify whether definite-length arrays/maps, indefinite-length arrays/maps or both may be sent. A profile should specify that the receiver be able to accept all length encodings that the sender is allowed to send.

This applies to individual EAT claims, CWT and COSE parts of the implementation.

For most use cases, specifying that only definite-length arrays/maps may be sent is suitable.

6.3.3. CBOR String Encoding

A profile should specify whether definite-length strings, indefinite-length strings or both may be sent. A profile should specify that the receiver be able to accept all types of string encodings that the sender is allowed to send.

For most use cases, specifying that only definite-length strings may be sent is suitable.

6.3.4. CBOR Preferred Serialization

A profile should specify whether or not CBOR preferred serialization must be sent or not. A profile should specify the receiver be able to accept preferred and/or non-preferred serialization so it will be able to accept anything sent by the sender.

6.3.5. CBOR Tags

The profile should specify whether the token should be a CWT Tag or not.

When COSE protection is used, the profile should specify whether COSE tags are used or not. Note that RFC 8392 requires COSE tags be used in a CWT tag.

Often a tag is unnecessary because the surrounding or carrying protocol identifies the object as an EAT.

6.3.6. COSE/JOSE Protection

COSE and JOSE have several options for signed, MACed and encrypted messages. JWT may use the JOSE NULL protection option. It is possible to implement no protection, sign only, MAC only, sign then encrypt and so on. All combinations allowed by COSE, JOSE, JWT, and CWT are allowed by EAT.

A profile should specify all signing, encryption and MAC message formats that may be sent. For example, a profile might allow only COSE_Sign1 to be sent. For another example, a profile might allow COSE_Sign and COSE_Encrypt to be sent to carry multiple signatures for post quantum cryptography and to use encryption to provide confidentiality.

A profile should specify the receiver accepts all message formats that are allowed to be sent.

When both signing and encryption are allowed, a profile should specify which is applied first.

6.3.7. COSE/JOSE Algorithms

See the section on "Application Profiling Considerations" in [RFC9052] for a discussion on selection of cryptographic algorithms and related issues.

The profile MAY require the protocol or system using EAT provide an algorithm negotiation mechanism.

If not, The profile document should list a set of algorithms for each COSE and JOSE message type allowed by the profile per Section 6.3.6. The verifier should implement all of them. The attester may implement any of them it wishes, possibly just one for each message type.

If detached submodule digests are used the profile should address the determination of the hash algorithm(s) for the digests.

6.3.8. Detached EAT Bundle Support

A profile should specify whether or not a detached EAT bundle (Section 5) can be sent. A profile should specify that a receiver be able to accept a detached EAT bundle if the sender is allowed to send it.

6.3.9. Key Identification

A profile should specify what must be sent to identify the verification, decryption or MAC key or keys. If multiple methods of key identification may be sent, a profile should require the receiver support them all.

Appendix F describes a number of methods for identifying verification keys. When encryption is used, there are further considerations. In some cases key identification may be very simple and in others involve multiple components. For example, it may be simple through use of COSE key ID or it may be complex through use of an X.509 certificate hierarchy.

While not always possible, a profile should specify or make reference to, a full end-end specification for key identification. For example, a profile should specify in full detail how COSE key IDs are

to be created, their lifecycle and such rather than just specifying that a COSE key ID be used. For example, a profile should specify the full details of an X.509 hierarchy including extension processing, algorithms allowed and so on rather than just saying X.509 certificates are used.

6.3.10. Endorsement Identification

Similar to, or perhaps the same as verification key identification, the profile may wish to specify how endorsements are to be identified. However note that endorsement identification is optional, whereas key identification is not.

6.3.11. Freshness

Security considerations, see Section 9.3, require a mechanism to provide freshness. This may be the EAT nonce claim in Section 4.1, or some claim or mechanism defined outside this document. The section on freshness in [RFC9334] describes several options. A profile should specify which freshness mechanism or mechanisms can be used.

If the EAT nonce claim is used, a profile should specify whether multiple nonces may be sent. If a profile allows multiple nonces to be sent, it should require the receiver to process multiple nonces.

6.3.12. Claims Requirements

A profile may define new claims that are not defined in this document.

This document requires an EAT receiver must accept tokens with claims it does not understand. A profile for a specific use case may reverse this and allow a receiver to reject tokens with claims it does not understand. A profile for a specific use case may specify that specific claims are prohibited.

A profile for a specific use case may modify this and specify that some claims are required.

A profile may constrain the definition of claims that are defined in this document or elsewhere. For example, a profile may require the EAT nonce be a certain length or the "location" claim always include the altitude.

Some claims are "pluggable" in that they allow different formats for their content. The "manifests" claim (Section 4.2.15) along with the measurement and "measurements" (Section 4.2.16) claims are examples

of this, allowing the use of CoSWID, SUIT Manifest and other formats. A profile should specify which formats are allowed to be sent, with the assumption that the corresponding CoAP content types have been registered. A profile should require the receiver to accept all formats that are allowed to be sent.

Further, if there is variation within a format that is allowed, the profile should specify which variations can be sent. For example, there are variations in the CoSWID format. A profile that require the receiver to accept all variations that are allowed to be sent.

6.4. The Constrained Device Standard Profile

It is anticipated that there will be many profiles defined for EAT for many different use cases. This section gives a normative definition of one profile that is good for many constrained device use cases.

The identifier for this profile is "urn:ietf:rfc:rfcTBD".

// RFC Editor: please replace rfcTBD with this RFC number and remove
// this note.

Issue	Profile Definition
CBOR/JSON	CBOR MUST be used
CBOR Encoding	Definite length maps and arrays MUST be used
CBOR Encoding	Definite length strings MUST be used
CBOR Serialization	Preferred serialization MUST be used
COSE Protection	COSE_Sign1 MUST be used
Algorithms	The receiver MUST accept ES256, ES384 and ES512; the sender MUST send one of these
Detached EAT Bundle Usage	Detached EAT bundles MUST not be sent with this profile
Verification Key Identification	Either the COSE kid or the UEID MUST be used to identify the verification key. If both are present, the kid takes precedence. (It is assumed the receiver has access to a database of trusted verification keys which allows lookup of the verification key ID; the key format and means of distribution are beyond the scope of this profile)
Endorsements	This profile contains no endorsement identifier
Freshness	A new single unique nonce MUST be used for every token request
Claims	No requirement is made on the presence or absence of claims other than requiring an EAT nonce. As per general EAT rules, the receiver MUST NOT error out on claims it doesn't understand.

Table 2: Constrained Device Profile Definition

Any profile with different requirements than those above MUST have a different profile identifier.

Note that many claims can be present for tokens conforming to this profile, even claims not defined in this document. Note also that even slight deviation from the above requirements is considered a different profile that MUST have a different identifier. For example, if a kid (key identifier) or UEID is not used for key identification, it is not in conformance with this profile. For another example, requiring the presence of some claim is also not in conformance and requires another profile.

Derivations of this profile are encouraged. For example another profile may be simply defined as The Constrained Device Standard Profile plus the requirement for the presence of claim xxxx and claim yyyy.

7. Encoding and Collected CDDL

An EAT is fundamentally defined using CDDL. This document specifies how to encode the CDDL in CBOR or JSON. Since CBOR can express some things that JSON can't (e.g., tags) or that are expressed differently (e.g., labels) there is some CDDL that is specific to the encoding.

7.1. Claims-Set and CDDL for CWT and JWT

CDDL was not used to define CWT or JWT. It was not available at the time.

This document defines CDDL for both CWT and JWT. This document does not change the encoding or semantics of anything in a CWT or JWT.

A Claims-Set is the central data structure for EAT, CWT and JWT. It holds all the claims and is the structure that is secured by signing or other means. It is not possible to define EAT, CWT, or JWT in CDDL without it. The CDDL definition of Claims-Set here is applicable to EAT, CWT and JWT.

This document specifies how to encode a Claims-Set in CBOR or JSON.

With the exception of nested tokens and some other externally defined structures (e.g., SWIDs) an entire Claims-Set must be in encoded in either CBOR or JSON, never a mixture.

CDDL for the seven claims defined by [RFC8392] and [RFC7519] is included here.

7.2. Encoding Data Types

This makes use of the types defined in [RFC8610] Appendix D, Standard Prelude.

7.2.1. Common Data Types

time-int is identical to the epoch-based time, but disallows floating-point representation.

For CBOR-encoded tokens, OIDs are specified using the CDDL type name "oid" from [RFC9090]. They are encoded without the tag number. For JSON-encoded tokens, OIDs are a text string in the common form of "nn.nn.nn...".

Unless expliclity indicated, URIs are not the URI tag defined in [RFC8949]. They are just text strings that contain a URI conforming to the format defined in [RFC3986].

```
time-int = #6.1(int)

binary-data = JC< base64-url-text, bstr>

base64-url-text = tstr .regexp "[A-Za-z0-9_-]+"

general-oid = JC< json-oid, ~oid >

json-oid = tstr .regexp "([0-2])((\\.0)|(\\.[1-9][0-9]*))*"

general-uri = JC< text, ~uri >

coap-content-format = uint .le 65535
```

7.2.2. JSON Interoperability

JSON should be encoded per [RFC8610], Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

* bstr -- MUST be base64url-encoded

* time -- MUST be encoded as NumericDate as described in Section 2 of [RFC7519].

* string-or-uri -- MUST be encoded as StringOrURI as described in Section 2 of [RFC7519].

* uri -- MUST be a URI [RFC3986].

* oid -- MUST be encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1") [RFC2252].

The CDDL generic "JC<>" is used in most places where there is a variance between CBOR and JSON. The first argument is the CDDL for JSON and the second is CDDL for CBOR.

7.2.3. Labels

Most map labels, Claims-Keys, Claim-Names and enumerated-type values are integers for CBOR-encoded tokens and strings for JSON-encoded tokens. When this is the case the "JC<>" CDDL construct is used to give both the integer and string values.

7.2.4. CBOR Interoperability

CBOR allows data items to be serialized in more than one form to accommodate a variety of use cases. This is addressed in Section 6.

7.3. Collected CDDL

7.3.1. Payload CDDL

This CDDL defines all the EAT Claims that are added to the main definition of a Claim-Set in Appendix D. Claims-Set is the payload for CWT, JWT and potentially other token types. This is for both CBOR and JSON. When there is variation between CBOR and JSON, the JC<> CDDL generic defined in Appendix D. Note that the JC<> generic uses the CDDL ".feature" control operator defined in [RFC9165].

This CDDL uses, but doesn’t define Submodule or nested tokens because the definition for these types varies between CBOR and JSON and the JC<> generic can’t be used to define it. The submodule claim is the one place where a CBOR token can be nested inside a JSON token and vice versa. Encoding-specific definitions are provided in the following sections.

```
time-int = #6.1(int)

binary-data = JC< base64-url-text, bstr>

base64-url-text = tstr .regexp "[A-Za-z0-9_-]+"

general-oid = JC< json-oid, ~oid >

json-oid = tstr .regexp "([0-2])((\\.0)|(\\.[1-9][0-9]*))*"

general-uri = JC< text, ~uri >

coap-content-format = uint .le 65535
```

```
$$Claims-Set-Claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

nonce-type = JC< tstr .size (8..88), bstr .size (8..64)>


$$Claims-Set-Claims //= (ueid-label => ueid-type)

ueid-type = JC<base64-url-text .size (10..44) , bstr .size (7..33)>

$$Claims-Set-Claims //= (sueids-label => sueids-type)

sueids-type = {
    + tstr => ueid-type
}

$$Claims-Set-Claims //= (
    oemid-label => oemid-pen / oemid-ieee / oemid-random
)

oemid-pen = int

oemid-ieee = JC<oemid-ieee-json, oemid-ieee-cbor>
oemid-ieee-cbor = bstr .size 3
oemid-ieee-json = base64-url-text .size 4

oemid-random = JC<oemid-random-json, oemid-random-cbor>
oemid-random-cbor = bstr .size 16
oemid-random-json = base64-url-text .size 24


$$Claims-Set-Claims //=  (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version:  tstr,
    ? scheme:  $version-scheme
]

$$Claims-Set-Claims //= (
    hardware-model-label => hardware-model-type
)

hardware-model-type = JC<base64-url-text .size (4..44),
                         bytes .size (1..32)>

$$Claims-Set-Claims //= ( sw-name-label => tstr )
```

```
$$Claims-Set-Claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version:  tstr
    ? scheme:  $version-scheme
]

$$Claims-Set-Claims //= (oem-boot-label => bool)

$$Claims-Set-Claims //= ( debug-status-label => debug-status-type )

debug-status-type = ds-enabled /
                    disabled /
                    disabled-since-boot /
                    disabled-permanently /
                    disabled-fully-and-permanently

ds-enabled                      = JC< "enabled", 0 >
disabled                        = JC< "disabled", 1 >
disabled-since-boot             = JC< "disabled-since-boot", 2 >
disabled-permanently            = JC< "disabled-permanently", 3 >
disabled-fully-and-permanently =
                       JC< "disabled-fully-and-permanently", 4 >

$$Claims-Set-Claims //= (location-label => location-type)

location-type = {
    latitude => number,
    longitude => number,
    ? altitude => number,
    ? accuracy => number,
    ? altitude-accuracy => number,
    ? heading => number,
    ? speed => number,
    ? timestamp => ~time-int,
    ? age => uint
}

latitude          = JC< "latitude",          1 >
longitude         = JC< "longitude",         2 >
altitude          = JC< "altitude",          3 >
accuracy          = JC< "accuracy",          4 >
altitude-accuracy = JC< "altitude-accuracy", 5 >
heading           = JC< "heading",           6 >
speed             = JC< "speed",             7 >
timestamp         = JC< "timestamp",         8 >
age               = JC< "age",               9 >
```

```
$$Claims-Set-Claims //= (uptime-label => uint)

$$Claims-Set-Claims //=  (boot-seed-label => binary-data)

$$Claims-Set-Claims //= (boot-count-label => uint)

$$Claims-Set-Claims //= ( intended-use-label => intended-use-type )

intended-use-type = generic /
                    registration /
                    provisioning /
                    csr /
                    pop

generic      = JC< "generic",      1 >
registration = JC< "registration", 2 >
provisioning = JC< "provisioning", 3 >
csr          = JC< "csr",          4 >
pop          = JC< "pop",          5 >

$$Claims-Set-Claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: general-uri
    dloa_platform_label: text
    ? dloa_application_label: text
]

$$Claims-Set-Claims //= (profile-label => general-uri / general-oid)

$$Claims-Set-Claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ manifest-format]

manifest-format = [
    content-type:   coap-content-format,
    content-format: JC< $manifest-body-json,
                        $manifest-body-cbor >
]

$manifest-body-cbor /= bytes .cbor untagged-coswid
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= bytes .cbor SUIT_Envelope
```

```
$manifest-body-json /= base64-url-text


$$Claims-Set-Claims //= (
    measurements-label => measurements-type
)

measurements-type = [+ measurements-format]

measurements-format = [
    content-type:   coap-content-format,
    content-format: JC< $measurements-body-json,
                        $measurements-body-cbor >
]

$measurements-body-cbor /= bytes .cbor untagged-coswid
$measurements-body-json /= base64-url-text


$$Claims-Set-Claims //= (
    measurement-results-label =>
        [ + measurement-results-group ] )

measurement-results-group = [
    measurement-system: tstr,
    measurement-results: [ + individual-result ]
]

individual-result = [
    result-id:  tstr / binary-data,
    result:     result-type,
]

result-type = comparison-successful /
              comparison-fail /
              comparison-not-run /
              measurement-absent

comparison-successful    = JC< "success",       1 >
comparison-fail          = JC< "fail",          2 >
comparison-not-run       = JC< "not-run",       3 >
measurement-absent       = JC< "absent",        4 >



Detached-Submodule-Digest = [
   hash-algorithm : text / int,
   digest         : binary-data
```

```
]


BUNDLE-Messages = BUNDLE-Tagged-Message / BUNDLE-Untagged-Message

BUNDLE-Tagged-Message   = #6.TBD602(BUNDLE-Untagged-Message)
BUNDLE-Untagged-Message = Detached-EAT-Bundle

Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => JC<json-wrapped-claims-set,
                     cbor-wrapped-claims-set>
    }
]

json-wrapped-claims-set = base64-url-text

cbor-wrapped-claims-set = bstr .cbor Claims-Set



nonce-label            = JC< "eat_nonce",  10 >
ueid-label             = JC< "ueid",       256 >
sueids-label           = JC< "sueids",     257 >
oemid-label            = JC< "oemid",      258 >
hardware-model-label   = JC< "hwmodel",    259 >
hardware-version-label = JC< "hwversion",  260 >
oem-boot-label         = JC< "oemboot",    262 >
debug-status-label     = JC< "dbgstat",    263 >
location-label         = JC< "location",   264 >
profile-label          = JC< "eat_profile",265 >
submods-label          = JC< "submods",    266 >

uptime-label           = JC< "uptime",     TBD >
boot-seed-label        = JC< "bootseed",   TBD >
intended-use-label     = JC< "intuse",     TBD >
dloas-label            = JC< "dloas",      TBD >
sw-name-label          = JC< "swname",     TBD >
sw-version-label       = JC< "swversion",  TBD >
manifests-label        = JC< "manifests",  TBD >
measurements-label     = JC< "measurements", TBD >
measurement-results-label = JC< "measres" , TBD >
boot-count-label       = JC< "bootcount",  TBD >
```

7.3.2. CBOR-Specific CDDL

```
EAT-CBOR-Token = $EAT-CBOR-Tagged-Token / $EAT-CBOR-Untagged-Token

$EAT-CBOR-Tagged-Token /= CWT-Tagged-Message
$EAT-CBOR-Tagged-Token /= BUNDLE-Tagged-Message

$EAT-CBOR-Untagged-Token /= CWT-Untagged-Message
$EAT-CBOR-Untagged-Token /= BUNDLE-Untagged-Message


Nested-Token = CBOR-Nested-Token

CBOR-Nested-Token =
    JSON-Token-Inside-CBOR-Token /
    CBOR-Token-Inside-CBOR-Token

CBOR-Token-Inside-CBOR-Token = bstr .cbor $EAT-CBOR-Tagged-Token

JSON-Token-Inside-CBOR-Token = tstr

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / CBOR-Nested-Token /
            Detached-Submodule-Digest
```

7.3.3. JSON-Specific CDDL

```
EAT-JSON-Token = $EAT-JSON-Token-Formats

$EAT-JSON-Token-Formats /= JWT-Message
$EAT-JSON-Token-Formats /= BUNDLE-Untagged-Message


Nested-Token = JSON-Selector

JSON-Selector = $JSON-Selector

$JSON-Selector /= [type: "JWT", nested-token: JWT-Message]
$JSON-Selector /= [type: "CBOR", nested-token: CBOR-Token-Inside-JSON-Token]
$JSON-Selector /= [type: "BUNDLE", nested-token: Detached-EAT-Bundle]
$JSON-Selector /= [type: "DIGEST", nested-token: Detached-Submodule-Digest]

CBOR-Token-Inside-JSON-Token = base64-url-text

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / JSON-Selector
```

8. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity; therefore, implementations should consider privacy-preserving options dependent on the usage of the EAT. For example, the location claim might be suppressed in EATs sent to unauthenticated consumers.

8.1. UEID and SUEID Privacy Considerations

A UEID is usually not privacy-preserving. Relying parties receiving tokens that happen to be from a particular entity will be able to know the tokens are from the same entity and be able to identify the entity issuing those tokens.

Thus the use of the claim may violate privacy policies. In other usage situations a UEID will not be allowed for certain products like browsers that give privacy for the end user. It will often be the case that tokens will not have a UEID for these reasons.

An SUEID is also usually not privacy-preserving. In some cases it may have fewer privacy issues than a UEID depending on when and how and when it is generated.

There are several strategies that can be used to still be able to put UEIDs and SUEIDs in tokens:

* The entity obtains explicit permission from the user of the entity to use the UEID/SUEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID/SUEID.

* The UEID/SUEID is used only in a particular context or particular use case. It is used only by one relying party.

* The entity authenticates the relying party and generates a derived UEID/SUEID just for that particular relying party. For example, the relying party could prove their identity cryptographically to the entity, then the entity generates a UEID just for that relying party by hashing a proofed relying party ID with the main entity UEID/SUEID.

Note that some of these privacy preservation strategies result in multiple UEIDs and SUEIDs per entity. Each UEID/SUEID is used in a different context, use case or system on the entity. However, from the view of the relying party, there is just one UEID and it is still globally universal across manufacturers.

8.2. Location Privacy Considerations

Geographic location is most always considered personally identifiable information. Implementers should consider laws and regulations governing the transmission of location data from end user devices to servers and services. Implementers should consider using location management facilities offered by the operating system on the entity generating the attestation. For example, many mobile phones prompt the user for permission before sending location data.

8.3. Boot Seed Privacy Considerations

The "bootseed" claim is effectively a stable entity identifier within a given boot epoch. Therefore, it is not suitable for use in attestation schemes that are privacy-preserving.

