

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 27 January 2022

H. Birkholz
Fraunhofer SIT
T. Fossati
Y. Deshpande
Arm Limited
N. Smith
Intel
W. Pan
Huawei Technologies
26 July 2021

Concise Reference Integrity Manifest
draft-birkholz-rats-corim-01

Abstract

Remote Attestation Procedures (RATS) enable Relying Parties to put trust in the trustworthiness of a remote Attester and therefore to decide if to engage in secure interactions with it - or not. Evidence about trustworthiness can be rather complex, voluminous or Attester-specific. As it is deemed unrealistic that every Relying Party is capable of the appraisal of Evidence, that burden is taken on by a Verifier. In order to conduct Evidence appraisal procedures, a Verifier requires not only fresh Evidence from an Attester, but also trusted Endorsements and Reference Values from Endorsers, such as manufacturers, distributors, or 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 27 January 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Requirements Notation	4
2. Concise Reference Integrity Manifests	4
2.1. Typographical Conventions	5
2.2. Prefixes and Namespaces	6
2.3. Extensibility	7
2.4. Concise RIM Extension Points	7
2.5. CDDL Generic Types	8
2.5.1. Non-Empty	8
2.5.2. One-Or-More	8
3. Concise RIM Data Definition	9
3.1. The signed-corim Container	9
3.1.1. The corim-meta-map Container	9
3.1.2. The corim-entity-map Container	10
3.1.3. The validity-map Container	10
3.2. The unsigned-corim-map Container	11
3.2.1. The corim-locator-map Container	11
3.3. The concise-mid-tag Container	12
3.4. The tag-identity-map Container	12
3.5. The entity-map Container	13

3.6.	The linked-tag-map Container	14
3.7.	The triples-map Container	14
3.8.	The environment-map Container	15
3.9.	The class-map Container	16
3.10.	The measurement-map and measurement-values-map Containers	17
3.10.1.	The version-map Container	19
3.10.2.	The svn-type-choice Enumeration	19
3.10.3.	The raw-value-group Container	20
3.10.4.	The ip-addr-type-choice Enumeration	20
3.10.5.	The mac-addr-type-choice Enumeration	20
3.11.	The verification-key-map Container	21
4.	Full CDDL Definition	21
5.	Privacy Considerations	35
6.	Security Considerations	35
7.	IANA Considerations	35
7.1.	COSE Header Parameters Registry	35
7.2.	CoRIM Map Items Registry	35
7.3.	CoRIM Entity-Map Items Registry	36
7.4.	CoRIM Entity-Types Registry	37
7.5.	CoMID Map Items Registry	38
7.6.	CoMID Entity-Map Items Registry	39
7.7.	CoMID Triples-Map Items Registry	40
7.8.	CoMID Measurement-Values-Map Items Registry	41
7.9.	CoMID Tag-Relationship-Types Registry	42
7.10.	CoMID Role-Types Registry	43
7.11.	rim+cbor Media Type Registration	44
7.12.	CoAP Content-Format Registration	45
7.13.	CoRIM CBOR Tag Registration	46
7.14.	CoMID CBOR Tag Registration	46
8.	References	47
8.1.	Normative References	47
8.2.	Informative References	48
	Authors' Addresses	48

1. Introduction

The Remote Attestation Procedures (RATS) architecture [I-D.ietf-rats-architecture] describes appraisal procedures for attestation Evidence and Attestation Results. Appraisal procedures for Evidence are conducted by Verifiers and are intended to assess the trustworthiness of a remote peer. Appraisal procedures for Attestation Results are conducted by Relying Parties and are intended to operationalize the assessment about a remote peer and to act appropriately based on the assessment. In order to enable their intent, appraisal procedures consume Appraisal Policies, Reference Values, and Endorsements.

This document specifies a binary encoding for Reference Values using the Concise Binary Object Representation (CBOR). The encoding is based on three parts that are defined using the Concise Data Definition Language (CDDL):

- * Concise Reference Integrity Manifests (CoRIM),
- * Concise Module Identifiers (CoMID), and
- * Concise Software Identifier (CoSWID).