8.4. Replay Protection and Privacy

EAT defines the EAT nonce claim for replay protection and token freshness. The nonce claim is based on a value usually derived remotely (outside of the entity). This claim might be used to extract and convey personally identifying information either inadvertently or by intention. For instance, an implementor may choose a nonce equivalent to a username associated with the device (e.g., account login). If the token is inspected by a 3rd-party then this information could be used to identify the source of the token or an account associated with the token. To avoid the conveyance of privacy-related information in the nonce claim, it should be derived using a salt that originates from a true and reliable random number generator or any other source of randomness that would still meet the target system requirements for replay protection and token freshness.

9. Security Considerations

The security considerations provided in Section 8 of [RFC8392] and Section 11 of [RFC7519] apply to EAT in its CWT and JWT form, respectively. Moreover, Chapter 12 of [RFC9334] is also applicable to implementations of EAT. In addition, implementors should consider the following.

9.1. Claim Trustworthiness

This specification defines semantics for each claim. It does not require any particular level of security in the implementation of the claims or even the attester itself. Such specification is far beyond the scope of this document which is about a message format not the security level of an implementation.

The receiver of an EAT comes to know the trustworthiness of the claims in it by understanding the implementation made by the attester vendor and/or understanding the checks and processing performed by the verifier.

For example, this document says that a UEID is permanent and that it must not change, but it doesn't say what degree of attack to change it must be defended.

The degree of security will vary from use case to use case. In some cases the receiver may only need to know something of the implementation such as that it was implemented in a TEE. In other cases the receiver may require the attester be certified by a particular certification program. Or perhaps the receiver is content with very little security.

9.2. Key Provisioning

Private key material can be used to sign and/or encrypt the EAT, or can be used to derive the keys used for signing and/or encryption. In some instances, the manufacturer of the entity may create the key material separately and provision the key material in the entity itself. The manufacturer of any entity that is capable of producing an EAT should take care to ensure that any private key material be suitably protected prior to provisioning the key material in the entity itself. This can require creation of key material in an enclave (see [RFC4949] for definition of "enclave"), secure transmission of the key material from the enclave to the entity using an appropriate protocol, and persistence of the private key material in some form of secure storage to which (preferably) only the entity has access.

9.2.1. Transmission of Key Material

Regarding transmission of key material from the enclave to the entity, the key material may pass through one or more intermediaries. Therefore some form of protection ("key wrapping") may be necessary. The transmission itself may be performed electronically, but can also be done by human courier. In the latter case, there should be minimal to no exposure of the key material to the human (e.g. encrypted portable memory). Moreover, the human should transport the key material directly from the secure enclave where it was created to a destination secure enclave where it can be provisioned.

9.3. Freshness

All EAT use MUST provide a freshness mechanism to prevent replay and related attacks. The extensive discussions on freshness in [RFC9334] including security considerations apply here. The EAT nonce claim, in Section 4.1, is one option to provide freshness.

9.4. Multiple EAT Consumers

In many cases, more than one EAT consumer may be required to fully verify the entity attestation. Examples include individual consumers for nested EATs, or consumers for individual claims with an EAT. When multiple consumers are required for verification of an EAT, it is important to minimize information exposure to each consumer. In addition, the communication between multiple consumers should be secure.

For instance, consider the example of an encrypted and signed EAT with multiple claims. A consumer may receive the EAT (denoted as the "receiving consumer"), decrypt its payload, verify its signature, but then pass specific subsets of claims to other consumers for evaluation ("downstream consumers"). Since any COSE encryption will be removed by the receiving consumer, the communication of claim subsets to any downstream consumer MUST leverage an equivalent communication security protocol (e.g. Transport Layer Security).

However, assume the EAT of the previous example is hierarchical and each claim subset for a downstream consumer is created in the form of a nested EAT. Then the nested EAT is itself encrypted and cryptographically verifiable (due to its COSE envelope) by a downstream consumer (unlike the previous example where a claims set without a COSE envelope is sent to a downstream consumer). Therefore, Transport Layer Security between the receiving and downstream consumers is not strictly required. Nevertheless, downstream consumers of a nested EAT should provide a nonce unique to the EAT they are consuming.

9.5. Detached EAT Bundle Digest Security Considerations

A detached EAT bundle is composed of a nested EAT and an claims set as per Section 5. Although the attached claims set is vulnerable to modification in transit, any modification can be detected by the receiver through the associated digest, which is a claim fully contained within an EAT. Moreover, the digest itself can only be derived using an appropriate COSE hash algorithm, implying that an attacker cannot induce false detection of modified detached claims because the algorithms in the COSE registry are assumed to be of sufficient cryptographic strength.

9.6. Verification Keys

In all cases there must be some way that the verification key is itself verified or determined to be trustworthy. The key identification itself is never enough. This will always be by some out-of-band mechanism that is not described here. For example, the verifier may be configured with a root certificate or a master key by the verifier system administrator.

Often an X.509 certificate or an endorsement carries more than just the verification key. For example, an X.509 certificate might have key usage constraints, and an endorsement might have reference values. When this is the case, the key identifier must be either a protected header or in the payload, such that it is cryptographically bound to the EAT. This is in line with the requirements in section 6 on Key Identification in JSON Web Signature [RFC7515].

10. IANA Considerations

10.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries

Claims defined for EAT are compatible with those of CWT and JWT so the CWT and JWT Claims Registries, [IANA.CWT.Claims] and [IANA.JWT.Claims], are re-used. No new IANA registry is created.

All EAT claims defined in this document are placed in both registries. All new EAT claims defined subsequently should be placed in both registries.

Appendix E describes some considerations when defining new claims.

10.2. CWT and JWT Claims Registered by This Document

This specification adds the following values to the "JSON Web Token Claims" registry established by [RFC7519] and the "CBOR Web Token Claims Registry" established by [RFC8392]. Each entry below is an addition to both registries.

The "Claim Description", "Change Controller" and "Specification Documents" are common and equivalent for the JWT and CWT registries. The "Claim Key" and "Claim Value Types(s)" are for the CWT registry only. The "Claim Name" is as defined for the CWT registry, not the JWT registry. The "JWT Claim Name" is equivalent to the "Claim Name" in the JWT registry.

IANA is requested to register the following claims. The "Claim Value Type(s)" here all name CDDL definitions and are only for the CWT registry.

// RFC editor: please see instructions in followg paragraph and
// remove for final publication

RFC Editor: Please make the following adjustments and remove this paragraph. Replace "*this document*" with this RFC number. In the following, the claims with "Claim Key: TBD" need to be assigned a value in the Specification Required Range, preferably starting around 267. Those below already with a Claim Key number were given early assignment. No change is requested for them except for Claim Key 262. Claim 262 should be renamed from "secboot" to "oemboot" in the JWT registry and its description changed in both the CWT and JWT registries.

* Claim Name: Nonce

* Claim Description: Nonce

* JWT Claim Name: "eat_nonce"

* Claim Key: 10

* Claim Value Type(s): bstr or array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: UEID

* Claim Description: The Universal Entity ID

* JWT Claim Name: "ueid"

* CWT Claim Key: 256

* Claim Value Type(s): bstr

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: SUEIDs

* Claim Description: Semi-permanent UEIDs

* JWT Claim Name: "sueids"

* CWT Claim Key: 257
* Claim Value Type(s): map
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Hardware OEM ID
* Claim Description: Hardware OEM ID
* JWT Claim Name: "oemid"
* Claim Key: 258
* Claim Value Type(s): bstr or int
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Hardware Model
* Claim Description: Model identifier for hardware
* JWT Claim Name: "hwmodel"
* Claim Key: 259
* Claim Value Type(s): bstr
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Hardware Version
* Claim Description: Hardware Version Identifier
* JWT Claim Name: "hwversion"
* Claim Key: TBD 260
* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: OEM Authorized Boot

* Claim Description: Indicates whether the software booted was OEM authorized

* JWT Claim Name: "oemboot"

* Claim Key: 262

* Claim Value Type(s): bool

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Debug Status

* Claim Description: Indicates status of debug facilities

* JWT Claim Name: "dbgstat"

* Claim Key: 263

* Claim Value Type(s): uint

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Location

* Claim Description: The geographic location

* JWT Claim Name: "location"

* Claim Key: 264

* Claim Value Type(s): map

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: EAT Profile
* Claim Description: Indicates the EAT profile followed
* JWT Claim Name: "eat_profile"
* Claim Key: 265
* Claim Value Type(s): uri or oid
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Submodules Section
* Claim Description: The section containing submodules
* JWT Claim Name: "submods"
* Claim Key: 266
* Claim Value Type(s): map
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Uptime
* Claim Description: Uptime
* JWT Claim Name: "uptime"
* Claim Key: TBD
* Claim Value Type(s): uint
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Boot Count

* Claim Description: The number times the entity or submodule has been booted

* JWT Claim Name: "bootcount"

* Claim Key: TBD

* Claim Value Type(s): uint

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Boot Seed

* Claim Description: Identifies a boot cycle

* JWT Claim Name: "bootseed"

* Claim Key: TBD

* Claim Value Type(s): bstr

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: DLOAs

* Claim Description: Certifications received as Digital Letters of Approval

* JWT Claim Name: "dloas"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Name

* Claim Description: The name of the software running in the entity

* JWT Claim Name: "swname"

* Claim Key: TBD

* Claim Value Type(s): tstr

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Version

* Claim Description: The version of software running in the entity

* JWT Claim Name: "swversion"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Manifests

* Claim Description: Manifests describing the software installed on the entity

* JWT Claim Name: "manifests"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Measurements

* Claim Description: Measurements of the software, memory configuration and such on the entity

* JWT Claim Name: "measurements"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Measurement Results

* Claim Description: The results of comparing software measurements to reference values

* JWT Claim Name: "measres"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Intended Use

* Claim Description: Indicates intended use of the EAT

* JWT Claim Name: "intuse"

* Claim Key: TBD

* Claim Value Type(s): uint

* Change Controller: IETF

* Specification Document(s): *this document*

10.3. UEID URN Registered by this Document

IANA is requested to register the following new subtypes in the "DEV URN Subtypes" registry under "Device Identification". See [RFC9039].

Subtype	Description	Reference
ueid	Universal Entity Identifier	This document
sueid	Semi-permanent Universal Entity Identifier	This document

Table 3: UEID URN Registration

ABNF for these two URNs is as follows where b64ueid is the base64url-encoded binary byte-string for the UEID or SUEID:

```
body =/ ueidbody
ueidbody = %sueid: b64ueid
```

10.4. CBOR Tag for Detached EAT Bundle Registered by this Document

In the registry [IANA.cbor-tags], IANA is requested to allocate the following tag from the Specification Required space, with the present document as the specification reference.

Tag	Data Items	Semantics
TBD602	array	Detached EAT Bundle Section 5

Table 4: Detached EAT Bundle Tag Registration

11. References

11.1. Normative References

[DLOA] "Digital Letter of Approval", November 2015, <https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf>.

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

[IANA.COSE.Algorithms] IANA, "CBOR Object Signing and Encryption (COSE)", <https://www.iana.org/assignments/cose>.

[IANA.CWT.Claims]
IANA, "CBOR Web Token (CWT) Claims", <https://www.iana.org/assignments/cwt>.

[IANA.JWT.Claims]
IANA, "JSON Web Token (JWT)", <https://www.iana.org/assignments/jwt>.

[PEN] "Private Enterprise Number (PEN) Request", n.d., <https://pen.iana.org/pen/PenApplication.page>.

[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/info/rfc2119>.

[RFC2252] Wahl, M., Coulbeck, A., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, DOI 10.17487/RFC2252, December 1997, <https://www.rfc-editor.org/info/rfc2252>.

[RFC3986] 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/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[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/info/rfc7519>.

[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/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[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/info/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/info/rfc8610>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[RFC8949] 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/info/rfc8949>.

[RFC9052] 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/info/rfc9052>.

[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/info/rfc9090>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/info/rfc9165>.

[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/info/rfc9334>.

[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/info/rfc9393>.

[ThreeGPP.IMEI] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification", 2019, <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729>.

[WGS84] National Geospatial-Intelligence Agency (NGA), "WORLD GEODETIC SYSTEM 1984, NGA.STND.0036_1.0.0_WGS84", 8 July 2014, <https://earth-info.nga.mil/php/download.php?file=coord-wgs84>.

11.2. Informative References

[BirthdayAttack] "Birthday attack", <https://en.wikipedia.org/wiki/Birthday_attack.>.

[CBOR.Cert.Draft] Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-07, 20 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-07>.

[COSE.X509.Draft] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", Work in Progress, Internet-Draft, draft-ietf-cose-x509-09, 13 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-x509-09>.

[EAT.media-types] Lundblade, L., Birkholz, H., and T. Fossati, "EAT Media Types", Work in Progress, Internet-Draft, draft-ietf-rats-eat-media-type-05, 7 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-media-type-05>.

[IEEE-RA] "IEEE Registration Authority", <https://standards.ieee.org/products-services/regauth/index.html>.

[IEEE.802-2001] "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture", IEEE standard, DOI 10.1109/ieeestd.2014.6847097, July 2014, <https://doi.org/10.1109/ieeestd.2014.6847097>.

[IEEE.802.1AR] "IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity", IEEE standard, DOI 10.1109/ieeestd.2018.8423794, July 2018, <https://doi.org/10.1109/ieeestd.2018.8423794>.

[JTAG] "IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture", February 2010, <https://ieeexplore.ieee.org/document/5412866>.

[OUI.Guide] "Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", August 2017, <https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf>.

[OUI.Lookup] "IEEE Registration Authority Assignments", <https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021, <https://www.rfc-editor.org/info/rfc9039>.

[SUIT.Manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. Rønningstad, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-24, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-24>.

[UCCS] Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-ietf-rats-uccs-07, 27 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-uccs-07>.

[W3C.GeoLoc]
Popescu, A., Ed., "Geolocation API Specification", W3C REC REC-geolocation-API-20131024, W3C REC-geolocation-API-20131024, 24 October 2013, <https://www.w3.org/TR/2013/REC-geolocation-API-20131024/>.

Appendix A. Examples

Most examples are shown as just a Claims-Set that would be a payload for a CWT, JWT, detached EAT bundle or future token types. The signing is left off so the Claims-Set is easier to see. Some examples of signed tokens are also given.

```
// RFC Editor: When the IANA values are permanently assigned, please
// contact the authors so the examples can be regenerated.
// Regeneration is required because IANA-assigned values are inside
// hex and based-64 encoded data and some of these are signed.
```

A.1. Claims Set Examples

A.1.1. Simple TEE Attestation

This is a simple attestation of a TEE that includes a manifest that is a payload CoSWID to describe the TEE's software.

```
/ This is an EAT payload that describes a simple TEE. /

{
    / eat_nonce /       10: h'48df7b172d70b5a18935d0460a73dd71',
    / oemboot /        262: true,
    / dbgstat /        263: 2, / disabled-since-boot /
    / manifests /      273: [
                              [
                               121, / CoAP Content ID. A     /
                                    / made up one until one  /
                                    / is assigned for CoSWID /

                               / This is byte-string wrapped      /
                               / payload CoSWID. It gives the TEE /
                               / software name, the version and   /
                               / the  name of the file it is in.  /
                               / {0: "3a24",                      /
                               /  12: 1,                          /
                               /   1: "Acme TEE OS",              /
                               /  13: "3.1.4",                    /
                               /   2: [{31: "Acme TEE OS", 33: 1}, /
                               /       {31: "Acme TEE OS", 33: 2}], /
                               /   6: {                           /
                               /       17: {                      /
                               /           24: "acme_tee_3.exe"   /
                               /       }                          /
                               /    }                             /
                               /  }                               /
                               h' a60064336132340c01016b
                                  41636d6520544545204f530d65332e31
                                  2e340282a2181f6b41636d6520544545
                                  204f53182101a2181f6b41636d652054
                                  4545204f5318210206a111a118186e61
                                  636d655f7465655f332e657865'
                              ]
                            ]
}
```

```
/ A payload CoSWID created by the SW vendor. All this really does /
/ is name the TEE SW, its version and lists the one file that     /
/ makes up the TEE. /

1398229316({
    / Unique CoSWID ID /    0: "3a24",
    / tag-version /        12: 1,
    / software-name /       1: "Acme TEE OS",
    / software-version /   13: "3.1.4",
    / entity /              2: [
                                   {
        / entity-name /                31: "Acme TEE OS",
        / role        /                33: 1 / tag-creator /
                                   },
                                   {
        / entity-name /                31: "Acme TEE OS",
        / role        /                33: 2 / software-creator /
                                   }
                               ],
    / payload /                6: {
        / ...file /                17: {
            / ...fs-name /             24: "acme_tee_3.exe"
                                   }
                               }
})
```

A.1.2. Submodules for Board and Device

```
/ This example shows use of submodules to give information  /
/ about the chip, board and overall device.                 /
/                                                           /
/ The main attestation is associated with the chip with the /
/ CPU and running the main OS. It is what has the keys and  /
/ produces the token.                                       /
/                                                           /
/ The board is made by a different vendor than the chip.    /
/ Perhaps it is some generic IoT board.                     /
/                                                           /
/ The device is some specific appliance that is made by a   /
/ different vendor than either the chip or the board.       /
/                                                           /
/ Here the board and device submodules aren't the typical   /
/ target environments as described by the RATS architecture /
/ document, but they are a valid use of submodules.         /

{
    / eat_nonce /       10: h'e253cabedc9eec24ac4e25bcbeaf7765'
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: h'894823', / IEEE OUI format OEM ID /
    / hwmodel /        259: h'549dcecc8b987c737b44e40f7c635ce8'
                              / Hash of chip model name /,
    / hwversion /      260: ["1.3.4", 1], / Multipartnumeric  /
    / swname /         271: "Acme OS",
    / swversion /      272: ["3.5.5", 1],
    / oemboot /        262: true,
    / dbgstat /        263: 3, / permanent-disable  /
    / timestamp (iat) /  6: 1526542894,
    / submods / 266: {
        / A submodule to hold some claims about the circuit board /
        "board" :  {
            / oemid /     258: h'9bef8787eba13e2c8f6e7cb4b1f4619a',
            / hwmodel /   259: h'ee80f5a66c1fb9742999a8fdab930893'
                                  / Hash of board module name /,
            / hwversion / 260: ["2.0a", 2] / multipartnumeric+sfx /
        },

        / A submodule to hold claims about the overall device /
        "device" :  {
            / oemid /     258: 61234, / PEN Format OEM ID /
            / hwversion / 260: ["4.0", 1] / Multipartnumeric /
        }
    }
}
```

A.1.3. EAT Produced by Attestation Hardware Block

```
/ This is an example of a token produced by a HW block            /
/ purpose-built for attestation.  Only the nonce claim changes    /
/ from one attestation to the next as the rest  either come       /
/ directly from the hardware or from one-time-programmable memory /
/ (e.g. a fuse). 47 bytes encoded in CBOR (8 byte nonce, 16 byte  /
/ UEID). /

{
    / eat_nonce /       10: h'd79b964ddd5471c1393c8888',
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: 64242, / Private Enterprise Number /
    / oemboot /        262: true,
    / dbgstat /        263: 3, / disabled-permanently /
    / hwversion /      260: [ "3.1", 1 ] / Type is multipartnumeric /
}
```

A.1.4. Key / Key Store Attestation

```
/ This is an attestation of a public key and the key store     /
/ implementation that protects and manages it. The key store   /
/ implementation is in a security-oriented execution           /
/ environment separate from the high-level OS (HLOS), for      /
/ example a Trusted Execution Environment (TEE). The key store /
/ is the Attester.                                             /
/                                                              /
/ There is some attestation of the high-level OS, just version /
/ and boot & debug status. It is a Claims-Set submodule because/
/ it has lower security level than the key store. The key      /
/ store's implementation has access to info about the HLOS, so /
/ it is able to include it.                                    /
/                                                              /
/ A key and an indication of the user authentication given to  /
/ allow access to the key is given. The labels for these are   /
/ in the private space since this is just a hypothetical       /
/ example, not part of a standard protocol.                    /


{
    / eat_nonce /       10: h'99b67438dba40743266f70bf75feb1026d5134
                              97a229bfe8'
    / oemboot /        262: true,
    / dbgstat /        263: 2, / disabled-since-boot /
    / manifests /      273: [
                                [ 121, / CoAP Content ID. A      /
                                       / made up one until one   /
                                       / is assigned for CoSWID  /
                                  h'a60068376262333438376 6
                                    0c000169436172626f6e6974650d6331
```

```
                                        2e320e0102a2181f75496e6475737472
                                        69616c204175746f6d6174696f6e1821
                                        02'
                                    ]
                                    / Above is an encoded CoSWID     /
                                    / with the following data        /
                                    /   SW Name: "Carbonite"         /
                                    /   SW Vers: "1.2"               /
                                    /   SW Creator:                  /
                                    /      "Industrial Automation"   /
                                ],
        / exp /              4: 1634324274, / 2021-10-15T18:57:54Z /
        / iat /              6: 1634317080, / 2021-10-15T16:58:00Z /
                       -80000 : "fingerprint",
                       -80001 : { / The key -- A COSE_Key  /
                    / kty /       1: 2, / EC2, eliptic curve with x & y /
                    / kid /       2: h'36675c206f96236c3f51f54637b94ced',
                    / curve /    -1: 2, / curve is P-256 /
                    / x-coord /  -2: h'65eda5a12577c2bae829437fe338701a
                                      10aaa375e1bb5b5de108de439c08551d',
                    / y-coord /  -3: h'1e52ed75701163f7f9e40ddf9f341b3d
                                      c9ba860af7e0ca7ca7e9eecd0084d19c'
                 },

        / submods /        266 : {
                                "HLOS" : { / submod for high-level OS /
             / eat_nonce /         10: h'8b0b28782a23d3f6',
               / oemboot /        262: true,
               / manifests /      273: [
                                       [ 121, / CoAP Content ID. A      /
                                              / made up one until one   /
                                              / is assigned for CoSWID  /
                                         h'a600687337
                                           6537346b78380c000168
                                           44726f6964204f530d65
                                           52322e44320e0302a218
                                           1F75496E647573747269
                                           616c204175746f6d6174
                                           696f6e182102'
                                         ]
                                         / Above is an encoded CoSWID /
                                         / with the following data:   /
                                         /   SW Name: "Droid OS"      /
                                         /   SW Vers: "R2.D2"         /
                                         /   SW Creator:              /
                                         /     "Industrial Automation"/
                                       ]
                                }
```

```
                        }
   }
```

A.1.5. Software Measurements of an IoT Device

This is a simple token that might be for an IoT device. It includes CoSWID format measurments of the SW. The CoSWID is in byte-string wrapped in the token and also shown in diagnostic form.

```
/ This EAT payload is for an IoT device with a TEE. The attestation /
/ is produced by the TEE. There is a submodule for the IoT OS (the  /
/ main OS of the IoT device that is not as secure as the TEE). The  /
/ submodule contains claims for the IoT OS. The TEE also measures   /
/ the IoT OS and puts the measurements in the submodule.            /

{
    / eat_nonce / 10: h'5e19fba4483c7896'
    / oemboot /  262: true,
    / dbgstat /  263: 2, / disabled-since-boot /
    / oemid /    258: h'8945ad', / IEEE CID based /
    / ueid /     256: h'0198f50a4ff6c05861c8860d13a638ea',
    / submods /  266: {
                        "OS" : {
        / oemboot /         262: true,
        / dbgstat /         263: 2, / disabled-since-boot /
        / measurements /    274: [
                                    [
                                      121, / CoAP Content ID. A     /
                                           / made up one until one  /
                                           / is assigned for CoSWID /

                                     / This is a byte-string wrapped /
                                     / evidence CoSWID. It has       /
                                     / hashes of the main files of   /
                                     / the IoT OS.  /
                                     h'a600663463613234350c
                                       17016d41636d6520522d496f542d4f
                                       530d65332e312e3402a2181f724163
                                       6d65204261736520417474657374 65
                                       7218210103a11183a318187161636d
                                       655f725f696f745f6f732e65786514
                                       1a0044b349078201582005f6b327c1
                                       73b4192bd2c3ec248a292215eab456
                                       611bf7a783e25c1782479905a31818
                                       6d7265736f75726365732e72736314
                                       1a000c38b10782015820c142b9aba4
                                       280c4bb8c75f716a43c99526694caa
```

```
                                        be529571f5569bb7dc542f98a31818
                                        6a636f6d6d6f6e2e6c6962141a0023
                                        3d3b0782015820a6a9dcdfb3884da5
                                        f884e4e1e8e8629958c2dbc7027414
                                        43a913e34de9333be6'
                                      ]
                                    ]
                                  }
                                }
   }

   / An evidence CoSWID created for the "Acme R-IoT-OS" created by /
   / the "Acme Base Attester" (both fictious names).  It provides  /
   / measurements of the SW (other than the attester SW) on the    /
   / device. /

   1398229316({
       / Unique CoSWID ID /    0: "4ca245",
       / tag-version /        12: 23, / Attester-maintained counter /
       / software-name /       1: "Acme R-IoT-OS",
       / software-version /   13: "3.1.4",
       / entity /              2: {
           / entity-name /        31: "Acme Base Attester",
           / role        /        33: 1 / tag-creator /
                                },
       / evidence /            3: {
           / ...file /             17: [
                                       {
               / ...fs-name /              24: "acme_r_iot_os.exe",
               / ...size    /              20: 4502345,
               / ...hash    /               7: [
                                                1, / SHA-256 /
                                                h'05f6b327c173b419
                                                  2bd2c3ec248a2922
                                                  15eab456611bf7a7
                                                  83e25c1782479905'
                                               ]
                                       },
                                       {
               / ...fs-name /              24: "resources.rsc",
               / ...size    /              20: 800945,
               / ...hash    /               7: [
                                                 1, / SHA-256 /
                                                h'c142b9aba4280c4b
                                                  b8c75f716a43c995
                                                  26694caabe529571
                                                  f5569bb7dc542f98'
                                               ]
```

```
                                        },
                                        {
                / ...fs-name /              24: "common.lib",
                / ...size    /              20: 2309435,
                / ...hash    /               7: [
                                                1, / SHA-256 /
                                                h'a6a9dcdfb3884da5
                                                  f884e4e1e8e86299
                                                  58c2dbc702741443
                                                  a913e34de9333be6'
                                             ]
                                        }
                                    ]
                                }
   })
```

A.1.6. Attestation Results in JSON

This is a JSON-encoded payload that might be the output of a verifier that evaluated the IoT Attestation example immediately above.

This particular verifier knows enough about the TEE attester to be able to pass claims like debug status directly through to the relying party. The verifier also knows the reference values for the measured software components and is able to check them. It informs the relying party that they were correct in the "measres" claim. "Trustus Verifications" is the name of the services that verifies the software component measurements.

```
{
   "eat_nonce": "jkd8KL-8=Qlzg4",
   "oemboot": true,
   "dbgstat": "disabled-since-boot",
   "oemid": "iUWt",
   "ueid": "AZj1Ck_2wFhhyIYNE6Y4",
   "swname": "Acme R-IoT-OS",
   "swversion": [
      "3.1.4"
   ],
   "measres": [
      [
         "Trustus Measurements",
         [
            [
               "all",
               "success"
            ]
         ]
      ]
   ]
}
```


A.1.7. JSON-encoded Token with Submodules

This example has its lines wrapped per [RFC8792].

```
{
   "eat_nonce": "lI-IYNE6Rj6O",
   "ueid": "AJj1Ck_2wFhhyIYNE6Y46g==",
   "secboot": true,
   "dbgstat": "disabled-permanently",
   "iat": 1526542894,
   "submods": {
      "Android App Foo": {
         "swname": "Foo.app"
      },
      "Secure Element Eat": [
         "CBOR",
         "2D3ShEOhASagWGaoCkiUj4hg0TpGPhkBAFABmPUKT_bAWGHIhg0TpjjqGQ\
ECGfryGQEFBBkBBvUZAQcDGQEEgmMzLjEBGQEKoWNURUWCL1gg5c-V_ST6txRGdC3VjU\
Pa4XjlX-K5QpGpKRCC_8JjWgtYQPaQywOIZ3-mJKN3X9fLxOhAnsmBa-MvpHRzOw-Ywn\
-67bvJljuctezAPD41s6_At7NbSV3qwJlxIuqGfwe41es="
      ],
      "Linux Android": {
         "swname": "Android"
      },
      "Subsystem J": [
         "JWT",
         "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJKLUF0dGVzd\
GVyIiwiaWF0IjoxNjUxNzc0ODY4LCJleHAiOm51bGwsImF1ZCI6IiIsInN1YiI6IiJ9.\
gjw4nFMhLpJUuPXvMPzK1GMjhyJq2vWXg1416XKszwQ"
      ]
   }
}
```


A.2. Signed Token Examples

A.2.1. Basic CWT Example

This is a simple CWT-format token signed with the ECDSA algorithm.

```
/ This is a full CWT-format token with a very simple payloal. /
/ The main structure visible here is that of the COSE_Sign1.  /

61( 18( [
    h'A10126',                          / protected headers  /
    {},                             / empty unprotected headers /
    h'A20B46024A6B0978DE0A49000102030405060708',    / payload /
    h'9B9B2F5E470000F6A20C8A4157B5763FC45BE759
      9A5334028517768C21AFFB845A56AB557E0C8973
      A07417391243A79C478562D285612E292C622162
      AB233787'                                     / signature /
] ) )
```

A.2.2. CBOR-encoded Detached EAT Bundle

In this detached EAT bundle, the main token is produced by a HW attestation block. The detached Claims-Set is produced by a TEE and is largely identical to the Simple TEE examples above. The TEE digests its Claims-Set and feeds that digest to the HW block.

In a better example the attestation produced by the HW block would be a CWT and thus signed and secured by the HW block. Since the signature covers the digest from the TEE that Claims-Set is also secured.

The detached EAT bundle itself can be assembled by untrusted software.

```
/ This is a detached EAT bundle tag.  Note that 602, the tag /
/ identifying a detached EAT bundle is not yet registered /
/ with IANA /

602([

    / First part is a full EAT token with claims like nonce and /
    / UEID. Most importantly, it includes a submodule that is a /
    / detached digest which is the hash of the "TEE" claims set /
    / in the next section. The COSE payload follows:            /
    / { /
    /      10: h'948F8860D13A463E', /
    /     256: h'0198F50A4FF6C05861C8860D13A638EA', /
    /     258: 64242, /
    /     262: true, /
    /     263: 3, /
    /     260: ["3.1", 1], /
    /     266: { /
    /         "TEE": [ /
    /             -16, /
    /              h'8DEF652F47000710D9F466A4C666E209  /
    /                DD74F927A1CEA352B03143E188838ABE' /
    /         ] /
    /     } /
    /   }  /
    h'D83DD28443A10126A05866A80A48948F8860D13A463E1901
      00500198F50A4FF6C05861C8860D13A638EA19010219FAF2
      19010504190106F519010703190104826333 2E310119010A
      A163544545822F58208DEF652F47000710D9F466A4C666E2
      09DD74F927A1CEA352B03143E188838ABE5840F690CB0388
      677FA624A3775FD7CBC4E8409EC9816BE32FA474733B0F98
      C27FBAEDBBC9963B9CB5ECC03C3E35B3AFC0B7B35B495DEA
      C0997122EA867F07B8D5EB',
    {
       / A CBOR-encoded byte-string wrapped EAT claims-set. It /
       / contains claims suitable for a TEE                    /
       "TEE" : h'a40a48948f8860d13a463e190106f519010702
                 190111818218795858a60064336132340c0101
                 6b41636d6520544545204f530d65332e312e34
                 0282a2181f6b41636d6520544545204f531821
                 01a2181f6b41636d6520544545204f53182102
                 06a111a118186e61636d655f7465655f332e65
                 7865'
    }
 ])
```

```
/ This example contains submodule that is a detached digest,   /
/ which is the hash of a Claims-Set convey outside this token. /
/ Other than that is is the other example of a token from an   /
/ attestation HW block                                         /

{
    / eat_nonce /       10: h'3515744961254b41a6cf9c02',
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: 64242, / Private Enterprise Number /
    / oemboot /        262: true,
    / dbgstat /        263: 3, / disabled-permanently /
    / hwversion /      260: [ "3.1", 1 ], / multipartnumeric /
    / submods/         266: {
                                "TEE": [ / detached digest submod /
                                           -16, / SHA-256 /
                                           h'e5cf95fd24fab7144674
                                             2dd58d43dae178e55fe2
                                             b94291a9291082ffc263
                                             5a0b'
                                         ]
                              }
}
```

A.2.3. JSON-encoded Detached EAT Bundle

In this bundle there are two detached Claims-Sets, "Audio Subsystem" and "Graphics Subsystem". The JWT at the start of the bundle has detached signature submodules with hashes that cover these two Claims-Sets. The JWT itself is protected using HMAC with a key of "xxxxxx".

This example has its lines wrapped per [RFC8792].

```
[
    [
        "JWT",
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlYXRfbm9uY2UiOiJ5dT\
c2Tk44SXVWNmUiLCJzdWJtb2RzIjp7IkF1ZGlvIFN1YnN5c3RlbSI6WyJESUdFU1QiLF\
siU0hBLTI1NiIsIkZSRW4yVlR3aTk5cWNNRVFzYmxtTVFnM2I1b2ZYUG5OM1BJYW5CME\
5RT3MiXV0sIkdyYXBoaWNzIFN1YnN5c3RlbSI6WyJESUdFU1QiLFsiU0hBLTI1NiIsIk\
52M3NqUVU3Q1Z0RFRka0RTUlhWcFZDNUNMVFBCWmVQWWhTLUhoVlZWMXMiXV19fQ.FYs\
7R-TKhgAk85NyCOPQlbtGGjFM_3chnhBEOuM6qCo"
    ],
    {
        "Audio Subsystem" : "ewogICAgImVhdF9ub25jZSI6ICJsSStJWU5FNlJ\
qNk8iLAogICAgInVlaWQiOiAiQWROSlU0b1lYdFVwQStIeDNqQTcvRFEiCiAgICAib2V\
taWQiOiAiaVVXdCIsCiAgICAib2VtYm9vdCI6IHRydWUsIAogICAgInN3bmFtZSI6ICJ\
BdWRpbyBQcm9jZXNzb3IgT1MiCn0K",
        "Graphics Subsystem" : "ewogICAgImVhdF9ub25jZSI6ICJZWStJWU5F\
NlJqNk8iLAogICAgInVlaWQiOiAiQWVUTUlRQ1NVMnhWQmtVdGlndHU3bGUiCiAgICAi\
b2VtaWQiOiA3NTAwMCwKICAgICJvZW1ib290IjogdHJ1ZSwgCiAgICAic3duYW1lIjog\
IkdyYXBoaWNzIE9TIgp9Cg"
    }
]
```


Appendix B. UEID Design Rationale

B.1. Collision Probability

This calculation is to determine the probability of a collision of type 0x01 UEIDs given the total possible entity population and the number of entities in a particular entity management database.

Three different sized databases are considered. The number of devices per person roughly models non-personal devices such as traffic lights, devices in stores they shop in, facilities they work in and so on, even considering individual light bulbs. A device may have individually attested subsystems, for example parts of a car or a mobile phone. It is assumed that the largest database will have at most 10% of the world’s population of devices. Note that databases that handle more than a trillion records exist today.

The trillion-record database size models an easy-to-imagine reality over the next decades. The quadrillion-record database is roughly at the limit of what is imaginable and should probably be accommodated. The 100 quadrillion database is highly speculative perhaps involving nanorobots for every person, livestock animal and domesticated bird. It is included to round out the analysis.

Note that the items counted here certainly do not have IP address and are not individually connected to the network. They may be connected to internal buses, via serial links, Bluetooth and so on. This is not the same problem as sizing IP addresses.

People	Devices / Person	Subsystems / Device	Database Portion	Database Size
10 billion	100	10	10%	trillion (10^12)
10 billion	100,000	10	10%	quadrillion (10^15)
100 billion	1,000,000	10	10%	100 quadrillion (10^17)

Table 5: Entity Database Size Examples

This is conceptually similar to the Birthday Problem where m is the number of possible birthdays, always 365, and k is the number of people. It is also conceptually similar to the Birthday Attack where collisions of the output of hash functions are considered.

The proper formula for the collision calculation is

$$p = 1 - e^{-k^2/(2n)}$$

p Collision Probability
n Total possible population
k Actual population

However, for the very large values involved here, this formula requires floating point precision higher than commonly available in calculators and software so this simple approximation is used. See [BirthdayAttack].

$$p = k^2 / 2n$$

For this calculation:

p Collision Probability
n Total population based on number of bits in UEID
k Population in a database

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	2 * 10^-15	8 * 10^-35	5 * 10^-55
quadrillion (10^15)	2 * 10^-09	8 * 10^-29	5 * 10^-49
100 quadrillion (10^17)	2 * 10^-05	8 * 10^-25	5 * 10^-45

Table 6: UEID Size Options

Next, to calculate the probability of a collision occurring in one year's operation of a database, it is assumed that the database size is in a steady state and that 10% of the database changes per year. For example, a trillion record database would have 100 billion states per year. Each of those states has the above calculated probability of a collision.

This assumption is a worst-case since it assumes that each state of the database is completely independent from the previous state. In reality this is unlikely as state changes will be the addition or deletion of a few records.

The following tables gives the time interval until there is a probability of a collision based on there being one tenth the number of states per year as the number of records in the database.

```
t = 1 / ((k / 10) * p)

t  Time until a collision
p  Collision probability for UEID size
k  Database size
```

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	60,000 years	10^24 years	10^44 years
quadrillion (10^15)	8 seconds	10^14 years	10^34 years
100 quadrillion (10^17)	8 microseconds	10^11 years	10^31 years

Table 7: UEID Collision Probability

Clearly, 128 bits is enough for the near future thus the requirement that type 0x01 UEIDs be a minimum of 128 bits.

There is no requirement for 256 bits today as quadrillion-record databases are not expected in the near future and because this time-to-collision calculation is a very worst case. A future update of the standard may increase the requirement to 256 bits, so there is a requirement that implementations be able to receive 256-bit UEIDs.

B.2. No Use of UUID

A UEID is not a Universally Unique Identifier (UUID) [RFC4122] by conscious choice for the following reasons.

UUIDs are limited to 128 bits which may not be enough for some future use cases.

Today, cryptographic-quality random numbers are available from common CPUs and hardware. This hardware was introduced between 2010 and 2015. Operating systems and cryptographic libraries give access to this hardware. Consequently, there is little need for implementations to construct such random values from multiple sources on their own.

Version 4 UUIDs do allow for use of such cryptographic-quality random numbers, but do so by mapping into the overall UUID structure of time and clock values. This structure is of no value here yet adds complexity. It also slightly reduces the number of actual bits with entropy.

The design of UUID accommodates the construction of a unique identifier by combination of several identifiers that separately do not provide sufficient uniqueness. UEID takes the view that this construction is no longer needed, in particular because cryptographic-quality random number generators are readily available. It takes the view that hardware, software and/or manufacturing process implement UEID in a simple and direct way.

Note also that that a type 2 UEID (EUI/MAC) is only 7 bytes compared to 16 for a UUID.

Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)

This section describes several distinct ways in which an IEEE IDevID [IEEE.802.1AR] relates to EAT, particularly to UEID and SUEID.

[IEEE.802.1AR] orients around the definition of an implementation called a "DevID Module." It describes how IDevIDs and LDevIDs are stored, protected and accessed using a DevID Module. A particular level of defense against attack that should be achieved to be a DevID is defined. The intent is that IDevIDs and LDevIDs can be used with any network protocol or message format. In these protocols and message formats the DevID secret is used to sign a nonce or similar to prove the association of the DevID certificates with the device.

By contrast, EAT standardizes a message format that is sent to a relying party, the very thing that is not defined in [IEEE.802.1AR]. Nor does EAT give details on how keys, data and such are stored protected and accessed. EAT is intended to work with a variety of different on-device implementations ranging from minimal protection of assets to the highest levels of asset protection. It does not define any particular level of defense against attack, instead providing a set of security considerations.

EAT and DevID can be viewed as complimentary when used together or as competing to provide a device identity service.

C.1. DevID Used With EAT

As just described, EAT standardizes a message format and [IEEE.802.1AR] doesn’t. Vice versa, EAT doesn’t define a an device implementation, but DevID does.

Hence, EAT can be the message format that a DevID is used with. The DevID secret becomes the attestation key used to sign EATs. The DevID and its certificate chain become the endorsement sent to the verifier.

In this case, the EAT and the DevID are likely to both provide a device identifier (e.g. a serial number). In the EAT it is the UEID (or SUEID). In the DevID (used as an endorsement), it is a device serial number included in the subject field of the DevID certificate. It is probably a good idea in this use for them to be the same serial number or for the UEID to be a hash of the DevID serial number.

C.2. How EAT Provides an Equivalent Secure Device Identity

The UEID, SUEID and other claims like OEM ID are equivalent to the secure device identity put into the subject field of a DevID certificate. These EAT claims can represent all the same fields and values that can be put in a DevID certificate subject. EAT explicitly and carefully defines a variety of useful claims.

EAT secures the conveyance of these claims by having them signed on the device by the attestation key when the EAT is generated. EAT also signs the nonce that gives freshness at this time. Since these claims are signed for every EAT generated, they can include things that vary over time like GPS location.

DevID secures the device identity fields by having them signed by the manufacturer of the device sign them into a certificate. That certificate is created once during the manufacturing of the device and never changes so the fields cannot change.

So in one case the signing of the identity happens on the device and the other in a manufacturing facility, but in both cases the signing of the nonce that proves the binding to the actual device happens on the device.

While EAT does not specify how the signing keys, signature process and storage of the identity values should be secured against attack, an EAT implementation may have equal defenses against attack. One reason EAT uses CBOR is because it is simple enough that a basic EAT implementation can be constructed entirely in hardware. This allows EAT to be implemented with the strongest defenses possible.

C.3. An X.509 Format EAT

It is possible to define a way to encode EAT claims in an X.509 certificate. For example, the EAT claims might be mapped to X.509 v3 extensions. It is even possible to stuff a whole CBOR-encoded unsigned EAT token into a X.509 certificate.

If that X.509 certificate is an IDevID or LDevID, this becomes another way to use EAT and DevID together.