CoRIM and CoMID tags are defined in this document, CoSWID tags are defined in [I-D.ietf-sacm-coswid]. CoRIM provide a wrapper structure, in which CoMID tags, CoSWID tags, as well as corresponding metadata can be bundled and signed as a whole. CoMID tags represent hardware components and provide a counterpart to CoSWID tags, which represent software components.

In accordance to [RFC4949], software components that are stored in hardware modules are referred to as firmware. While firmware can be represented as a software component, it is also very hardware-specific and often resides directly on block devices instead of a file system. In this specification, firmware and their Reference Values are represented via CoMID tags. Reference Values for any other software components stored on a file system are represented via CoSWID tags.

In addition to CoRIM - and respective CoMID tags - this specification defines a Concise Manifest Revocation that represents a list of reference to CoRIM that are actively marked as invalid before their expiration time.

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.

2. Concise Reference Integrity Manifests

This section specifies the Concise RIM (CoRIM) format, the Concise MID format (CoMID), and the extension to the CoSWID specification that augments CoSWID tags to express specific relationships to CoMID tags.

While each specification defines its own start rule, only CoMID and CoSWID are stand-alone specifications. The CoRIM specification - as the bundling format - has a dependency on CoMID and CoSWID and is not a stand-alone specification.

While stand-alone CoSWID tags may be signed [I-D.ietf-sacm-coswid], CoMID tags are not intended to be stand-alone and are always part of a CoRIM that must be signed. [I-D.ietf-sacm-coswid] specifies the use of COSE [RFC7231] for signing. This specification defines how to generate signed CoRIM tags with COSE to enable proof of authenticity and temper-evidence.

This document uses the Concise Data Definition Language (CDDL [RFC8610]) to define the data structure of CoRIM and CoMID tags, as well as the extensions to CoSWID. The CDDL definitions provided define nested containers. Typically, the CDDL types used for nested containers are maps. Every key used in the maps is a named type that is associated with an corresponding uint via a block of rules appended at the end of the CDDL definition.

Every set of uint keys that is used in the context of the "collision domain" of map is intended to be collision-free (each key is intended to be unique in the scope of a map, not a multimap). To accomplish that, for each map there is an IANA registry for the map members of maps.

2.1. Typographical Conventions

Type names in the following CDDL definitions follow the naming convention illustrated in table Table 1.

type trait	example	typo 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 Convention

2.2. Prefixes and Namespaces

The semantics of the information elements (attributes) defined for CoRIM, CoMID tags, and CoSWID tags are sometimes very similar, but often do not share the same scope or are actually quite different. In order to not overload the already existing semantics of the software-centric IANA registries of CoSWID tags with, for example, hardware-centric semantics of CoMID tags, new type names are introduced. For example: both CoSWID tags and CoMID tags define a tag-id. As CoSWID already specifies "tag-id", the tag-id in CoMID tags is prefixed with "comid." to disambiguate the context, resulting in "comid.tag-id". This prefixing provides a well-defined scope for the use of the types defined in this document and guarantees interoperability (no type name collisions) with the CoSWID CDDL definition. Effectively, the prefixes used in this specification enable simple hierarchical namespaces. The prefixing introduced is also based on the anticipated namespace features for CDDL.

2.3. Extensibility

Both the CoRIM and the CoMID tag specification include extension points using CDDL sockets (see [RFC8610] Section 3.9). The use of CDDL sockets allows for well-formed extensions to be defined in supplementary CDDL definitions that support additional uses of CoRIM and CoMID tags.

There are two types of extensibility supported via the extension points defined in this document. Both types allow for the addition of keys in the scope of a map.

Custom Keys: The CDDL definition allows for the use of negative integers as keys. These keys cannot take on a well-defined global semantic. They can take on custom-defined semantics in a limited or local scope, e.g. vendor-defined scope.

Registered Keys: Additional keys can be registered at IANA via separate specifications.

Both types of extensibility also allow for the definition of new nested maps that again can include additional defined keys.

2.4. Concise RIM Extension Points

The following CDDL sockets (extension points) are defined in the CoRIM specification, which allow the addition of new information structures to their respective CDDL groups.

Map Name	CDDL Socket	Defined in
corim-entity-map	\$\$corim-entity-map-extension	Section 3.1.2
unsigned-corim-map	\$\$unsigned-corim-map-extension	Section 3.2
concise-mid-tag	\$\$comid-extension	Section 3.3
tag-identity-map	\$\$tag-identity-map-extension	Section 3.4
entity-map	\$\$entity-map-extension	Section 3.5
triples-map	\$\$triples-map-extension	Section 3.7
measurement-values-map	\$\$measurement-values-map-extension	Section 3.10

Table 2: CoMID CDDL Group Extension Points

2.5. CDDL Generic Types

The CDDL definitions for CoRIM and CoMID tags use the two following generic types.

2.5.1. 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 optional members.

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

2.5.2. One-Or-More

The one-or-more generic type allows to omit an encapsulating array, if only one member would be present.

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


3. Concise RIM Data Definition

A CoRIM is a bundle of CoMID tags and/or CoSWID tags that can reference each other and that includes additional metadata about that bundle.

The root of the CDDL specification provided for CoRIM is the rule "corim" :

```
start = corim
```

3.1. The signed-corim Container

A CoRIM is signed using [RFC7231]. The additional CoRIM-specific COSE header member label corim-meta is defined as well as the corresponding type corim-meta-map as its value. This rule and its constraints MUST be followed when generating or validating a signed CoMID tag.

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

```
protected-signed-corim-header-map = {  
  corim.alg-id => int  
  corim.content-type => "application/rim+cbor"  
  corim.issuer-key-id => bstr  
  corim.meta => corim-meta-map  
  * cose-label => cose-values  
}
```

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

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

3.1.1. The corim-meta-map Container

This map contains the two additionally defined attributes "corim-entity-map" and "validity-map" that are used to annotate a CoRIM with metadata.

```
corim-meta-map = {  
  corim.signer => one-or-more<corim-entity-map>  
  ? corim.validity => validity-map  
}
```

corim.signer: One or more entities that created and/or signed the issued CoRIM.

corim.validity: A time period defining the validity span of a CoRIM.

3.1.2. The corim-entity-map Container

This map is used to identify the signer of a CoRIM via a dedicated entity name, a corresponding role and an optional identifying URI.

```
corim-entity-map = {  
  corim.entity-name => $entity-name-type-choice  
  ? corim.reg-id => uri  
  corim.role => $corim-role-type-choice  
  * $$corim-entity-map-extension  
}
```

\$corim-role-type-choice /= corim.manifest-creator

\$corim-role-type-choice /= corim.manifest-signer

corim.entity-name: The name of the organization that takes on the role expressed in "corim.role"

corim.reg-id: The registration identifier of the organization that has authority over the namespace for "corim.entity-name".

corim.role: The role type that is associated with the entity, e.g. the creator of the CoRIM or the signer of the CoRIM.

\$\$corim-entity-map-extension: This CDDL socket is used to add new information elements to the corim-entity-map container. See FIXME.

3.1.3. The validity-map Container

The members of this map indicate the life-span or period of validity of a CoRIM that is baked into the protected header at the time of signing.

```
validity-map = {  
  ? corim.not-before => time  
  corim.not-after => time  
}
```

corim.not-before: The timestamp indicating the CoRIM's begin of its validity period.

corim.not-after: The timestamp indicating the CoRIM's end of its validity period.

3.2. The unsigned-corim-map Container

This map contains the payload of the COSE envelope that is used to sign the CoRIM. This rule and its constraints MUST be followed when generating or validating an unsigned Concise RIM.

```
unsigned-corim-map = {  
  corim.id => $corim-id-type-choice  
  corim.tags => one-or-more<$concise-tag-type-choice>  
  ? corim.dependent-rims => one-or-more<corim-locator-map>  
  * $$unsigned-corim-map-extension  
}  
  
$corim-id-type-choice /= tstr  
$corim-id-type-choice /= uuid-type  
  
$concise-tag-type-choice /= #6.TBD-SWID(bytes .cbor concise-swid-tag)  
$concise-tag-type-choice /= #6.TBD-CoMID(bytes .cbor concise-mid-tag)  
  
corim.id: Typically a UUID or a text string that MUST uniquely  
  identify a CoRIM in a given scope.  
  
corim.tags: A collection of one or more CoMID tags and/or CoSWID  
  tags.  
  
corim.dependent-rims: One or more services available via the  
  Internet that can supply additional, possibly dependent manifests  
  (or other associated resources).  
  
$$unsigned-corim-map-extension: This CDDL socket is used to add new  
  information elements to the unsigned-corim-map container. See  
  FIXME.
```