Note that the DevID must still be used with an authentication protocol that has a nonce or equivalent. The EAT here is not being used as the protocol to interact with the rely party.

C.4. Device Identifier Permanence

In terms of permanence, an IDevID is similar to a UEID in that they do not change over the life of the device. They cease to exist only when the device is destroyed.

An SUEID is similar to an LDevID. They change on device life-cycle events.

[IEEE.802.1AR] describes much of this permanence as resistant to attacks that seek to change the ID. IDevID permanence can be described this way because [IEEE.802.1AR] is oriented around the definition of an implementation with a particular level of defense against attack.

EAT is not defined around a particular implementation and must work on a range of devices that have a range of defenses against attack. EAT thus can’t be defined permanence in terms of defense against attack. EAT’s definition of permanence is in terms of operations and device lifecycle.

Appendix D. CDDL for CWT and JWT

[RFC8392] was published before CDDL was available and thus is specified in prose, not CDDL. Following is CDDL specifying CWT as it is needed to complete this specification. This CDDL also covers the Claims-Set for JWT.

Note that Section 4.3.1 requires that the iat claim be the type ~time-int (Section 7.2.1), not the type ~time when it is used in an EAT as floating-point values are not allowed for the "iat" claim in EAT.

The COSE-related types in this CDDL are defined in [RFC9052].

This however is NOT a normative or standard definition of CWT or JWT in CDDL. The prose in CWT and JWT remain the normative definition.

```
; This is replicated from draft-ietf-rats-uccs

Claims-Set = {
    * $$Claims-Set-Claims
    * Claim-Label .feature "extended-claims-label" => any
}
Claim-Label = int / text
string-or-uri = text

$$Claims-Set-Claims //= ( iss-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( sub-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( aud-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( exp-claim-label => ~time )
$$Claims-Set-Claims //= ( nbf-claim-label => ~time )
$$Claims-Set-Claims //= ( iat-claim-label => ~time )
$$Claims-Set-Claims //= ( cti-claim-label => bytes )

iss-claim-label = JC<"iss", 1>
sub-claim-label = JC<"sub", 2>
aud-claim-label = JC<"aud", 3>
exp-claim-label = JC<"exp", 4>
nbf-claim-label = JC<"nbf", 5>
iat-claim-label = JC<"iat", 6>
cti-claim-label = CBOR-ONLY<7>  ; jti in JWT: different name and text

JSON-ONLY<J> = J .feature "json"
CBOR-ONLY<C> = C .feature "cbor"

JC<J,C> = JSON-ONLY<J> / CBOR-ONLY<C>

; A JWT message is either a JWS or JWE in compact serialization form
; with the payload a Claims-Set. Compact serialization is the
; protected headers, payload and signature, each b64url encoded and
; separated by a ".". This CDDL simply matches top-level syntax of of
; a JWS or JWE since it is not possible to do more in CDDL.

JWT-Message =
   text .regexp "[A-Za-z0-9_-]+\\.[A-Za-z0-9_-]+\\.[A-Za-z0-9_-]+"


; Note that the payload of a JWT is defined in claims-set.cddl. That
; definition is common to CBOR and JSON.
```

```
; This is some CDDL describing a CWT at the top level This is
; not normative. RFC 8392 is the normative definition of CWT.

CWT-Messages = CWT-Tagged-Message / CWT-Untagged-Message

; The payload of the COSE_Message is always a Claims-Set

; The contents of a CWT Tag must always be a COSE tag
CWT-Tagged-Message = #6.61(COSE_Tagged_Message)

; An untagged CWT may be a COSE tag or not
CWT-Untagged-Message = COSE_Messages
```

Appendix E. New Claim Design Considerations

The following are design considerations that may be helpful to take into account when creating new EAT claims. It is the product of discussion in the working group.

EAT reuses the CWT and JWT claims registries. There is no registriy exclusively for EAT claims. This is not an update to the expert review criteria for the JWT and CWT claims registries as that would be an overreach for this document.

E.1. Interoperability and Relying Party Orientation

It is a broad goal that EATs can be processed by relying parties in a general way regardless of the type, manufacturer or technology of the device from which they originate. It is a goal that there be general-purpose verification implementations that can verify tokens for large numbers of use cases with special cases and configurations for different device types. This is a goal of interoperability of the semantics of claims themselves, not just of the signing, encoding and serialization formats.

This is a lofty goal and difficult to achieve broadly requiring careful definition of claims in a technology neutral way. Sometimes it will be difficult to design a claim that can represent the semantics of data from very different device types. However, the goal remains even when difficult.

E.2. Operating System and Technology Neutral

Claims should be defined such that they are not specific to an operating system. They should be applicable to multiple large high-level operating systems from different vendors. They should also be applicable to multiple small embedded operating systems from multiple vendors and everything in between.

Claims should not be defined such that they are specific to a software environment or programming language.

Claims should not be defined such that they are specific to a chip or particular hardware. For example, they should not just be the contents of some HW status register as it is unlikely that the same HW status register with the same bits exists on a chip of a different manufacturer.

The boot and debug state claims in this document are an example of a claim that has been defined in this neutral way.

E.3. Security Level Neutral

Many use cases will have EATs generated by some of the most secure hardware and software that exists. Secure Elements and smart cards are examples of this. However, EAT is intended for use in low-security use cases the same as high-security use case. For example, an app on a mobile device may generate EATs on its own.

Claims should be defined and registered on the basis of whether they are useful and interoperable, not based on security level. In particular, there should be no exclusion of claims because they are just used only in low-security environments.

E.4. Reuse of Extant Data Formats

Where possible, claims should use already standardized data items, identifiers and formats. This takes advantage of the expertise put into creating those formats and improves interoperability.

Often extant claims will not be defined in an encoding or serialization format used by EAT. It is preferred to define a CBOR and JSON encoding for them so that EAT implementations do not require a plethora of encoders and decoders for serialization formats.

In some cases, it may be better to use the encoding and serialization as is. For example, signed X.509 certificates and CRLs can be carried as-is in a byte string. This retains interoperability with the extensive infrastructure for creating and processing X.509 certificates and CRLs.

E.5. Proprietary Claims

It is not always possible or convenient to achieve the above goals, so the definition and use of proprietary claims is an option.

For example, a device manufacturer may generate a token with proprietary claims intended only for verification by a service offered by that device manufacturer. This is a supported use case.

In many cases proprietary claims will be the easiest and most obvious way to proceed, however for better interoperability, use of general standardized claims is preferred.

Appendix F. Endorsements and Verification Keys

The verifier must possess the correct key when it performs the cryptographic part of an EAT verification (e.g., verifying the COSE/JOSE signature). This section describes several ways to identify the verification key. There is not one standard method.

The verification key itself may be a public key, a symmetric key or something complicated in the case of a scheme like Direct Anonymous Attestation (DAA).

RATS Architecture [RFC9334] describes what is called an endorsement. This is an input to the verifier that is usually the basis of the trust placed in an EAT and the attester that generated it. It may contain the public key for verification of the signature on the EAT. It may contain implied claims, those that are passed on to the relying party in attestation results.

There is not yet any standard format(s) for an endorsement. One format that may be used for an endorsement is an X.509 certificate. Endorsement data like reference values and implied claims can be carried in X.509 v3 extensions. In this use, the public key in the X.509 certificate becomes the verification key, so identification of the endorsement is also identification of the verification key.

The verification key identification and establishment of trust in the EAT and the attester may also be by some other means than an endorsement.

For the components (attester, verifier, relying party,...) of a particular end-end attestation system to reliably interoperate, its definition should specify how the verification key is identified. Usually, this will be in the profile document for a particular attestation system.

See also security consideration in Section 9.6.

F.1. Identification Methods

Following is a list of possible methods of key identification. A specific attestation system may employ any one of these or one not listed here.

The following assumes endorsements are X.509 certificates or equivalent and thus does not mention or define any identifier for endorsements in other formats. If such an endorsement format is created, new identifiers for them will also need to be created.

F.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC9052] and [RFC7515]

COSE leaves the semantics of the key ID open-ended. It could be a record locator in a database, a hash of a public key, an input to a Key Derivation Function (KDF), an Authority Key Identifier (AKI) for an X.509 certificate or other. The profile document should specify what the key ID's semantics are.

F.1.2. JWS and COSE X.509 Header Parameters

COSE X.509 [COSE.X509.Draft] and JSON Web Signature [RFC7515] define several header parameters (x5t, x5u,...) for referencing or carrying X.509 certificates any of which may be used.

The X.509 certificate may be an endorsement and thus carrying additional input to the verifier. It may be just an X.509 certificate, not an endorsement. The same header parameters are used in both cases. It is up to the attestation system design and the verifier to determine which.

F.1.3. CBOR Certificate COSE Header Parameters

Compressed X.509 and CBOR Native certificates are defined by CBOR Certificates [CBOR.Cert.Draft]. These are semantically compatible with X.509 and therefore can be used as an equivalent to X.509 as described above.

These are identified by their own header parameters (c5t, c5u,...).

F.1.4. Claim-Based Key Identification

For some attestation systems, a claim may be re-used as a key identifier. For example, the UEID uniquely identifies the entity and therefore can work well as a key identifier or endorsement identifier.

This has the advantage that key identification requires no additional bytes in the EAT and makes the EAT smaller.

This has the disadvantage that the unverified EAT must be substantially decoded to obtain the identifier since the identifier is in the COSE/JOSE payload, not in the headers.

Appendix G. Changes from Previous Drafts

// RFC editor: please remove this paragraph.

The following is a list of known changes since the immediately previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences. A comprehensive history is available via the IETF Datatracker's record for this document.

G.1. From draft-ietf-rats-eat-24

* Use only CDDL definition names for "Claim Value Type" column in CWT claim registry

* Correct the "Claim Value Type" for some claims

* Make SUIT reference informative (it use is optional in an optional claim)

Contributors

Many thanks to the following contributors to draft versions of this document:

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@sit.fraunhofer.de

Thomas Fossati
Arm Limited
Email: thomas.fossati@arm.com

Miguel Ballesteros

Michael Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca

Patrick Uiterwijk

Mathias Brossard

Hannes Tschofenig
Arm Limited
Email: hannes.tschofenig@arm.com

Paul Crowley

Authors' Addresses

Laurence Lundblade
Security Theory LLC
Email: lgl@securitytheory.com

Giridhar Mandyam
Email: giridhar.mandyam@gmail.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough
GU14 7LS
United Kingdom
Phone: +44 1252 363189
Email: jodonogh@qti.qualcomm.com

Carl Wallace
Red Hound Software, Inc.
Email: carl@redhoundsoftware.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 13 March 2024

H. Birkholz
Fraunhofer SIT
E. Voit
Cisco
W. Pan
Huawei

10 September 2023

Attestation Event Stream Subscription
draft-ietf-rats-network-device-subscription-04

Abstract

This memo defines how to subscribe to YANG Event Streams for Remote Attestation Procedures (RATS). In RATS, Conceptional Messages, are defined. Analogously, the YANG module defined in this memo augments the YANG module for TPM-based Challenge-Response based Remote Attestation (CHARRA) to allow for subscription to remote attestation Evidence. Additionally, this memo provides the methods and means to define additional Event Streams for other Conceptual Message as illustrated in the RATS Architecture, e.g. Attestation Results, Endorsements, or Event Logs.

Status of This Memo

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

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

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

This Internet-Draft will expire on 13 March 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 3
2. Terminology 5
  2.1. Requirements Notation 5
3. Operational Model 5
  3.1. Sequence Diagram 5
  3.2. Continuously Verifying Freshness 7
    3.2.1. TPM 1.2 Quote 8
    3.2.2. TPM 2 Quote 8
4. Remote Attestation Event Stream 9
  4.1. Subscription to the <attestation> Event Stream 9
  4.2. Replaying a history of previous TPM extend operations . . 10
    4.2.1. TPM2 Heartbeat 11
  4.3. YANG notifications placed on the <attestation> Event Stream 11
    4.3.1. pcr-extend 11
    4.3.2. tpm12-attestation 13
    4.3.3. tpm20-attestation 13
  4.4. Filtering Evidence at the Attester 14
  4.5. Replaying previous PCR Extend events 14
  4.6. Configuring the <attestation> Event Stream 14
5. YANG Module 15
6. Event Streams for Conceptual Messages 22
7. Security Considerations 22
8. IANA Considerations 22
9. References 22
  9.1. Normative References 22
  9.2. Informative References 23
Appendix A. Change Log 24
Acknowledgements 24
Authors' Addresses 24

1. Introduction

[I-D.ietf-rats-tpm-based-network-device-attest] and [I-D.ietf-rats-yang-tpm-charra] define the operational prerequisites and a YANG Model for the acquisition of Evidence and other Conceptional Messages from a TPM-based network device. However, there are limitations inherent in the challenge-response based remote attestation (CHARRA [I-D.ietf-rats-reference-interaction-models]) upon which these documents are based. One of these limitation is that it is a RATS role's duty to request Conceptional Messages, such as Evidence as provided by [I-D.ietf-rats-yang-tpm-charra], from another RATS entity. The result is that the interval between the occurrence of a security-relevant change event, and the event's visibility within the interested RATS entity, such as a Verifier or a Relying Party, can be unacceptably long. It is common to convey Conceptual Messages ad-hoc or periodically via requests. As new technologies emerge, some of these solutions require Conceptual Messages to be conveyed from one RATS entity to another without the need of continuous polling. Subscription to YANG Notifications [RFC8639] provides a set of standardized tools to facilitate these emerging requirements. This memo specifies a YANG augment to subscribe to YANG modeled remote attestation Evidence as defined in [I-D.ietf-rats-yang-tpm-charra]. Additionally, this memo provides the means to define further Event Streams to convey Conceptional Messages other than Evidence, such as Attestation Results, Endorsements, or Event Logs.

In essence, the limitation of poll-based interactions results in two adverse effects:

1. Conceptual Messages are not streamed to an interested consumer of information, e.g., Verifiers or Relying Parties, as soon as they are generated.

2. If they were to be streamed, Conceptual Messages are not appraisable for their freshness in every scenario. This becomes more important with Conceptional Messages that have a strong dependency on freshness, such as Evidence and corresponding Attestation Results.

This specification addresses the first adverse effect by enabling a consumer of Conceptual Messages (the subscriber) to request a continuous stream of new or updated Conceptual Messages via an [RFC8639] subscription to an <attestation> Event Stream. This new Event Stream is defined in this document and exists upon the producer of Conceptual Messages (the publisher). In the case of a Verifier's subscription to an Attester's Evidence, the Attester will continuously stream a requested set of freshly generated Evidence to the subscribing Verifier.

The second adverse effect results from the use of nonces in the challenge-response interaction model [I-D.ietf-rats-reference-interaction-models] realized in [I-D.ietf-rats-yang-tpm-charra]. In [I-D.ietf-rats-yang-tpm-charra], an Attester must wait for a new nonce from a Verifier before it generates a new TPM Quote. To address delays resulting from such a wait, this specification enables freshness to be asserted asynchronously via the streaming attestation interaction model [I-D.ietf-rats-reference-interaction-models]. To convey a RATS Conceptual Message, an initial nonce is provided during the subscription to an Event Stream.

There are several options to refresh a nonce provided by the initial subscription or its freshness characteristics. All of these methods are out-of-band of an established subscription to YANG Notifications. Two complementary methods are taken into account by this memo:

1. a central provider supplies new fresh nonces, e.g. via a Handle Provider that distributes Epoch IDs to all entities in a domain as described in [I-D.ietf-rats-architecture] and as facilitated by the Uni-Directional Remote Attestation described in [I-D.ietf-rats-reference-interaction-models] or

2. the freshness characteristics of a received nonce are updated by -- potentially periodic or ad-hoc -- out-of-band TPM Quote requests as facilitated by [I-D.ietf-rats-yang-tpm-charra].

Both approaches to update the freshness characteristics of the Conceptual Messages conveyed via subscription to YANG Notification that are taken into account by this memo assume that clock drift between involved entities can occur. In consequence, in some usage scenarios the timing considerations for freshness [I-D.ietf-rats-architecture] might have to be updated in some regular interval. Analogously, there are can be additional methods that are not describe by but nevertheless supported by this memo.

This memo enables to remove the two adverse effects described by using the YANG augment specified. The YANG augment supports, for example, a RATS Verifier to maintain a continuous appraisal procedure of verifiably fresh Attester Evidence without relying on continuous polling.

2. Terminology

The following terms are imported from [I-D.ietf-rats-architecture]: Attester, Conceptual Message, Evidence, Relying Party, and Verifier. Also imported are the time definitions time(VG), time(NS), time(EG), time(RG), and time(RA) from that document's Appendix A. The following terms are imported from [RFC8639]: Event Stream, Subscription, Event Stream Filter, Dynamic Subscription.

2.1. Requirements Notation

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.

3. Operational Model

[I-D.ietf-rats-tpm-based-network-device-attest] describes the conveyance of TPM-based Evidence from a Verifier to an Attester using the CHARRA interaction model [I-D.ietf-rats-reference-interaction-models]. The operational model and corresponding sequence diagram described in this section is based on [I-D.ietf-rats-yang-tpm-charra]. The basis for interoperability required for additional types of Event Streams is covered in Section 6. The following sub-section focuses on subscription to YANG Notifications to the <attestation> Event Stream.

3.1. Sequence Diagram

Figure 1 below is a sequence diagram which updates Figure 5 of [I-D.ietf-rats-tpm-based-network-device-attest]. This sequence diagram replaces the [I-D.ietf-rats-tpm-based-network-device-attest] TPM-specific challenge-response interaction model with a [RFC8639] Dynamic Subscription to an <attestation> Event Stream. The contents of the <attestation> Event Stream are defined below within Section 4.

```
   .----------.                          .--------------------------.
   | Attester |                          | Relying Party / Verifier |
   '----------'                          '--------------------------'
      time(VG)                                                      |
   generateClaims(targetEnvironment)                                |
         | => claims, eventLogs                                     |
         |                                                          |
         |<---------establish-subscription(<attestation>)------time(NS)
         |                                                          |
      time(EG)                                                      |
   generateEvidence(nonce, PcrSelection, collectedClaims)           |
         | => SignedPcrEvidence(nonce, PcrSelection)                |
         | => LogEvidence(collectedClaims)                          |
         |                                                          |
         |--filter(<pcr-extend>)---------------------------------->|
         |--<tpm12-attestation> or <tpm20-attestation>------------>|
         |                                                          |
         |                                                   time(RG,RA)
         |     appraiseEvidence(SignedPcrEvidence, eventLog, refClaims)
         |                                     attestationResult <= |
         |                                                          |
         ~                                                          ~
      time(VG')                                                     |
   generateClaimes(targetEnvironment)                               |
         | => claims                                                |
         |                                                          |
      time(EG')                                                     |
   generateEvidence(handle, PcrSelection, collectedClaims)          |
         | => SignedPcrEvidence(nonce, PcrSelection)                |
         | => LogEvidence(collectedClaims)                          |
         |                                                          |
         |--filter(<pcr-extend>)---------------------------------->|
         |--<tpm12-attestation> or <tpm20-attestation>------------>|
         |                                                          |
         |                                                  time(RG',RA')
         |    appraiseEvidence(SignedPcrEvidence, eventLog, refClaims)
         |                                     attestationResult <= |
         |                                                          |
```

Figure 1: YANG Subscription Model for Remote Attestation

* time(VG,RG,RA) are identical to the corresponding time definitions from [I-D.ietf-rats-tpm-based-network-device-attest].

* time(VG',RG',RA') are subsequent instances of the corresponding times from Figure 5 in [I-D.ietf-rats-tpm-based-network-device-attest].

* time(NS) - the subscriber generates a nonce and makes an [RFC8639] <establish-subscription> request based on a nonce. This request also includes the augmentations defined in this document's YANG model. Key subscription RPC parameters include:

  - the nonce,

  - a set of PCRs of interest which the wants to appraise, and

  - an optional filter which can reduce the logged events on the <attestation> stream pushed to the Verifier.

* time(EG) - an initial response of Evidence is returned to the Verifier. This includes:

  - a replay of filtered log entries which have extended into a PCR of interest since boot are sent in the <pcr-extend> notification, and

  - a signed TPM quote that contains at least the PCRs from the <establish-subscription> RPC are included in a <tpm12-attestation> or <tpm20-attestation>). This quote must have included the nonce provided at time(NS).

* time(VG',EG') - this occurs when a PCR is extended subsequent to time(EG). Immediately after the extension, the following information needs to be pushed to the Verifier:

  - any values extended into a PCR of interest,

  - a signed TPM Quote showing the result the PCR extension, and

  - and a handle (see Section 6. in [I-D.ietf-rats-reference-interaction-models], which is either the initially received nonce or a more recently received Epoch ID (see Section 10.3. in [I-D.ietf-rats-architecture] that contains a new nonce or equivalent qualified data.

One way to acquire a new time synchronisation that allows for the reuse of the initially received nonce as a fresh handle is elaborated on in the follow section Section 3.2.

3.2. Continuously Verifying Freshness

As there is no new Verifier nonce provided at time(EG'), it is important to validate the freshness of TPM Quotes which are delivered at that time. The method of doing this verification will vary based on the capabilities of the TPM cryptoprocessor used.