3.2.1. The corim-locator-map Container

This map is used to locate and verify the integrity of resources provided by external services, e.g. the CoRIM provider.

```
corim-locator-map = {  
  corim.href => uri  
  ? corim.thumbprint => hash-entry  
}
```

`corim.href`: A pointer to a services that supplies dependent files or records.

`corim.thumbprint`: A digest of the reference resource.

3.3. The concise-mid-tag Container

The CDDL specification for the root concise-mid-tag map is as follows. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
concise-mid-tag = {  
  ? comid.language => language-type  
  comid.tag-identity => tag-identity-map  
  ? comid.entity => one-or-more<entity-map>  
  ? comid.linked-tags => one-or-more<linked-tag-map>  
  comid.triples => triples-map  
  * $$concise-mid-tag-extension  
}
```

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

`comid.language`: A textual language tag that conforms with the IANA Language Subtag Registry [IANA.language-subtag-registry].

`comid.tag-identity`: A composite identifier containing identifying attributes that enable global unique identification of a CoMID tag across versions.

`comid.entity`: A list of entities that contributed to the CoMID tag.

`comid.linked-tags`: A list of tags that are linked to this CoMID tag.

`comid.triples`: A set of relationships in the form of triples, representing a graph-like and semantic reference structure between tags.

`$$comid-mid-tag-extension`: This CDDL socket is used to add new information elements to the concise-mid-tag root container. See FIXME.

3.4. The tag-identity-map Container

The CDDL specification for the tag-identity-map includes all identifying attributes that enable a consumer of information to anticipate required capabilities to process the corresponding tag that map is included in. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
tag-identity-map = {  
  comid.tag-id => $tag-id-type-choice  
  comid.tag-version => tag-version-type  
  * $$tag-identity-map-extension  
}
```

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

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

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

comid.tag-id: An identifier for a CoMID that MUST be globally unique.

comid.tag-version: An unsigned integer used as a version identifier.

\$\$tag-identity-map-extension: This CDDL socket is used to add new information elements to the tag-identity-map container. See FIXME.

3.5. The entity-map Container

This Container provides qualifying attributes that provide more context information describing the module as well its origin and purpose. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
entity-map = {  
  comid.entity-name => $entity-name-type-choice  
  ? comid.reg-id => uri  
  comid.role => one-or-more<$comid-role-type-choice>  
  * $$entity-map-extension  
}
```

```
$comid-role-type-choice /= comid.tag-creator  
$comid-role-type-choice /= comid.creator  
$comid-role-type-choice /= comid.maintainer
```

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

comid.entity-name: The name of an organization that performs the roles as indicated by comid.role.

comid.reg-id: The registration identifier of the organization that

has authority over the namespace for "comid.entity-name".

comid.role: The list of roles a CoMID entity is associated with. The entity that generates the concise-mid-tag SHOULD include a \$comid-role-type-choice value of comid.tag-creator.

\$entity-map-extension: This CDDL socket is used to add new information elements to the entity-map container. See FIXME.

3.6. The linked-tag-map Container

A list of tags that are linked to this CoMID tag.

```
linked-tag-map = {  
  comid.linked-tag-id => $tag-id-type-choice  
  comid.tag-rel => $tag-rel-type-choice  
}
```

```
$tag-rel-type-choice /= comid.supplements  
$tag-rel-type-choice /= comid.replaces
```

The following describes each member of the linked-tag-map container.

comid.linked-tag-id: The tag-id of the linked tag. A linked tag MAY be a CoMID tag or a CoSWID tag.

comid.tag-rel: The relationship type with the linked tag. The relationship type MAY be "supplements" or "replaces", as well as other types well-defined by additional specifications.

3.7. The triples-map Container

A set of directed properties that associate sets of data to provide reference values, endorsed values, verification key material or identifying key material for a specific hardware module that is a component of a composite device. The map provides the core element of CoMID tags that associate remote attestation relevant data with a distinct hardware component that runs an execution environment (a module that is either a Target Environment and/or an Attesting Environment). This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
triples-map = non-empty<{  
  ? comid.reference-triples => one-or-more<reference-triple-record>  
  ? comid.endorsed-triples => one-or-more<endorsed-triple-record>  
  ? comid.attest-key-triples => one-or-more<attest-key-triple-record>  
  ? comid.identity-triples => one-or-more<identity-triple-record>  
  * $$triples-map-extension  
>
```

The following describes each member of the triple-map container.

comid.reference-triples: A directed property that associates reference measurements with a module that is a Target Environment.

comid.endorsed-triples: A directed property that associates endorsed measurements with a module that is a Target Environment or Attesting Environment.

comid.attest-key-triples: A directed property that associates key material used to verify evidence generated from a module that is an attesting environment.

comid.identity-triples: A directed property that associates key material used to identify a module instance or a module class that is an identifying part of a device(-set).

\$\$triples-map-extension: This CDDL socket is used to add new information elements to the triples-map container. See FIXME.

3.8. The environment-map Container

This map represents the module(s) that a triple-map can point directed properties (relationships) from in order to associate them with external information for remote attestation, such as reference values, endorsement and endorsed values, verification key material for evidence, or identifying key material for module (re-)identification. This map can identify a single module instance via "comid.instance" or groups of modules via "comid.group". Referencing classes of modules requires the use of the more complex "class-map" container. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
environment-map = non-empty<{  
  ? comid.class => class-map  
  ? comid.instance => $instance-id-type-choice  
  ? comid.group => $group-id-type-choice  

```

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

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

The following describes each member of the environment-map container.

comid.class: A composite identifier for classes of environments/modules.

comid.instance: An identifier for distinct instances of environments/modules that is either a UEID or a UUID.

comid.group: An identifier for a group of environments/modules that is a UUID.

3.9. The class-map Container

This map enables a composite identifier intended to uniquely identify modules that are of a distinct class of devices. Effectively, all provided members in combination are a composite module class identifier. This rule and its constraints MUST be followed when generating or validating a CoMID tag. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
class-map = non-empty<{  
  ? comid.class-id => $class-id-type-choice  
  ? comid.vendor => tstr  
  ? comid.model => tstr  
  ? comid.layer => uint  
  ? comid.index => uint  

```

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

The following describes each member of the class-map container.

3.10. The measurement-map and measurement-values-map Containers

One of the targets (range) that a triple-map can point to in order to associate it with a module (domain) is the measurement-map. This map is used to provide reference measurements values that can be compared with Evidence Claim values or Endorsements and endorsed values from other sources than the corresponding CoRIM. "measurement-map" comes with a measurement key that identifies the measured element with via a OID reference or a UUID. "measurement-values-map" contains the actual measurements associated with the module(s). Byte strings with corresponding bit masks that highlights which bits in the byte string are used as reference measurements or endorsement are located in "raw-value-group". The members of "measurement-values-map" provide well-defined and well-scoped semantics for reference measurement or endorsements with respect to a given module instance, class, or group. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
measurement-map = {
  ? comid.mkey => $measured-element-type-choice
  comid.mval => measurement-values-map
}
```

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

```
measurement-values-map = non-empty<{
  ? comid.ver => version-map
  ? comid.svn => svn-type-choice
  ? comid.digests => digests-type
  ? comid.flags => flags-type
  ? raw-value-group
  ? comid.mac-addr => mac-addr-type-choice
  ? comid.ip-addr => ip-addr-type-choice
  ? comid.serial-number => serial-number-type
  ? comid.ueid => ueid-type
  ? comid.uuid => uuid-type
  * $$measurement-values-map-extension
}>
```

```
flags-type = bytes .bits operational-flags
```

```
operational-flags = &(
  not-configured: 0
  not-secure: 1
  recovery: 2
  debug: 3
)
```