3.2.1. TPM 1.2 Quote

The [RFC8639] notification format includes the <eventTime> object. This can be used to determine the amount of time subsequent to the initial subscription each notification was sent. However this time is not part of the signed results which are returned from the Quote, and therefore is not trustworthy as objects returned in the Quote. Therefore a Verifier MUST periodically issue a new nonce, and receive this nonce within a TPM quote response in order to ensure the freshness of the results. This can be done using the <tpm12-challenge-response-attestation> RPC from [I-D.ietf-rats-yang-tpm-charra].

3.2.2. TPM 2 Quote

When the Attester includes a TPM2 compliant cryptoprocessor, internal time-related counters are included within the signed TPM Quote. By including a initial nonce in the [RFC8639] subscription request, fresh values for these counters are pushed as part of the first TPM Quote returned to the Verifier. And then as shown by [I-D.birkholz-rats-tuda], subsequent TPM Quotes delivered to the Verifier can the be appraised for freshness based on the predictable incrementing of these time-related countersr.

The relevant internal time-related counters defined within [TPM2.0] can be seen within <tpms-clock-info>. These counters include the <clock>, <reset-counter>, and <restart-counter> objects. The rules for appraising these objects are as follows:

* If the <clock> has incremented for no more than the same duration as both the <eventTime> and the Verifier's internal time since the initial time(EG) and any previous time(EG'), then the TPM Quote may be considered fresh. Note that [TPM2.0] allows for +/- 15% clock drift. However many chips significantly improve on this maximum drift. If available, chip specific maximum drifts SHOULD be considered during the appraisal process.

* If the <reset-counter>, <restart-counter> has incremented. The existing subscription MUST be terminated, and a new <establish-subscription> SHOULD be generated.

* If a TPM Quote on any subscribed PCR has not been pushed to the Verifier for a duration of an Attester defined heartbeat interval, then a new TPM Quote notification should be sent to the Verifier. This may often be the case, as certain PCRs might be infrequently updated.

```
.----------.                        .--------------------------.
| Attester |                        | Relying Party / Verifier |
'----------'                        '--------------------------'
   time(VG',EG')                                               |
     |-<tpm20-attestation>------------------------------>|
     |                                   :                |
     ~                            Heartbeat interval      ~
     |                                   :                |
   time(EG')                             :                |
     |-<tpm20-attestation>------------------------------>|
     |                                                    |
```

4. Remote Attestation Event Stream

The <attestation> Event Stream is an [RFC8639] compliant Event Stream which is defined within this section and within the YANG Module of [I-D.ietf-rats-yang-tpm-charra]. This Event Stream contains YANG notifications which carry Evidence to assists a Verifier in appraising the Trustworthiness Level of an Attester. Data Nodes within Section 4.6 allow the configuration of this Event Stream's contents on an Attester.

This <attestation> Event Stream may only be exposed on Attesters supporting [I-D.ietf-rats-tpm-based-network-device-attest]. As with [I-D.ietf-rats-tpm-based-network-device-attest], it is up to the Verifier to understand which types of cryptoprocessors and keys are acceptable.

4.1. Subscription to the <attestation> Event Stream

To establish a subscription to an Attester in a way which provides provably fresh Evidence, initial randomness must be provided to the Attester. This is done via the augmentation of a <nonce-value> into [RFC8639] the <establish-subscription> RPC. Additionally, a Verifier must ask for PCRs of interest from a platform.

```
augment /sn:establish-subscription/sn:input:
  +---w nonce-value    binary
  +---w pcr-index*     tpm:pcr
```

The result of the subscription will be that passing of the following information:

1. <tpm12-attestation> and <tpm20-attestation> notifications which include the provided <nonce-value>. These attestation notifications MUST at least include all the <pcr-indicies> requested in the RPC.

2. a series of <pcr-extend> notifications which reference the requested PCRs on all TPM based cryptoprocessors on the Attester.

3. <tpm12-attestation> and <tpm20-attestation> notifications generated within a few seconds of the <pcr-extend> notifications. These attestation notifications MUST at least include any PCRs extended.

If the Verifier does not want to see the logged extend operations for all PCRs available from an Attester, an Event Stream Filter should be applied. This filter will remove Evidence from any PCRs which are not interesting to the Verifier.

4.2. Replaying a history of previous TPM extend operations

Unless it is relying on Known Good Values, a Verifier will need to acquire a history of PCR extensions since the Attester has been booted. This history may be requested from the Attester as part of the <establish-subscription> RPC. This request is accomplished by placing a very old <replay-start-time> within the original RPC request. As the very old <replay-start-time> will pre-date the time of Attester boot, a <replay-start-time-revision> will be returned in the <establish-subscription> RPC response, indicating when the Attester booted. Immediately following the response (and before the notifications above) one or more <pcr-extend> notifications which document all extend operations which have occurred for the requested PCRs since boot will be sent. Many extend operations to a single PCR index on a single TPM SHOULD be included within a single notification.

Note that if a Verifier has a partial history of extensions, the <replay-start-time> can be adjusted so that known extensions are not forwarded.

The end of this history replay will be indicated with the [RFC8639] <replay-completed> notification. For more on this sequence, see Section 2.4.2.1 of [RFC8639].

After the <replay-complete> notification is provided, a TPM Quote will be requested and the result passed to the Verifier via a <tpm12-attestation> and <tpm20-attestation> notification. If there have been any additional extend operations which have changed a subscribed PCR value in this quote, these MUST be pushed to the Verifier before the <tpm12-attestation> and <tpm20-attestation> notification.

At this point the Verifier has sufficient Evidence appraise the reported extend operations for each PCR, as well compare the expected value of the PCR value against that signed by the TPM.

4.2.1. TPM2 Heartbeat

For TPM2, make sure that every requested PCR is sent within an <tpm20-attestation> no less frequently than once per heartbeat interval. This MAY be done with a single <tpm20-attestation> notification that includes all requested PCRs every heartbeat interval. This MAY be done with several <tpm20-attestation> notifications at different times during that heartbeat interval.

4.3. YANG notifications placed on the <attestation> Event Stream

4.3.1. pcr-extend

This notification documents when a subscribed PCR is extended within a single TPM cryptoprocessor. It SHOULD be emmitted no less than the <marshalling-period> after an the PCR is first extended. (The reason for the marshalling is that it is quite possible that multiple extensions to the same PCR have been made in quick succession, and these should be reflected in the same notification.) This notification MUST be emmitted prior to a <tpm12-attestation> or <tpm20-attestation> notification which has included and signed the results of any specific PCR extension. If pcr extending events occur during the generation of the <tpm12-attestation> or <tpm20-attestation> notification, the marshalling period MUST be extended so that a new <pcr-extend> is not sent until the corresponding notifications have been sent.

```
+---n pcr-extend
   +--ro certificate-name     certificate-name-ref
   +--ro pcr-index-changed*   tpm:pcr
   +--ro attested-event* []
      +--ro attested-event
         +--ro extended-with             binary
         +--ro (event-details)?
            +--:(bios-event-log)
            |  +--ro bios-event-entry* [event-number]
            |     +--ro event-number    uint32
            |     +--ro event-type?     uint32
            |     +--ro pcr-index?      pcr
            |     +--ro digest-list* []
            |     |  +--ro hash-algo?   identityref
            |     |  +--ro digest*      binary
            |     +--ro event-size?     uint32
            |     +--ro event-data*     uint8
            +--:(ima-event-log)
            |  +--ro ima-event-entry* [event-number]
            |     +--ro event-number               uint64
            |     +--ro ima-template?              string
            |     +--ro filename-hint?             string
            |     +--ro filedata-hash?             binary
            |     +--ro filedata-hash-algorithm?   string
            |     +--ro template-hash-algorithm?   string
            |     +--ro template-hash?             binary
            |     +--ro pcr-index?                 pcr
            |     +--ro signature?                 binary
            +--:(netequip-boot-event-log)
               +--ro boot-event-entry* [event-number]
                  +--ro event-number               uint64
                  +--ro filename-hint?             string
                  +--ro filedata-hash?             binary
                  +--ro filedata-hash-algorithm?   string
                  +--ro file-version?              string
                  +--ro file-type?                 string
                  +--ro pcr-index?                 pcr
```

Each <pcr-extend> MUST include one or more values being extended into the PCR. These are passed within the <extended-with> object. For each extension, details of the event SHOULD be provided within the <event-details> object. The format of any included <event-details> is identified by the <event-type>. This document includes two YANG structures which may be inserted into the <event-details>. These two structures are: <ima-event-log> and <bios-event-log>. Implementations wanting to provide additional documentation of a type of PCR extension may choose to define additional YANG structures which can be placed into <event-details>.

4.3.2. tpm12-attestation

This notification contains an instance of a TPM1.2 style signed cryptoprocessor measurement. It is supplemented by Attester information which is not signed. This notification is generated and emitted from an Attester when at least one PCR identified within the subscribed <pcr-indices> has changed from the previous <tpm12-attestation> notification. This notification MUST NOT include the results of any PCR extensions not previously reported by a <pcr-extend>. This notification SHOULD be emitted as soon as a TPM Quote can extract the latest PCR hashed values. This notification MUST be emitted prior to a subsequent <pcr-extend>.

```
+---n tpm12-attestation {taa:TPM12}?
   +--ro certificate-name       tpm:certificate-name-ref
   +--ro up-time?               uint32
   +--ro TPM_QUOTE2?            binary
   +--ro TPM12-hash-algo?       identityref
   +--ro unsigned-pcr-values* []
      +--ro pcr-index*   tpm:pcr
      +--ro pcr-value*   binary
```

All YANG objects above are defined within [I-D.ietf-rats-yang-tpm-charra]. The <tpm12-attestation> is not replayable.

4.3.3. tpm20-attestation

This notification contains an instance of TPM2 style signed cryptoprocessor measurements. It is supplemented by Attester information which is not signed. This notification is generated at two points in time:

* every time at least one PCR has changed from a previous tpm20-attestation. In this case, the notification SHOULD be emitted within 10 seconds of the corresponding <pcr-extend> being sent:

* after a locally configurable minimum heartbeat period since a previous tpm20-attestation was sent.

```
+---n tpm20-attestation {taa:TPM20}?
   +--ro certificate-name       tpm:certificate-name-ref
   +--ro TPMS_QUOTE_INFO        binary
   +--ro quote-signature?       binary
   +--ro up-time?               uint32
   +--ro unsigned-pcr-values* []
      +--ro TPM20-hash-algo?   identityref
      +--ro pcr-values* [pcr-index]
         +--ro pcr-index    pcr
         +--ro pcr-value?   binary
```

All YANG objects above are defined within [I-D.ietf-rats-yang-tpm-charra]. The <tpm20-attestation> is not replayable.

4.4. Filtering Evidence at the Attester

It can be useful _not_ to receive all Evidence related to a PCR. An example of this is would be a when a Verifier maintains known good values of a PCR. In this case, it is not necessary to send each extend operation.

To accomplish this reduction, when an RFC8639 <establish-subscription> RPC is sent, a <stream-filter> as per RFC8639, Section 2.2 can be set to discard a <pcr-extend> notification when the <pcr-index-changed> is uninteresting to the verifier.

4.5. Replaying previous PCR Extend events

To verify the value of a PCR, a Verifier must either know that the value is a known good value [KGV] or be able to reconstruct the hash value by viewing all the PCR-Extends since the Attester rebooted. Wherever a hash reconstruction might be needed, the <attestation> Event Stream MUST support the RFC8639 <replay> feature. Through the <replay> feature, it is possible for a Verifier to retrieve and sequentially hash all of the PCR extending events since an Attester booted. And thus, the Verifier has access to all the evidence needed to verify a PCR’s current value.

4.6. Configuring the <attestation> Event Stream

Figure 2 is tree diagram which exposes the operator configurable elements of the <attestation> Event Stream. This allows an Attester to select what information should be available on the stream. A fetch operation also allows an external device such as a Verifier to understand the current configuration of stream.

Almost all YANG objects below are defined via reference from [I-D.ietf-rats-yang-tpm-charra]. There is one object which is new with this model however. <tpm2-heartbeat> defines the maximum amount of time which should pass before a subscriber to the Event Stream should get a <tpm20-attestation> notification from devices which contain a TPM2.

```
  augment /tpm:rats-support-structures:
    +--rw marshalling-period?                  uint8
    +--rw tpm12-subscribed-signature-scheme?
    |   -> ../tpm:attester-supported-algos/tpm12-asymmetric-signing
    |      {taa:tpm12}?
    +--rw tpm20-subscribed-signature-scheme?
    |   -> ../tpm:attester-supported-algos/tpm20-asymmetric-signing
    |      {taa:tpm20}?
    +--rw tpm20-subscription-heartbeat?        uint16
  augment /tpm:rats-support-structures/tpm:tpms:
    +--rw subscription-aik?        tpm:certificate-name-ref
    +--rw (subscribable)?
       +--:(tpm12-stream) {taa:tpm12}?
       |  +--rw TPM12-hash-algo?   identityref
       |  +--rw tpm12-pcr-index*   tpm:pcr
       +--:(tpm20-stream) {taa:tpm20}?
          +--rw TPM20-hash-algo?   identityref
          +--rw tpm20-pcr-index*   tpm:pcr
```

Figure 2: Configuring the \<attestation\> Event Stream

5. YANG Module

This YANG module imports modules from [I-D.ietf-rats-yang-tpm-charra] and [RFC8639]. It is also work-in-progress.

```
<CODE BEGINS> ietf-rats-attestation-stream@2023-09-10.yang
module ietf-tpm-remote-attestation-stream {
  yang-version 1.1;
  namespace
     "urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation-stream";
  prefix tras;

  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-tpm-remote-attestation {
    prefix tpm;
    reference
```

```
         "draft-ietf-rats-yang-tpm-charra";
     }
     import ietf-tcg-algs {
       prefix taa;
     }

     organization "IETF";
     contact
       "WG Web:   <http://tools.ietf.org/wg/rats/>
        WG List:  <mailto:rats@ietf.org>

        Editor:   Eric Voit
                  <mailto:evoit@cisco.com>";

     description
       "This module contains conceptual YANG specifications for
        subscribing to attestation streams being generated from TPM chips.

        Copyright (c) 2023 IETF Trust and the persons identified
        as authors of the code. All rights reserved.

        Redistribution and use in source and binary forms, with
        or without modification, is permitted pursuant to, and
        subject to the license terms contained in, the Simplified
        BSD License set forth in Section 4.c of the IETF Trust's
        Legal Provisions Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
        itself for full legal notices.";

     revision 2023-09-10 {
       description
         "Initial version.";
       reference
         "draft-birkholz-rats-network-device-subscription";
     }

     /*
      * IDENTITIES
      */

     identity pcr-unsubscribable {
       base sn:establish-subscription-error;
       description
         "Requested PCR is subscribable by the Attester.";
     }
```

```
/*
 * Groupings
 */

grouping heartbeat {
  description
    "Allows an Attester to push verifiable, current TPM PCR values
    even when there have been no recent changes to PCRs.";
  leaf tpm20-subscription-heartbeat {
    type uint16;
    description
      "Number of seconds before the Attestation stream should send a
      new notification with a fresh quote.  This allows confirmation
      that the PCR values haven't changed since the last
      tpm20-attestation.";
  }
}


/*
 * RPCs
 */

augment "/sn:establish-subscription/sn:input" {
  when 'derived-from-or-self(sn:stream, "attestation")';
  description
    "This augmentation adds a nonce to as a subscription parameters
     that apply specifically to datastore updates to RPC input.";
  uses tpm:nonce;
  leaf-list pcr-index {
    type tpm:pcr;
    min-elements 1;
    description
      "The numbers/indexes of the PCRs. This will act as a filter for
      the subscription so that 'tpm-extend' notifications related to
      non-requested PCRs will not be sent to a subscriber.";
  }
}

/*
 * NOTIFICATIONS
 */

notification pcr-extend {
  description
    "This notification indicates that one or more PCRs have been
    extended within a TPM based cryptoprocessor.  In less than the
    'marshalling-period', it MUST be followed with either a
```

```
        corresponding tpm12-attestation or tpm20-attestation notification
        which exposes the result of the PCRs updated.";
      uses tpm:certificate-name-ref;
      leaf-list pcr-index-changed {
        type tpm:pcr;
        min-elements 1;
        description
          "The number of each PCR extended.  This list MUST contain the
          set of PCRs descibed within the event log details.  This leaf
          can be derived from the list of attested events, but exposing
          it here allows for easy filtering of the notifications of
          interest to a verifier.";
      }
      list attested-event {
        description
          "A set of events which extended an Attester PCR.  The sequence
          of elements represented in list must match the sequence of
          events placed into the TPM's PCR.";
        container attested-event {
          description
            "An instance of an event which extended an Attester PCR";
          leaf extended-with {
            type binary;
            mandatory true;
            description
              "Information extending the PCR.";
          }
          choice event-details {
            description
              "Contains the event happened the Attester thought
              was worthy of recording in a PCR.

              choices are of types defined by the identityref
              base tpm:attested_event_log_type";
            case bios-event-log {
              if-feature "tpm:bios";
              description
                "BIOS/UEFI event log format";
              uses tpm:bios-event-log;
            }
            case ima-event-log {
              if-feature "tpm:ima";
              description
                "IMA event log format";
              uses tpm:ima-event-log;
            }
            case netequip-boot-event-log {
              if-feature "tpm:netequip_boot";
```

```
              description
                "IMA event log format";
              uses tpm:network-equipment-boot-event-log;
            }
          }
        }
      }
    }

    notification tpm12-attestation {
      if-feature "taa:tpm12";
      description
        "Contains an instance of TPM1.2 style signed cryptoprocessor
        measurements.  It is supplemented by unsigned Attester
        information.";
      leaf certificate-name {
        type tpm:certificate-name-ref;
        mandatory true;
        description
          "Allows a TPM quote to be associated with a certificate.";
      }
      uses tpm:tpm12-attestation;
      uses tpm:tpm12-hash-algo;
      list unsigned-pcr-values {
        description
          "Allows notifications to be filtered by PCR number or
          PCR value based on via YANG related mechanisms such as PATH.
          This is done without requiring the filtering structure to be
          applied against TCG structured data.";
        leaf-list pcr-index {
          type tpm:pcr;
          min-elements 1;
          description
            "PCR index number.";
        }
        leaf-list pcr-value {
          type binary;
          description
            "PCR value in a sequence which matches to the 'pcr-index'.";
        }
      }
    }

    notification tpm20-attestation {
      if-feature "taa:tpm20";
      description
        "Contains an instance of TPM2 style signed cryptoprocessor
        measurements.  It is supplemented by unsigned Attester
```

```
        information.";
      leaf certificate-name {
        type tpm:certificate-name-ref;
        mandatory true;
        description
          "Allows a TPM quote to be associated with a certificate.";
      }
      uses tpm:tpm20-attestation {
        description
          "Provides the attestation info.  Also ensures PCRs can be XPATH
          filtered by refining the unsigned data so that it appears.";
        refine unsigned-pcr-values {
          min-elements 1;
        }
        refine unsigned-pcr-values/pcr-values {
          min-elements 1;
        }
      }
    }


    /*
     * DATA NODES
     */

    augment "/tpm:rats-support-structures" {
      description
        "Defines platform wide 'attestation' stream subscription
        parameters.";
      leaf marshalling-period {
        type uint8;
        default 5;
        description
          "The maximum number of seconds between the time an event
          extends a PCR, and the 'tpm-extend' notification which reports
          it to a subscribed Verifier.  This period allows multiple
          extend operations bundled together and handled as a group.";
      }
      leaf tpm12-subscribed-signature-scheme {
        if-feature "taa:tpm12";
        type leafref {
          path "../tpm:attester-supported-algos" +
                 "/tpm:tpm12-asymmetric-signing";
        }
        description
          "A single signature-scheme which will be used to sign the
          evidence from a TPM 1.2. which is then placed onto the
          'attestation' event stream.";
```

```
      }
      leaf tpm20-subscribed-signature-scheme {
        if-feature "taa:tpm20";
        type leafref {
          path "../tpm:attester-supported-algos" +
                 "/tpm:tpm20-asymmetric-signing";
        }
        description
          "A single signature-scheme which will be used to sign the
          evidence from a TPM 2.0. which is then placed onto the
          'attestation' event stream.";
      }
      uses heartbeat{
        if-feature "taa:tpm20";
      }
    }

    augment "/tpm:rats-support-structures/tpm:tpms" {
      description
        "Allows the configuration 'attestation' stream parameters for a
        TPM.";
      leaf subscription-aik {
        type tpm:certificate-name-ref;
        description
          "Identifies the certificate-name associated with the
          notifications in the 'attestation' stream.";
      }
      choice subscribable {
        config true;
        description
          "Indicates that the set of notifications which comprise the
          'attestation' event stream can be modified or tuned by a
          network administrator.";
        case tpm12-stream {
          if-feature "taa:tpm12";
          description
            "Configuration elements for a TPM1.2 event stream.";
          uses tpm:tpm12-hash-algo;
          leaf-list tpm12-pcr-index {
            type tpm:pcr;
            description
              "The numbers/indexes of the PCRs which can be subscribed.";
          }
        }
        case tpm20-stream {
          if-feature "taa:tpm20";
          description
            "Configuration elements for a TPM2.0 event stream.";
```

```
          uses tpm:tpm20-hash-algo;
          leaf-list tpm20-pcr-index {
            type tpm:pcr;
            description
              "The numbers/indexes of the PCRs which can be subscribed.";
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Event Streams for Conceptual Messages

Analogous to the [RFC8639] compliant <attestation> Event Stream for the conveyance of remote attestation Evidence as defined in Section Section 4, additional Event Streams can be defined for this YANG augment. Additional Event Streams require separate YANG augment specifications that provide the Event Stream definition and optionally a content format definition either via subscriptions to YANG datastores or dedicated YANG Notifications. It is possible to use either YANG subscription methods to other YANG modules for RATS Conceptual Messages or to define Event Streams for other none-YANG-modeled data. In the context of RATS Conceptual Messages, both options MUST be a specified via YANG augments to this specification.

7. Security Considerations

To be written.

8. IANA Considerations

To be written.

9. References

9.1. Normative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-22, 28 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-22>.

[I-D.ietf-rats-reference-interaction-models]
Birkholz, H., Eckel, M., Pan, W., and E. Voit, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-ietf-rats-reference-interaction-models-08, 10 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-reference-interaction-models-08>.

[I-D.ietf-rats-tpm-based-network-device-attest]
Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", Work in Progress, Internet-Draft, draft-ietf-rats-tpm-based-network-device-attest-14, 22 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-tpm-based-network-device-attest-14>.

[I-D.ietf-rats-yang-tpm-charra]
Birkholz, H., Eckel, M., Bhandari, S., Voit, E., Sulzen, B., Xia, L., Laffey, T., and G. Fedorkow, "A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs", Work in Progress, Internet-Draft, draft-ietf-rats-yang-tpm-charra-21, 18 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-yang-tpm-charra-21>.

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

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

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://doi.org/10.17487/RFC8639>.

[TPM2.0] TCG, "TPM 2.0 Library Specification", n.d., <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

9.2. Informative References

[I-D.birkholz-rats-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in

Progress, Internet-Draft, draft-birkholz-rats-tuda-07, 10 July 2022, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-tuda-07>.

[KGV] TCG, "KGV", October 2003, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-NetEq-Attestation-Workflow-Outline_v1r9b_pubrev.pdf>.

Appendix A. Change Log

v01-v02

* Match YANG changes/simplifications made to charra

v00-v01

* rename notification: pcr-extended, which supports multiple PCRs

* netequip boot added

* YANG structure extension removed

* Matched to structural changes made within charra

Acknowledgements

Thanks to ...

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Eric Voit
Cisco Systems, Inc.
Email: evoit@cisco.com

Wei Pan
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu
210012
China
Email: william.panwei@huawei.com

RATS
Internet-Draft
Intended status: Standards Track
Expires: 27 November 2022

L. Lundblade
Security Theory LLC
H. Birkholz
Fraunhofer SIT
T. Fossati
arm

26 May 2022

EAT Media Types
draft-lundblade-rats-eat-media-type-00

Abstract

Payloads used in Remote Attestation Procedures may require an associated media type for their conveyance, for example when used in RESTful APIs.

This memo defines media types to be used for Entity Attestation Tokens (EAT).

Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://github.com/thomas-fossati/draft-eat-mt.

Status of This Memo

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

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

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

This Internet-Draft will expire on 27 November 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
2. EAT Types 3
3. A Media Type Parameter for EAT Profiles 4
4. Examples 5
5. Security Considerations 5
6. IANA Considerations 5
6.1. Media Types 6
6.2. application/eat-cwt Registration 6
6.3. application/eat-jwt Registration 7
6.4. application/eat-deb+cbor Registration 7
6.5. application/eat-deb+json Registration 8
6.6. application/eat-ucs+cbor Registration 8
6.7. application/eat-ucs+json Registration 9
6.8. Content-Format 9
7. References 10
7.1. Normative References 10
7.2. Informative References 11
Acknowledgments 11
Authors' Addresses 11

1. Introduction

Payloads used in Remote Attestation Procedures [RATS-Arch] may require an associated media type for their conveyance, for example when used in RESTful APIs (Figure 1).

```
    .----.                    .----------.                .----------.
    | RP |                    | Attester |                | Verifier |
    '-+--'                    '----+-----'                '-----+----'
      |                            | POST /verify                |
      |                            | EAT(Evidence)               |
      |                            +---------------------------->|
      |                            |                     200 OK  |
      |                            |   EAT(Attestation Results)  |
      |                            |<----------------------------+
      |                 POST /auth |                             |
      |   EAT(Attestation Results) |                             |
      |<---------------------------+                             |
      | 201 Created                |                             |
      +---------------------------->|                             |
      |                            |                             |
      |                            |                             |
```

Figure 1: Conveying RATS conceptual messages in REST APIs using EAT

This memo defines media types to be used for Entity Attestation Token (EAT) [EAT] payloads independently of the RATS Conceptual Message in which they manifest themselves.

1.1. Requirements Language

This document uses the terms and concepts defined in [RATS-Arch].

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.

2. EAT Types

Figure 2 illustrates the six EAT wire formats and how they relate to each other. [EAT] defines four of them (CWT, JWT and DEB in its JSON and CBOR flavours), whilst [UCCS] defines the remaining two: UCCS and UJCS.

```
                  .-----.
             .----+ UJCS |<-------------------------.
            |     '-----'                            |
            |                                        |
            |      .-----.                           |
            +-----+ UCCS |<-----------------------.  |
            |     '-----'                          | |
            |                                      | |
            |      .------.                        | |
            +-----+  JWT  |<------.                | |
            |     '------'      .--+---.           | |
            |                  | Crypto |<------.  | |
            |      .------.     '--+---'          | | |
            +-----+  CWT  |<------'               | | |
            |     '------'                     .---+-+-+----.
            |                                  | Claims-Set +--.
            |      .------.                    '---+---+----'   |
            +-----+ DEB-J |<------.              | ^ |        v
            |     '------'      .--+---.         | | |     .------.
            |                  | Bundle |<------'  | |    | Digest |
            |      .------.     '--+---'           | v     '--+---'
            +-----+ DEB-C |<------'  ^         .---+----.     |
            |     '------'           |         | submod |<---'
            |                        |         '--------'
            v                        |             ^
     .--------------.                |             |
     | Nested-Token +----------------+------------'
     '--------------'

                 .-------.     .---------.   .------.
       Legenda:  | Process |   | Wire Fmt |   | CDDL |
                 '-------'     '---------'   '------'
```

Figure 2: EAT Types

3. A Media Type Parameter for EAT Profiles

EAT is an open and flexible format. To improve interoperability, Section 7 of [EAT] defines the concept of EAT profiles. Profiles are used to constrain the parameters that producers and consumers of a specific EAT profile need to understand in order to interoperate. For example: the number and type of claims, which serialisation format, the supported signature schemes, etc. EATs carry an in-band profile identifier using the eat_profile claim (see Section 4.3.3 of [EAT]). The value of the eat_profile claim is either an OID or a URI.

The media types defined in this document include an optional profile parameter that can be used to mirror the eat_profile claim of the transported EAT. Exposing the EAT profile at the API layer allows API routers to dispatch payloads directly to the profile-specific processor without having to snoop into the request bodies. This design also provides a finer-grained and scalable type system that matches the inherent extensibility of EAT. The expectation being that a certain EAT profile automatically obtains a media type derived from the base (e.g., application/eat-cwt) by populating the profile parameter with the corresponding OID or URL.

4. Examples

The example in Figure 3 illustrates the usage of EAT media types for transporting attestation evidence.

```
POST /challenge-response/v1/session/1234567890 HTTP/1.1
Host: verifier.example
Accept: application/eat-cwt; profile=tag:ar4si.example,2021
Content-Type: application/eat-cwt; profile=tag:evidence.example,2022

[ CBOR-encoded EAT w/ profile=tag:evidence.example,2022 ]
```

Figure 3: Example REST Verification API (request)

The example in Figure 4 illustrates the usage of EAT media types for transporting attestation results.

```
HTTP/1.1 200 OK
Content-Type: application/eat-cwt; profile=tag:ar4si.example,2021

[ CBOR-encoded EAT w/ profile=tag:ar4si.example,2021 ]
```

Figure 4: Example REST Verification API (response)

In both cases the profile is carried as an explicit parameter.

5. Security Considerations

The security consideration of [EAT] and [UCCS] apply in full.

6. IANA Considerations

// RFC Editor: please replace RFCthis with this RFC number and remove
// this note.

6.1. Media Types

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

Name	Template	Reference
EAT CWT	application/eat-cwt	RFCthis, Section 6.2
EAT JWT	application/eat-jwt	RFCthis, Section 6.3
EAT CBOR DEB	application/eat-deb+cbor	RFCthis, Section 6.4
EAT JSON DEB	application/eat-deb+json	RFCthis, Section 6.5
EAT UCCS	application/eat-ucs+cbor	RFCthis, Section 6.6
EAT UJCS	application/eat-ucs+json	RFCthis, Section 6.7

Table 1: New Media Types

6.2. application/eat-cwt Registration

Type name: application
Subtype name: eat-cwt
Required parameters: n/a
Optional parameters: "profile" (EAT profile in string format. OIDs MUST use the dotted-decimal notation. The parameter value is case-insensitive.)
Encoding considerations: binary
Security considerations: Section 5 of RFCthis
Interoperability considerations: n/a
Published specification: Section 6.1 of RFCthis
Applications that use this media type: Attesters, Verifiers, Endorsers and Reference-Value providers, Relying Parties that need to transfer EAT payloads over HTTP(S), CoAP(S), and other transports.
Fragment identifier considerations: n/a
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration:
// maybe

6.3. application/eat-jwt Registration

```
Type name:  application
Subtype name:  eat-jwt
Required parameters:  n/a
Optional parameters:  "profile" (EAT profile in string format.  OIDs
   MUST use the dotted-decimal notation.  The parameter value is
   case-insensitive.)
Encoding considerations:  8bit
Security considerations:  Section 5 of RFCthis
Interoperability considerations:  n/a
Published specification:  Section 6.1 of RFCthis
Applications that use this media type  Attesters, Verifiers,
   Endorsers and Reference-Value providers, Relying Parties that need
   to transfer EAT payloads over HTTP(S), CoAP(S), and other
   transports.
Fragment identifier considerations:  n/a
Person & email address to contact for further information:  RATS WG
   mailing list (rats@ietf.org)
Intended usage:  COMMON
Restrictions on usage:  none
Author/Change controller:  IETF
Provisional registration:
   // maybe
```

6.4. application/eat-deb+cbor Registration

```
Type name:  application
Subtype name:  eat-deb+cbor
Required parameters:  n/a
Optional parameters:  "profile" (EAT profile in string format.  OIDs
   MUST use the dotted-decimal notation.  The parameter value is
   case-insensitive.)
Encoding considerations:  binary
Security considerations:  Section 5 of RFCthis
Interoperability considerations:  n/a
Published specification:  Section 6.1 of RFCthis
Applications that use this media type:  Attesters, Verifiers,
   Endorsers and Reference-Value providers, Relying Parties that need
   to transfer EAT payloads over HTTP(S), CoAP(S), and other
   transports.
Fragment identifier considerations:  n/a
Person & email address to contact for further information:  RATS WG
   mailing list (rats@ietf.org)
Intended usage:  COMMON
Restrictions on usage:  none
Author/Change controller:  IETF
Provisional registration:
```

```
      // maybe
```

6.5. application/eat-deb+json Registration

```
   Type name:  application
   Subtype name:  eat-deb+json
   Required parameters:  n/a
   Optional parameters:  "profile" (EAT profile in string format.  OIDs
      MUST use the dotted-decimal notation.  The parameter value is
      case-insensitive.)
   Encoding considerations:  Same as [RFC7159]
   Security considerations:  Section 5 of RFCthis
   Interoperability considerations:  n/a
   Published specification:  Section 6.1 of RFCthis
   Applications that use this media type  Attesters, Verifiers,
      Endorsers and Reference-Value providers, Relying Parties that need
      to transfer EAT payloads over HTTP(S), CoAP(S), and other
      transports.
   Fragment identifier considerations:  n/a
   Person & email address to contact for further information:  RATS WG
      mailing list (rats@ietf.org)
   Intended usage:  COMMON
   Restrictions on usage:  none
   Author/Change controller:  IETF
   Provisional registration:
      // maybe
```

6.6. application/eat-ucs+cbor Registration

```
   Type name:  application
   Subtype name:  eat-ucs+cbor
   Required parameters:  n/a
   Optional parameters:  "profile" (EAT profile in string format.  OIDs
      MUST use the dotted-decimal notation.  The parameter value is
      case-insensitive.)
   Encoding considerations:  binary
   Security considerations:  Section 5 of RFCthis
   Interoperability considerations:  n/a
   Published specification:  Section 6.1 of RFCthis
   Applications that use this media type:  Attesters, Verifiers,
      Endorsers and Reference-Value providers, Relying Parties that need
      to transfer EAT payloads over HTTP(S), CoAP(S), and other
      transports.
   Fragment identifier considerations:  n/a
   Person & email address to contact for further information:  RATS WG
      mailing list (rats@ietf.org)
   Intended usage:  COMMON
   Restrictions on usage:  none
```

Author/Change controller: IETF
Provisional registration:
// maybe

6.7. application/eat-ucs+json Registration

Type name: application
Subtype name: eat-ucs+json
Required parameters: n/a
Optional parameters: "profile" (EAT profile in string format. OIDs MUST use the dotted-decimal notation. The parameter value is case-insensitive.)
Encoding considerations: Same as [RFC7159]
Security considerations: Section 5 of RFCthis
Interoperability considerations: n/a
Published specification: Section 6.1 of RFCthis
Applications that use this media type Attesters, Verifiers, Endorsers and Reference-Value providers, Relying Parties that need to transfer EAT payloads over HTTP(S), CoAP(S), and other transports.
Fragment identifier considerations: n/a
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration:
// maybe

6.8. Content-Format

> (*Issue*: need a way to pass the profile information when using content formats. A new CoAP option?)

IANA is requested to register a Content-Format number in the "CoAP Content-Formats" sub-registry, within the "Constrained RESTful Environments (CoRE) Parameters" Registry [IANA.core-parameters], as follows:

Content-Type	Content Coding	ID	Reference
application/eat-cwt	-	TBD1	RFCthis
application/eat-jwt	-	TBD2	RFCthis
application/eat-deb+cbor	-	TBD3	RFCthis
application/eat-deb+json	-	TBD4	RFCthis
application/eat-ucs+cbor	-	TBD5	RFCthis
application/eat-ucs+json	-	TBD6	RFCthis

Table 2: New Content-Formats

TBD1..6 are to be assigned from the space 256..999.

In the registry as defined by Section 12.3 of [CoAP] at the time of writing, the column "Content-Type" is called "Media type" and the column "Content Coding" is called "Encoding".
// RFC editor: please remove this paragraph.

7. References

7.1. Normative References

[CoAP] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

[EAT] Lundblade, L., Mandyam, G., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-13, 20 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-13>.

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

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/rfc/rfc7159>.

[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>.

[UCCS] Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-ietf-rats-uccs-02, 12 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-uccs-02>.

7.2. Informative References

[RATS-Arch] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-16, 24 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-16>.

Acknowledgments

TODO

Authors' Addresses

Laurence Lundblade
Security Theory LLC
Email: lgl@securitytheory.com

Henk Birkholz
Fraunhofer Institute for Secure Information Technology
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Thomas Fossati
arm
Email: thomas.fossati@arm.com

Remote ATtestation ProcedureS
Internet-Draft
Intended status: Standards Track
Expires: 13 April 2023

C. Wallace
Red Hound
R. Housley
Vigil Security
T. Fossati
Y. Deshpande
arm

10 October 2022

Concise TA Stores (CoTS)
draft-wallace-rats-concise-ta-stores-01

Abstract

Trust anchor (TA) stores may be used for several purposes in the Remote Attestation Procedures (RATS) architecture including verifying endorsements, reference values, digital letters of approval, attestations, or public key certificates. This document describes a Concise Reference Integrity Manifest (CoRIM) extension that may be used to convey optionally constrained trust anchor stores containing optionally constrained trust anchors in support of these purposes.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-wallace-rats-concise-ta-stores/.

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 13 April 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 3
  1.1. Constraints 4
2. Conventions and Definitions 4
3. Trust anchor management for RATS 5
  3.1. TA and CA conveyance 5
    3.1.1. The concise-ta-stores Container 6
    3.1.2. The concise-ta-store-map Container 6
    3.1.3. The cas-and-tas-map Container 8
  3.2. Environment definition 9
    3.2.1. The environment-group-list Array 9
    3.2.2. The abbreviated-swid-tag-map Container 10
    3.2.3. The named-ta-store Type 11
  3.3. Constraints definition 11
    3.3.1. The $$tas-list-purpose Type 11
    3.3.2. Claims 12
  3.4. Processing a concise-ta-stores RIM 12
  3.5. Verifying a concise-ta-stores RIM 12
4. CDDL definitions 12
5. Examples 14
6. Security Considerations 21
7. IANA Considerations 21
  7.1. CoRIM CBOR Tag Registration 21
8. References 21
  8.1. Normative References 21
  8.2. Informative References 22

Acknowledgments 23
Appendix A 23
Authors' Addresses 25

1. Introduction

The RATS architecture [I-D.draft-ietf-rats-architecture] uses the definition of a trust anchor from [RFC6024]: "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative." In the context of RATS, a trust anchor may be a public key or a symmetric key. This document focuses on trust anchors that are represented as public keys.

The Concise Reference Integrity Manifest (CoRIM) [I-D.draft-birkholz-rats-corim] specification defines a binary encoding for reference values using the Concise Binary Object Representation (CBOR) [RFC8949]. Amongst other information, a CoRIM may include key material for use in verifying evidence from an attesting environment (see section 3.11 in [I-D.draft-birkholz-rats-corim]). The extension in this document aims to enable public key material to be decoupled from reference data for several reasons, described below.

Trust anchor (TA) and certification authority (CA) public keys may be less dynamic than the reference data that comprises much of a reference integrity manifest (RIM). For example, TA and CA lifetimes are typically fairly long while software versions change frequently. Conveying keys less frequently and indepedent from reference data enables a reduction in size of RIMs used to convey dynamic information and may result in a reduction in the size of aggregated data transferred to a verifier. CoRIMs themselves are signed and some means of conveying CoRIM verification keys is required, though ultimately some out-of-band mechanism is required at least for bootstrapping purposes. Relying parties may verify attestations from both hardware and software sources and some trust anchors may be used to verify attestations from both hardware and software sources, as well. The verification information included in a CoRIM optionally includes a trust anchor, leaving trust anchor management to other mechanisms. Additionally, the CoRIM verification-map structure is tied to CoMIDs, leaving no simple means to convey verification information for CoSWIDs [I-D.draft-ietf-sacm-coswid].

This document defines means to decouple TAs and CAs from reference data and adds support for constraining the use of trust anchors, chiefly by limiting the environments to which a set of trust anchors is applicable. This constraints mechanism is similar to that in

[fido-metadata] and [fido-service] and should align with existing attestation verification practices that tend to use per-vendor trust anchors. TA store instances may be further constrained using coarse-grained purpose values or a set of finer-grained permitted or excluded claims. The trust anchor formats supported by this draft allow for per-trust anchor constraints, if desired. Conveyance of trust anchors is the primary goal, CA certificates may optionally be included for convenience.

1.1. Constraints

This document aims to support different PKI architectures including scenarios with various combinations of the following characteristics:

* TA stores that contain a TA or set of TAs from a single organization

* TA stores that contain a set of TAs from multiple organizations

* TAs that issue certificates to CAs within the same organiation as the TA

* TAs that issue certificates to CAs from multiple organizations

* CAs that issue certificates that may be used to verify attestations or certificates from the same organization as the TA and CA

* CAs that issue certificates that may be used to verify attestations or certificates from multiple organizations

Subsequent specifications may define extensions to express constraints as well as processing rules for evaluating constraints expressed in TA stores, TAs, CA certificates and end entity (EE) certificates. Support for constraints is intended to enable misissued certificates to be rejected at verification time. Any public key that can be used to verify a certificate is assumed to also support verification of revocation information, subject to applicable constraints defined by the revocation mechanism.

2. Conventions and Definitions

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

3. Trust anchor management for RATS

Within RATS, trust anchors may be used to verify digital signatures for a variety of objects, including entity attestation tokens (EATs), CoRIMs, X.509 CA certificates (possibly containing endorsement information), X.509 EE certificates (possibly containing endorsement or attestation information), other attestation data, digital letters of approval [dloa], revocation information, etc. Depending on context, a raw public key may suffice or additional information may be required, such as subject name or subject public key identifier information found in an X.509 certificate. Trust anchors are usually aggregated into sets that are referred to as "trust anchor stores". Different trust anchor stores may serve different functional purposes.

Historically, trust anchors and trust anchor stores are not constrained other than by the context(s) in which a trust anchor store is used. The path validation algorithm in [RFC5280] only lists name, public key, public key algorithm and public key parameters as the elements of "trust anchor information". However, there are environments that do constrain trust anchor usage. The RPKI uses extensions from trust anchor certificates as defined in [RFC3779]. FIDO provides a type of constraint by grouping attestation verification root certificates by authenticator model in [fido-metadata].

This document aims to support each of these types of models by allowing constrained or unconstrained trust anchors to be grouped by abstract purpose, i.e., similar to traditional trust anchor stores, or grouped by a set of constraints, such as vendor name.

3.1. TA and CA conveyance

An unsigned concise TA stores object is a list of one or more TA stores, each represented below as a concise-ta-store-map element.

```
concise-ta-stores
  concise-ta-store-map #1
  ...
  concise-ta-store-map #n
```

Each TA store instance identifies a target environment and features one or more public keys. Optional constraints on usage may be defined as well.

```
concise-ta-store-map
  language
  store-identity
  target environment
  abstract coarse-grained constraints on TA store usage
  concrete fine-grained constraints on TA store usage
  public keys (possibly included per-instance constraints)
```

The following sections define the structures to support the concepts shown above.

3.1.1. The concise-ta-stores Container

The concise-ta-stores type is the root element for distrbuting sets of trust anchor stores. It contains one or more concise-ta-store-map elements where each element in the list identifies the environments for which a given set of trust anchors is applicable, along with any constraints.

```
concise-ta-stores = [+ concise-ta-store]
```

The $concise-tag-type-choice [I-D.draft-birkholz-rats-corim] is extended to include the concise-ta-stores structure. As shown in Section 4 of [I-D.draft-birkholz-rats-corim], the $concise-tag-type-choice type is used within the unsigned-corim-map structure, which is used within COSE-Sign1-corim structure. The COSE-Sign1-corim provides for integrity of the CoTS data. CoTS structures are not intended for use as stand-alone, unsigned structures. The signature on a CoTS instance SHOULD be verified using a TA associated with the cots purpose (Section 3.3.1).

```
$concise-tag-type-choice /= #6.TBD(bytes .cbor concise-ta-stores)
```

3.1.2. The concise-ta-store-map Container

A concise-ta-store-map is a trust anchor store where the applicability of the store is established by the tastore.environment field with optional constraints on use of trust anchors found in the tastore.keys field defined by the tastore.purpose, tastore.perm_claims and tastore.excl_claims fields.

```
concise-ta-store-map = {
 ? tastore.language => language-type
 ? tastore.store-identity => tag-identity-map
 tastore.environments => environment-group-list
 ? tastore.purposes => [+ $$tas-list-purpose]
 ? tastore.perm_claims => [+ $$claims-set-claims]
 ? tastore.excl_claims => [+ $$claims-set-claims]
 tastore.keys => cas-and-tas-map
}

; concise-ta-store-map indices
tastore.language = 0
tastore.store-identity = 1
tastore.environment = 2
tastore.purpose = 3
tastore.perm_claims = 4
tastore.excl_claims = 5
tastore.keys = 6
```

The following describes each member of the concise-ta-store-map.

tastore.language: A textual language tag that conforms with the IANA Language Subtag Registry [IANA.language-subtag-registry].

tastore.store-identity: A composite identifier containing identifying attributes that enable global unique identification of a TA store instance across versions and facilitate linking from other artifacts. The tag-identity-map type is defined in [I-D.draft-birkholz-rats-corim].

tastore.environment: A list of environment definitions that limit the contexts for which the tastore.keys list is applicable. If the tastore.environment is empty, TAs in the tastore.keys list may be used for any environment.

tastore.purpose: Contains a list of purposes (Section 3.3.1) for which the tastore.keys list may be used. When absent, TAs in the tastore.keys list may be used for any purpose. This field is simliar to the extendedKeyUsage extension defined in [RFC5280]. The initial list of purposes are: cots, corim, comid, coswid, eat, key-attestation, certificate

tastore.perm_claims: Contains a list of claim values (Section 3.3.2) [I-D.draft-ietf-rats-eat] for which tastore.keys list MAY be used to verify. When this field is absent, TAs in the tastore.keys list MAY be used to verify any claim subject to other restrictions.

tastore.excl_claims: Contains a list of claim values (Section 3.3.2) [I-D.draft-ietf-rats-eat] for which tastore.keys list MUST NOT be used to verify. When this field is absent, TAs in the tastore.keys list may be used to verify any claim subject to other restrictions.

tastore.keys: Contains a list of one or more TAs and an optional list of one or more CA certificates.

The perm_claims and excl_claims constraints MAY alternatively be expressed as extensions in a TA or CA. Inclusion of support here is intended as an aid for environments that find CBOR encoding support more readily available than DER encoding support.

3.1.3. The cas-and-tas-map Container

The cas-and-tas-map container provides the means of representing trust anchors and, optionally, CA certificates.

```
trust-anchor = [
  format => $pkix-ta-type
  data => bstr
]

cas-and-tas-map = {
 tastore.tas => [ + trust-anchor ]
 ? tastore.cas => [ + pkix-cert-data ]
}

; cas-and-tas-map indices
tastore.tas = 0
tastore.cas = 1

; format values
$pkix-ta-type /= tastore.pkix-cert-type
$pkix-ta-type /= tastore.pkix-tainfo-type
$pkix-ta-type /= tastore.pkix-spki-type

tastore.pkix-cert-type = 0
tastore.pkix-tainfo-type = 1
tastore.pkix-spki-type = 2

; certificate type
pkix-cert-data = bstr
```

The tastore.tas element is used to convey one or more trust anchors and an optional set of one or more CA certificates. TAs are implicitly trusted, i.e., no verification is required prior to use.

However, limitations on the use of the TA may be asserted in the corresponding concise-ta-store-map or within the TA itself. The tastore.cas field provides certificates that may be useful in the context where the corresponding concise-ta-store-map is used. These certificates are not implicitly trusted and MUST be validated to a trust anchor before use. End entity certificates SHOULD NOT appear in the tastore.cas list.

The structure of the data contained in the data field of a trust-anchor is indicated by the format field. The pkix-cert-type is used to represent a binary, DER-encoded X.509 Certificate as defined in section 4.1 of [RFC5280]. The pkix-key-type is used to represent a binary, DER-encoded SubjectPublicKeyInfo as defined in section 4.1 of [RFC5280]. The pkix-tainfo-type is used to represent a binary, DER-encoded TrustAnchorInfo as defined in section 2 of [RFC5914].

The $pkix-ta-type provides an extensible means for representing trust anchor information. It is defined here as supporting the pkix-cert-type, pkix-spki-type or pkix-tainfo-type. The pkix-spki-type may be used where only a raw pubilc key is necessary. The pkix-cert-type may be used for most purposes, including scenarios where a raw public key is sufficient and those where additional information from a certificate is required. The pkix-tainfo-type is included to support scenarios where constraints information is directly associated with a public key or certificate (vs. constraints for a TA set as provided by tastore.purpose, tastore.perm_claims and tastore.excl_claims).

The pkix-cert-data type is used to represent a binary, DER-encoded X.509 Certificate.

3.2. Environment definition

3.2.1. The environment-group-list Array

In CoRIM, "composite devices or systems are represented by a collection of Concise Module Identifiers (CoMID) and Concise Software Identifiers (CoSWID)". For trust anchor management purposes, targeting specific devices or systems may be too granular. For example, a trust anchor or set of trust anchors may apply to multiple device models or versions. The environment-map definition as used in a CoRIM is tightly bound to a CoMID. To allow for distribution of key material applicable to a specific or range of devices or software, the envrionment-group-list and environment-group-map are defined as below. These aim to enable use of coarse-grained naturally occurring values, like vendor, make, model, etc. to determine if a set of trust anchors is applicable to an environment.

```
environment-group-list = [* environment-group-list-map]

environment-group-list-map = {
  ? tastore.environment_map => environment-map,
  ? tastore.concise_swid_tag => abbreviated-swid-tag,
  ? tastore.named_ta_store => named-ta-store,
}

; environment-group-list-map indices
tastore.environment_map = 0
tastore.abbreviated_swid_tag = 1
tastore.named_ta_store = 2
```

An environment-group-list is a list of one or more environment-group-list-map elements that are used to determine if a given context is applicable. An empty list signifies all contexts SHOULD be considered as applicable.

An environment-group-list-map is one of environment-map[I-D.draft-birkholz-rats-corim], abbreviated-swid-tag-map (Section 3.2.2) or named-ta-store (Section 3.2.3).

As defined in [I-D.draft-birkholz-rats-corim], an envirionment-map may contain class-map, $instance-id-type-choice, $group-id-type-choice.

QUESTION: Should the above dispense with environment_map and concise_swid_tag and use or define some identity-focused structure with information common to both (possibly class-map from [I-D.draft-birkholz-rats-corim])? If not, should a more complete CoMID representation be used (instead of environment_map)?

3.2.2. The abbreviated-swid-tag-map Container

The abbreviated-swid-tag-map allows for expression of fields from a concise-swid-tag [I-D.draft-ietf-sacm-coswid] with all fields except entity designated as optional, compared to the concise-swid-tag definition that requires tag-id, tag-version and software-name to be present.

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

3.2.3. The named-ta-store Type

This specification allows for defining sets of trust anchors that are associated with an arbitrary name instead of relative to information typically expressed in a CoMID or CoSWID. Relying parties MUST be configured using the named-ta-store value to select a corresponding concise-ta-store-map for use.

```
named-ta-store = tstr
```

3.3. Constraints definition

3.3.1. The $$tas-list-purpose Type

The $$tas-list-purpose type provides an extensible means of expressions actions for which the corresponding keys are applicable. For example, trust anchors in a concise-ta-store-map with purpose field set to eat may not be used to verify certification paths. Extended key usage values corresponding to each purpose listed below (except for certificate) are defined in a companion specification.

```
$$tas-list-purpose /= "cots"
$$tas-list-purpose /= "corim"
$$tas-list-purpose /= "coswid"
$$tas-list-purpose /= "eat"
$$tas-list-purpose /= "key-attestation"
$$tas-list-purpose /= "certificate"
$$tas-list-purpose /= "dloa"
```

TODO - define verification targets for each purpose. QUESTION - should this have a registry?

3.3.2. Claims

A concise-ta-store-map may include lists of permitted and/or excluded claims [I-D.draft-ietf-rats-eat] that limit the applicability of trust anchors present in a cas-and-tas-map. A subsequent specification will define processing rules for evaluating constraints expressed in TA stores, TAs, CA certificates and end entity certificates.

3.4. Processing a concise-ta-stores RIM

When verifying a signature using a public key that chains back to a concise-ta-stores instance, elements in the concise-ta-stores array are processed beginning with the first element and proceeding until either a matching set is found that serves the desired purpose or no more elements are available. Each element is evaluated relative to the context, i.e., environment, purpose, artifact contents, etc.

For example, when verifying a CoRIM, each element in a triples-group MUST have an environment value that matches an environment-group-list-map element associated with the concise-ta-store-map containing the trust anchor used to verify the CoMID. Similarly, when verifying a CoSWID, the values in a abbreviated-swid-tag element from the concise-ta-store-map MUST match the CoSWID tag being verified. When verifying a certificate with DICE attestation extension, the information in each DiceTcbInfo element MUST be consistent with an environment-group-list-map associated with the concise-ta-store-map.

3.5. Verifying a concise-ta-stores RIM

[I-D.draft-birkholz-rats-corim] defers verification rules to [RFC8152] and this document follows suit with the additional recommendation that the public key used to verify the RIM SHOULD be present in or chain to a public key present in a concise-ta-store-map with purpose set to cots.

4. CDDL definitions

The CDDL definitions present in this document are provided below. Definitions from [I-D.draft-birkholz-rats-corim] are not repeated here.

```
concise-ta-stores = [+ concise-ta-store-map]
$concise-tag-type-choice /= #6.TBD(bytes .cbor concise-ta-stores)

concise-ta-store-map = {
 ? tastore.language => language-type
 ? tastore.store-identity => tag-identity-map
 tastore.environments => environment-group-list
 ? tastore.purposes => [+ $$tas-list-purpose]
 ? tastore.perm_claims => [+ $$claims-set-claims]
 ? tastore.excl_claims => [+ $$claims-set-claims]
 tastore.keys => cas-and-tas-map
}

; concise-ta-store-map indices
tastore.language = 0
tastore.store-identity = 1
tastore.environment = 2
tastore.purpose = 3
tastore.perm_claims = 4
tastore.excl_claims = 5
tastore.keys = 6

trust-anchor = [
  format => $pkix-ta-type
  data => bstr
]

cas-and-tas-map = {
 tastore.tas => [ + trust-anchor ]
 ? tastore.cas => [ + pkix-cert-type ]
}

; cas-and-tas-map indices
tastore.tas = 0
tastore.cas = 1

; format values
$pkix-ta-type /= tastore.pkix-cert-type
$pkix-ta-type /= tastore.pkix-tainfo-type
$pkix-ta-type /= tastore.pkix-spki-type

tastore.pkix-cert-type = 0
tastore.pkix-tainfo-type = 1
tastore.pkix-spki-type = 2

; certificate type
pkix-cert-data = bstr
```

```
environment-group-list = [* environment-group-list-map]

environment-group-list-map = {
  ? environment-map => environment-map,
  ? concise-swid-tag => abbreviated-swid-tag,
  ? named-ta-store => named-ta-store,
}

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

named-ta-store = tstr

$tas-list-purpose /= "cots"
$tas-list-purpose /= "corim"
$tas-list-purpose /= "comid"
$tas-list-purpose /= "coswid"
$tas-list-purpose /= "eat"
$tas-list-purpose /= "key-attestation"
$tas-list-purpose /= "certificate"
$tas-list-purpose /= "dloa"
```

5. Examples

The following examples are isolated concise-ta-store-map instances shown as JSON for ease of reading. The final example is an ASCII hex representation of a CBOR-encoded concise-ta-stores instance containing each example below (and using a placeholder value for the concise-ta-stores tag).

The TA store below contains a TA from a single organization ("Zesty Hands, Inc,") that is used to verify CoRIMs for that organization. Because this TA does not verify certificates, a bare public key is appropriate. It features a tag identity field containing a UUID for the tag identity and a version indication.


```
{
  "tag-identity": {
    "id": "ab0f44b1-bfdc-4604-ab4a-30f80407ebcc",
    "version": 5
  },
  "environments": [
    {
      "environment": {
        "class": {
          "vendor": "Worthless Sea, Inc."
        }
      }
    }
  ],
  "keys": {
    "tas": [
      {
        "format": 2,
        "data":
"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAErYoMAdqe2gJT3CvCcifZxyE9+
N8T6Jy5zbeo5LYtnOipmi1wXA9/gNtlwAbRCRQitH/GEcvUaGlzPZxIOITV/g=="
      }
    ]
  }
}
```


The TA store below features three TAs from different organizations grouped as a TA store with the name "Miscellaneous TA Store". The first TA is an X.509 certificate. The second and third TAs are TrustAnchorInfo objects containing X.509 certificates. Though not shown in this example, constraints could be added to the TrustAnchorInfo elements, i.e., to restrict verification to attestations asserting a specific vendor name. It features a tag identity field containing a string as the tag identity with no version field present.


```
{
  "tag-identity": {
    "id": "some_tag_identity"
  },
  "environments": [
    {
```

```
        "namedtastore": "Miscellaneous TA Store"
      }
    ],
    "keys": {
      "tas": [
        {
          "format": 0,
          "data":
          "
MIIBvTCCAWSgAwIBAgIVANCdkL89UlzHc9Ui7XfVniK7pFuIMAoGCCqGSM49BAMCMD4
xCzAJBgNVBAYMAlVTMRAwDgYDVQQKDAdFeGFtcGxlMR0wGwYDVQQDDBRFeGFtcGxlIF
RydXN0IEFuY2hvcjAeFw0yMjA1MTkxNTEzMDdaFw0zMjA1MTYxNTEzMDdaMD4xCzAJB
gNVBAYMAlVTMRAwDgYDVQQKDAdFeGFtcGxlMR0wGwYDVQQDDBRFeGFtcGxlIFRydXN0
IEFuY2hvcjBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABONRqhA5JAekvQN8oLwRVND
nAfBnTznLLE+SEGks677sHSeXfcVhZXUeDiN7/fsVNumaiEWRQpZh3zXPwL8rUMyjPz
A9MB0GA1UdDgQWBBQBXEXJrLBGKnFd1xCgeMAVSfEBPzALBgNVHQ8EBAMCAoQwDwYDV
R0TAQH/BAUwAwEB/zAKBggqhkjOPQQDAgNHADBEAiALBidABsfpzG0lTL9Eh9b6AUbq
nzF+koEZbgvppvvt9QIgVoE+bhEN0j6wSPzePjLrEdD+PEgyjHJ5rbA11SPq/1M="
        },
        {
          "format": 1,
          "data":
          "
ooICtjCCArIwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAASXz21w12owQAx58euratY
WiHEkhxDU9MEgetrvAtGYZxNnkfLCsp9vLcw8ISXC8tL97k9ZCUtnr0MzLw37XKRABB
T22tHlEou/DenpU0Ozccb3/+fibjCCAj0wUjELMAkGA1UEBgwCVVMxGjAYBgNVBAoME
Vplc3R5IEhhbmRzLCBJbmMuMScwJQYDVQQDDB5aZXN0eSBIYW5kcywgSW5jLiBUcnVz
dCBBbmNob3KgggHlMIIBi6ADAgECAhQL3EqgUXlQPljyddVSRnNHvK1MzAKBggqhkjO
PQQDAjBSMQswCQYDVQQGDAJVUzEaMBgGA1UECgwRWmVzdHkgSGFuZHMsIEluYy4xJzA
lBgNVBAMMHlplc3R5IEhhbmRzLCBJbmMuIFRydXN0IEFuY2hvcjAeFw0yMjA1MTkxNT
EzMDdaFw0zMjA1MTYxNTEzMDdaMFIxCzAJBgNVBAYMAlVTMRowGAYDVQQKDBFaZXN0e
SBIYW5kcywgSW5jLjEnMCUGA1UEAwweWmVzdHkgSGFuZHMsIEluYy4gVHJ1c3QgQW5j
aG9yMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEl89tcNdqMEAMefHrq2rWFohxJIc
Q1PTBIHra7wLRmGcTZ5HywrKfby3MPCElwvLS/e5PWQlLZ69DMy8N+1ykQKM/MD0wHQ
YDVR0OBBYEFPba0eUSi78N6elTQ7Nxxvf/5+JuMAsGA1UdDwQEAwIChDAPBgNVHRMBA
f8EBTADAQH/MAoGCCqGSM49BAMCA0gAMEUCIB2li+f6RCxs2EnvNWciSpIDwiUViWay
Gv1A8xks80eYAiEAmCez4KGrolFKOZT6bvqf1sYQuJBfvtk/y1JQdUvoqlg="
        },
        {
          "format": 1,
          "data":
          "
ooIC1TCCAtEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATN0f5kzywEzZOYbaV23O3
N8cku39JoLNjlHPwECbXDDWp0LpAO1z248/hoy6UW/TZMTPPR/93XwHsG16mSFy8XBB
SKhM/5gJWjvDbW7qUY1peNm9cfYDCCAlwwXDELMAkGA1UEBgwCVVMxHzAdBgNVBAoMF
lNub2JiaXNoIEFwcGFyZWwsIEluYy4xLDAqBgNVBAMMI1Nub2JiaXNoIEFwcGFyZWws
IEluYy4gVHJ1c3QgQW5jaG9yoIIB+jCCAZ+gAwIBAgIUEBuTRGXAEEVEHhu4xafAnqm
+qYgwCgYIKoZIzj0EAwIwXDELMAkGA1UEBgwCVVMxHzAdBgNVBAoMFlNub2JiaXNoIE
```

FwcGFyZWwsIEluYy4xLDAqBgNVBAMMI1Nub2JiaXNoIEFwcGFyZWwsIEluYy4gVHJ1c
3QgQW5jaG9yMB4XDTIyMDUxOTE1MTMwOFoXDTMyMDUxNjE1MTMwOFowXDELMAkGA1UE
BgwCVVMxHzAdBgNVBAoMFlNub2JiaXNoIEFwcGFyZWwsIEluYy4xLDAqBgNVBAMMI1N
ub2JiaXNoIEFwcGFyZWwsIEluYy4gVHJ1c3QgQW5jaG9yMFkwEwYHKoZIzj0CAQYIKo
ZIzj0DAQcDQgAEzdH+ZM8sBM2TmG2ldtztzfHJLt/SaCzY5Rz8BAm1ww1qdC6QDtc9u
PP4aMulFv02TEzz0f/d18B7BtepkhcvF6M/MD0wHQYDVR0OBBYEFIqEz/mAlaO8Ntbu
pRjWl42b1x9gMAsGA1UdDwQEAwIChDAPBgNVHRMBAf8EBTADAQH/MAoGCCqGSM49BAM
CA0kAMEYCIQC2cf43f3PPlCO6/dxv40ftIgxxToKHF72UzENv7+y4ygIhAIGtC/r6SG
aFMaP7zD2EloBuIXTtyWu8Hwl+YGdXRY93"
      }
    ]
  }
}

The TA Store below features one TA with an environment targeting CoSWIDs with entity named "Zesty Hands, Inc," and one permitted EAT claim for software named "Bitter Paper".

```
{
  "environments": [
    {
      "swidtag": {
        "entity": [
          {
            "entity-name": "Zesty Hands, Inc.",
            "role": "softwareCreator"
          }
        ]
      }
    }
  ],
  "permclaims": [
    {
      "swname": "Bitter Paper"
    }
  ],
  "keys": {
    "tas": [
      {
        "format": 0,
        "data":
        "
MIIB5TCCAYugAwIBAgIUC9xKoFF5UD5Y8nXVUkZzR7yvtTMwCgYIKoZIzj0EAwI
wUjELMAkGA1UEBgwCVVMxGjAYBgNVBAoMEVplc3R5IEhhbmRzLCBJbmMuMScwJQ
YDVQQDDB5aZXN0eSBIYW5kcywgSW5jLiBUcnVzdCBBbmNob3IwHhcNMjIwNTE5M
TUxMzA3WhcNMzIwNTE2MTUxMzA3WjBSMQswCQYDVQQGDAJVUzEaMBgGA1UECgwR
WmVzdHkgSGFuZHMsIEluYy4xJzAlBgNVBAMMHlplc3R5IEhhbmRzLCBJbmMuIFR
ydXN0IEFuY2hvcjBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABJfPbXDXajBADH
nx66tq1haIcSSHENT0wSB62u8C0ZhnE2eR8sKyn28tzDwhJcLy0v3uT1kJS2evQ
zMvDftcpECjPzA9MB0GA1UdDgQWBBT22tHlEou/DenpU0Ozccb3/+fibjALBgNV
HQ8EBAMCAoQwDwYDVR0TAQH/BAUwAwEB/zAKBggqhkjOPQQDAgNIADBFAiAdpYv
n+kQsbNhJ7zVnIkqSA8IlFYlmshr9QPMZLPNHmAIhAJgns+Chq6JRSjmU+m76n9
bGELiQX77ZP8tSUHVL6KpY"
      }
    ]
  }
}
```


The dump below shows the COSE-Sign1-corim contents from the ASCII hex above. A full base64-encoded version of this example is given in Appendix A.

```
18([h'
A3012603746170706C69636174696F6E2F72696D2B63626F72085841A200A20
07441434D45204C7464207369676E696E67206B657901D8207468747470733A
2F2F61636D652E6578616D706C6501A200C11A61CE480001C11A69546780',
{},
h'
A30050EBA916FB1E3E42679214E07E1A9BF9130181590A56D901FB83A301A20
050FB51FAC913C546C39390DC306B167F5A01050281A101A100A10173576F72
74686C657373205365612C20496E632E06A100818202585B3059301306072A8
648CE3D020106082A8648CE3D03010703420004AD8A0C01DA9EDA0253DC2BC2
7227D9C7213DF8DF13E89CB9CDB7A8E4B62D9CE8A99A2D705C0F7F80DB65C00
6D1091422B47FC611CBD46869733D9C483884D5FEA301A10071736F6D655F74
61675F6964656E746974790281A103764D697363656C6C616E656F757320544
12053746F726506A1008382005901C1308201BD30820164A003020102021500
D09D90BF3D525CC773D522ED77D59E22BBA45B88300A06082A8648CE3D04030
2303E310B300906035504060C0255533110300E060355040A0C074578616D70
6C65311D301B06035504030C144578616D706C652054727573742041 6E63686
F72301E170D323230353139313531333037 5A170D3332303531363135313330
375A303E310B300906035504060C0255533110300E060355040A0C074578616
D706C65311D301B06035504030C144578616D706C652054727573742041 6E63
686F723059301306072A8648CE3D020106082A8648CE3D03010703420004E35
1AA10392407A4BD037CA0BC1154D0E701F0674F39CB2C4F9210692CEBBEEC1D
27977DC56165751E0E237BFDFB1536E99A884591429661DF35CFC0BF2B50CCA
33F303D301D0603551D0E04160414015C45C9ACB0462A715DD710A078C01549
F1013F300B0603551D0F040403020284300F0603551D130101FF04053003010
1FF300A06082A8648CE3D040302034700304402200B06274006C7E9CC6D254C
BF4487D6FA0146EA9F317E9281196E0BE9A6FBEDF5022056813E6E110DD23EB
048FCDE3E32EB11D0FE3C48328C7279ADB035D523EAFF5382015902BAA28202
B6308202B23059301306072A8648CE3D020106082A8648CE3D0301070342000
497CF6D70D76A30400C79F1EBAB6AD6168871248710D4F4C1207ADAEF02D198
67136791F2C2B29F6F2DCC3C2125C2F2D2FDEE4F59094B67AF43332F0DFB5CA
4400414F6DAD1E5128BBF0DE9E95343B371C6F7FFE7E26E3082023D3052310B
300906035504060C025553311A3018060355040A0C115A657374792048616E6
4732C20496E632E312730250603550403 0C1E5A657374792048616E64732C20
496E632E2054727573742041 6E63686F72A08201E53082018BA003020102021
40BDC4AA05179503E58F275D552467347BCAFB533300A06082A8648CE3D0403
023052310B300906035504060C025553311A3018060355040A0C115A6573747
92048616E64732C20496E632E312730250603550403 0C1E5A65737479204861
6E64732C20496E632E2054727573742041 6E63686F72301E170D32323035313
93135313330375A170D3332303531363135313330375A3052310B3009060355
04060C025553311A3018060355040A0C115A657374792048616E64732C20496
E632E312730250603550403 0C1E5A657374792048616E64732C20496E632E20
547275737420416E63686F723059301306072A8648CE3D020106082A8648CE3
D0301070342000497CF6D70D76A30400C79F1EBAB6AD6168871248710D4F4C1
207ADAEF02D19867136791F2C2B29F6F2DCC3C2125C2F2D2FDEE4F59094B67A
F43332F0DFB5CA440A33F303D301D0603551D0E04160414F6DAD1E5128BBF0D
E9E95343B371C6F7FFE7E26E300B0603551D0F040403020284300F0603551D1
30101FF040530030101FF300A06082A8648CE3D040302034800304502201DA5
```

8BE7FA442C6CD849EF3567224A9203C225158966B21AFD40F3192CF34798022
1009827B3E0A1ABA2514A3994FA6EFA9FD6C610B8905FBED93FCB5250754BE8
AA5882015902D9A28202D5308202D13059301306072A8648CE3D020106082A8
648CE3D03010703420004CDD1FE64CF2C04CD93986DA576DCEDCDF1C92EDFD2
682CD8E51CFC0409B5C30D6A742E900ED73DB8F3F868CBA516FD364C4CF3D1F
FDDD7C07B06D7A992172F1704148A84CFF98095A3BC36D6EEA518D6978D9BD7
1F603082025C305C310B300906035504060C025553311F301D060355040A0C1
6536E6F6262697368204170706172656C2C20496E632E312C302A0603550403
0C23536E6F6262697368204170706172656C2C20496E632E205472757374204
16E63686F72A08201FA3082019FA0030201020214101B934465C01045441E1B
B8C5A7C09EA9BEA988300A06082A8648CE3D040302305C310B3009060355040
60C025553311F301D060355040A0C16536E6F6262697368204170706172656C
2C20496E632E312C302A06035504030C23536E6F6262697368204170706172
56C2C20496E632E2054727573742041 6E63686F72301E170D32323035313931
35313330385A170D3332303531363135313330385A305C310B3009060355040
60C025553311F301D060355040A0C16536E6F6262697368204170706172656C
2C20496E632E312C302A06035504030C23536E6F6262697368204170706172
56C2C20496E632E2054727573742041 6E63686F723059301306072A8648CE3D
020106082A8648CE3D03010703420004CDD1FE64CF2C04CD93986DA576DCEDC
DF1C92EDFD2682CD8E51CFC0409B5C30D6A742E900ED73DB8F3F868CBA516FD
364C4CF3D1FFDDD7C07B06D7A992172F17A33F303D301D0603551D0E0416041
48A84CFF98095A3BC36D6EEA518D6978D9BD71F60300B0603551D0F04040302
0284300F0603551D130101FF040530030101FF300A06082A8648CE3D0403020
349003046022100B671FE377F73CF9423BAFDDC6FE347ED220C714E828717BD
94CC436FEFECB8CA02210081AD0BFAFA48668531A3FBCC3D8496806E2174EDC
96BBC1F097E606757458F77A30281A102A102A2181F715A657374792048616E
64732C20496E632E1821020481A11903E66C4269747465722050617065720 6A
1008182005901E9308201E53082018BA00302010202140BDC4AA05179503E58
F275D552467347BCAFB533300A06082A8648CE3D0403023052310B300906035
504060C025553311A3018060355040A0C115A657374792048616E64732C2049
6E632E31273025060355040 30C1E5A657374792048616E64732C20496E632E2
0547275737420416E63686F72301E170D3232303531393135313330375A170D
3332303531363135313330375A3052310B300906035504060C025553311A301
8060355040A0C115A657374792048616E64732C20496E632E31273025060355
04030C1E5A657374792048616E64732C20496E632E2054727573742041 6E636
86F723059301306072A8648CE3D020106082A8648CE3D0301070342000497CF
6D70D76A30400C79F1EBAB6AD6168871248710D4F4C1207ADAEF02D19867136
791F2C2B29F6F2DCC3C2125C2F2D2FDEE4F59094B67AF43332F0DFB5CA440A3
3F303D301D0603551D0E04160414F6DAD1E5128BBF0DE9E95343B371C6F7FFE
7E26E300B0603551D0F040403020284300F0603551D130101FF040530030101
FF300A06082A8648CE3D0403020348003045022 01DA58BE7FA442C6CD849EF3
567224A9203C225158966B21AFD40F3192CF347980221009827B3E0A1ABA251
4A3994FA6EFA9FD6C610B8905FBED93FCB5250754BE8AA5804A200C11A61CE4
80001C11A69546780’,
h’
19E82D7A5C7A73B44F06305AECF0EF8CF8764286323F6D2BA27D7291F92FF5B
0CF789F6FF88B7E2EE8EF262B4FA1DFD7D7AFB0AE2C0062C98DB332243B3E99
94’])

6. Security Considerations

As a profile of CoRIM, the security considerations from [I-D.draft-birkholz-rats-corim] apply.

As a means of managing trust anchors, the security considerations from [RFC6024] and [RFC5934] apply. a CoTS signer is roughly analogous to a "management trust anchor" as described in [RFC5934].

7. IANA Considerations

7.1. CoRIM CBOR Tag Registration

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

Tag	Data Item	Semantics
507	tagged array	Concise Trust Anchor Stores (CoTS)

Table 1

8. References

8.1. Normative References

[I-D.draft-birkholz-rats-corim]
Birkholz, H., Fossati, T., Deshpande, Y., Smith, N., and W. Pan, "Concise Reference Integrity Manifest", Work in Progress, Internet-Draft, draft-birkholz-rats-corim-03, 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-corim-03>.

[I-D.draft-ietf-rats-eat]
Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-16, 9 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-16>.

[I-D.draft-ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-22, 20 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-sacm-coswid-22>.

[IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", 19 September 2013, <https://www.iana.org/assignments/cbor-tags>.

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

[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>.

[RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <https://www.rfc-editor.org/rfc/rfc5914>.

[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>.

[RFC8949] 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>.

8.2. Informative References

[dloa] GlobalPlatform, "GlobalPlatform Card - Digital Letter of Approval Version 1.0", November 2015, <https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf>.

[fido-metadata]
FIDO Alliance, "FIDO Metadata Statement", May 2021, <https://fidoalliance.org/specs/mds/fido-metadata-statement-v3.0-ps-20210518.html>.

[fido-service]
FIDO Alliance, "FIDO Metadata Service", May 2021, <https://fidoalliance.org/specs/mds/fido-metadata-service-v3.0-ps-20210518.html>.

[I-D.draft-ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-22, 28 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-22>.

[RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", RFC 3779, DOI 10.17487/RFC3779, June 2004, <https://www.rfc-editor.org/rfc/rfc3779>.

[RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", RFC 5934, DOI 10.17487/RFC5934, August 2010, <https://www.rfc-editor.org/rfc/rfc5934>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/rfc/rfc8152>.

Acknowledgments

TODO acknowledge.

Appendix A

The base64 encoded data below represents a signed CoRIM that features a concise-ta-stores containing the three examples shown above.

0oRYXaMBJgN0YXBwbGljYXRpb24vcmltK2Nib3IIWEGiAKIAdEFDTUUgTHRkIHN
pZ25pbmcga2V5AdggdGh0dHBzOi8vYWNtZS5leGFtcGxlAaIAwRphzkgAAcEaaV
RngKBZCn6jAFDrqRb7Hj5CZ5IU4H4am/kTAYFZClbZAfuDowGiAFD7UfrJE8VGw
5OQ3DBrFn9aAQUCgaEBoQChAXNXb3J0aGxlc3MgU2VhLCBJbmMuBqEAgYICWFsw
WTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAStigwB2p7aAlPcK8JyJ9nHIT343xP
onLnNt6jkti2c6KmaLXBcD3+A22XABtEJFCK0f8YRy9RoaXM9nEg4hNX+owGhAH
Fzb21lX3RhZ19pZGVudGl0eQKBoQN2TWlzY2VsbGFuZW91cyBUQSBTdG9yZQahA
IOCAFkBwTCCAb0wggFkoAMCAQICFQDQnZC/PVJcx3PVIu131Z4iu6RbiDAKBggq

hkjOPQQDAjA+MQswCQYDVQQGDAJVUzEQMA4GA1UECgwHRXhhbXBsZTEdMBsGA1U
EAwwURXhhbXBsZSBUcnVzdCBBbmNob3IwHhcNMjIwNTE5MTUxMzA3WhcNMzIwNT
E2MTUxMzA3WjA+MQswCQYDVQQGDAJVUzEQMA4GA1UECgwHRXhhbXBsZTEdMBsGA
1UEAwwURXhhbXBsZSBUcnVzdCBBbmNob3IwWTATBgcqhkjOPQIBBggqhkjOPQMB
BwNCAATjUaoQOSQHpL0DfKC8EVTQ5wHwZ085yyxPkhBpLOu+7B0nl33FYWV1Hg4
je/37FTbpmohFkUKWYd81z8C/K1DMoz8wPTAdBgNVHQ4EFgQUAVxFyaywRipxXd
cQoHjAFUnxAT8wCwYDVR0PBAQDAgKEMA8GA1UdEwEB/wQFMAMBAf8wCgYIKoZIz
j0EAwIDRwAwRAIgCwYnQAbH6cxtJUy/RIfW+gFG6p8xfpKBGW4L6ab77fUCIFaB
Pm4RDdI+sEj83j4y6xHQ/jxIMoxyea2wNdUj6v9TggFZArqiggK2MIICsjBZMBM
GByqGSM49AgEGCCqGSM49AwEHA0IABJfPbXDXajBADHnx66tq1haIcSSHENT0wS
B62u8C0ZhnE2eR8sKyn28tzDwhJcLy0v3uT1kJS2evQzMvDftcpEAEFPba0eUSi
78N6elTQ7Nxxvf/5+JuMIICPTBSMQswCQYDVQQGDAJVUzEaMBgGA1UECgwRWmVz
dHkgSGFuZHMsIEluYy4xJzAlBgNVBAMMHlplc3R5IEhhbmRzLCBJbmMuIFRydXN
0IEFuY2hvcqCCAeUwggGLoAMCAQICFAvcSqBReVA+WPJ11VJGc0e8r7UzMAoGCC
qGSM49BAMCMFIxCzAJBgNVBAYMAlVTMRowGAYDVQQKDBFaZXN0eSBIYW5kcywgS
W5jLjEnMCUGA1UEAwweWmVzdHkgSGFuZHMsIEluYy4gVHJ1c3QgQW5jaG9yMB4X
DTIyMDUxOTE1MTMwN1oXDTMyMDUxNjE1MTMwN1owUjELMAkGA1UEBgwCVVMxGjA
YBgNVBAoMEVplc3R5IEhhbmRzLCBJbmMuMScwJQYDVQQDDB5aZXN0eSBIYW5kcy
wgSW5jLiBUcnVzdCBBbmNob3IwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAASXz
21w12owQAx58euratYWiHEkhxDU9MEgetrvAtGYZxNnkfLCsp9vLcw8ISXC8tL9
7k9ZCUtnr0MzLw37XKRAoz8wPTAdBgNVHQ4EFgQU9trR5RKLvw3p6VNDs3HG9//
n4m4wCwYDVR0PBAQDAgKEMA8GA1UdEwEB/wQFMAMBAf8wCgYIKoZIzj0EAwIDSA
AwRQIgHaWL5/pELGzYSe81ZyJKkgPCJRWJZrIa/UDzGSzzR5gCIQCYJ7PgoauiU
Uo5lPpu+p/WxhC4kF++2T/LUlB1S+iqWIIBWQLZooIC1TCCAtEwWTATBgcqhkjO
PQIBBggqhkjOPQMBBwNCAATN0f5kzywEzZOYbaV23O3N8cku39JoLNjlHPwECbX
DDWp0LpAO1z248/hoy6UW/TZMTPPR/93XwHsG16mSFy8XBBSKhM/5gJWjvDbW7q
UY1peNm9cfYDCCAlwwXDELMAkGA1UEBgwCVVMxHzAdBgNVBAoMFlNub2JiaXNoI
EFwcGFyZWwsIEluYy4xLDAqBgNVBAMMI1Nub2JiaXNoIEFwcGFyZWwsIEluYy4g
VHJ1c3QgQW5jaG9yoIIB+jCCAZ+gAwIBAgIUEBuTRGXAEEVEHhu4xafAnqm+qYg
wCgYIKoZIzj0EAwIwXDELMAkGA1UEBgwCVVMxHzAdBgNVBAoMFlNub2JiaXNoIE
FwcGFyZWwsIEluYy4xLDAqBgNVBAMMI1Nub2JiaXNoIEFwcGFyZWwsIEluYy4gV
HJ1c3QgQW5jaG9yMB4XDTIyMDUxOTE1MTMwOFoXDTMyMDUxNjE1MTMwOFowXDEL
MAkGA1UEBgwCVVMxHzAdBgNVBAoMFlNub2JiaXNoIEFwcGFyZWwsIEluYy4xLDA
qBgNVBAMMI1Nub2JiaXNoIEFwcGFyZWwsIEluYy4gVHJ1c3QgQW5jaG9yMFkwEw
YHKoZIzj0CAQYIKoZIzj0DAQcDQgAEzdH+ZM8sBM2TmG2ldtztzfHJLt/SaCzY5
Rz8BAm1ww1qdC6QDtc9uPP4aMulFv02TEzz0f/d18B7BtepkhcvF6M/MD0wHQYD
VR0OBBYEFIqEz/mAlaO8NtbupRjWl42b1x9gMAsGA1UdDwQEAwIChDAPBgNVHRM
BAf8EBTADAQH/MAoGCCqGSM49BAMCA0kAMEYCIQC2cf43f3PPlCO6/dxv40ftIg
xxToKHF72UzENv7+y4ygIhAIGtC/r6SGaFMaP7zD2EloBuIXTtyWu8Hwl+YGdXR
Y93owKBoQKhAqIYH3FaZXN0eSBIYW5kcywgSW5jLhghAgSBoRkD5mxCaXR0ZXIg
UGFwZXIGoQCBggBZAekwggHlMIIBi6ADAgECAhQL3EqgUXlQPljyddVSRnNHvK+
1MzAKBggqhkjOPQQDAjBSMQswCQYDVQQGDAJVUzEaMBgGA1UECgwRWmVzdHkgSG
FuZHMsIEluYy4xJzAlBgNVBAMMHlplc3R5IEhhbmRzLCBJbmMuIFRydXN0IEFuY
2hvcjAeFw0yMjA1MTkxNTEzMDdaFw0zMjA1MTYxNTEzMDdaMFIxCzAJBgNVBAYM
AlVTMRowGAYDVQQKDBFaZXN0eSBIYW5kcywgSW5jLjEnMCUGA1UEAwweWmVzdHk
gSGFuZHMsIEluYy4gVHJ1c3QgQW5jaG9yMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ
cDQgAEl89tcNdqMEAMefHrq2rWFohxJIcQ1PTBIHra7wLRmGcTZ5HywrKfby3MP
CElwvLS/e5PWQlLZ69DMy8N+1ykQKM/MD0wHQYDVR0OBBYEFPba0eUSi78N6elT

Q7Nxxvf/5+JuMAsGA1UdDwQEAwIChDAPBgNVHRMBAf8EBTADAQH/MAoGCCqGSM4
9BAMCA0gAMEUCIB2li+f6RCxs2EnvNWciSpIDwiUViWayGv1A8xks80eYAiEAmC
ez4KGrolFKOZT6bvqf1sYQuJBfvtk/y1JQdUvoqlgEogDBGmHOSAABwRppVGeAW
EAZ6C16XHpztE8GMFrs8O+M+HZChjI/bSuifXKR+S/1sM94n2/4i34u6O8mK0+h
39fXr7CuLABiyY2zMiQ7PpmU

Authors' Addresses

Carl Wallace
Red Hound Software
United States of America
Email: carl@redhoundsoftware.com

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA 20170
United States of America
Email: housley@vigilsec.com

Thomas Fossati
arm
Email: Thomas.Fossati@arm.com

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