```
serial-number-type = text
```

```
digests-type = one-or-more<hash-entry>
```

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

comid.mkey: An identifier for the set of measurements expressed in measurement-values-map that is either an OID or a UUID.

comid.ver: A version number measurement.

comid.svn: A security related version number measurement.

comid.digests: A digest (typically a hash value) measurement.

comid.flags: Measurements that reflect operational modes that are

made permanent at manufacturing time such that they are not expected to change during normal operation of the Attester.

raw-value-group: A measurement in the form of a byte string that can come with a corresponding bit mask defining the relevance of each bit in the byte string as a measurement.

comid.mac-addr: An EUI-48 or EUI-64 MAC address measurement.

comid.ip-addr: An Ipv4 or Ipv6 address measurement.

comid.serial-number: A measurement of a serial number in text.

comid.ueid: A measurement of a Unique Enough Identifier (UEID).

comid.uuid: A measurement of a Universally Unique Identifier (UUID).

\$\$measurement-values-map-extension: This CDDL socket is used to add new information elements to the measurement-values-map container. See FIXME.

3.10.1. The version-map Container

This map expresses reference values about version information.

```
version-map = {  
  comid.version => version-type  
  ? comid.version-scheme => $version-scheme  
}
```

```
version-type = text .default '0.0.0'
```

The following describes each member of the version-map container.

comid.version: The version in the form of a text string.

comid-version-scheme: The version-scheme of the text string value as defined in [I-D.ietf-sacm-coswid]

3.10.2. The svn-type-choice Enumeration

This choice defines the CBOR tagged Security Version Numbers (SVN) that can be used as reference values for Evidence and Endorsements.

```
svn = int
min-svn = int
tagged-svn = #6.TBD-SVN(svn)
tagged-min-svn = #6.TBD-minSVN(min-svn)
svn-type-choice = tagged-svn / tagged-min-svn
```

The following describes the types in the svn-type-choice enumeration.

tagged-svn: A specific SVN.

tagged-min-svn: A lower bound for allowed SVN.

3.10.3. The raw-value-group Container

FIXME This group can express a single raw byte value and can come with an optional bit mask that defines which bits in the byte string is used as a reference value, by setting corresponding position in the bit mask to 1.

```
raw-value-group = (
  comid.raw-value => raw-value-type
  ? comid.raw-value-mask => raw-value-mask-type
)
```

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

The following describes the types in the raw-value-group Container.

comid.raw-value: FIXME Bit positions in raw-value-type that correspond to bit positions in raw-value-mask-type.

comid.raw-value-mask: A raw-value-mask-type bit corresponding to a bit in raw-value-type MUST be 1 to evaluate the corresponding raw-value-type bit.

3.10.4. The ip-addr-type-choice Enumeration

This type choice expresses IP addresses as reference values.

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

3.10.5. The mac-addr-type-choice Enumeration

This type choice expresses MAC addresses as reference values.

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

3.11. The verification-key-map Container

One of the targets (range) that a triple-map can point to in order to associate it with a module (domain). This map is used to provide the key material for evidence verification (effectively signature checking or a lightweight proof-of-possession of private signing key material) or for identity assertion/check (where a proof-of-possession implies a certain device identity). In support of informed trust decisions, an optional trust anchor in the form a PKIX certification path that is associated with the provided key material can be included. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
verification-key-map = {
  comid.key => pkix-base64-key-type
  ? comid.keychain => [ + pkix-base64-cert-type ]
}
```

```
pkix-base64-key-type = tstr
pkix-base64-cert-type = tstr
```

The following describes each member of the verification-key-map container.

comid.key: Verification key material in DER format base64 encoded. Typically, but not necessarily, a public key.

comid.keychain: One or more base64 encoded PKIX certificates. The certificate containing the public key in comid.key MUST be the first certificate. Additional certificates MAY follow. Each subsequent certificate SHOULD certify the previous certificate.

4. Full CDDL Definition

This section aggregates the CDDL definitions specified in this document in a full CDDL definitions including:

- * the COSE envelope for CoRIM: signed-corim
- * the CoRIM document: unsigned-corim
- * the CoMID document: concise-mid-tag

Not included in the full CDDL definition are CDDL dependencies to CoSWID. The following CDDL definitions can be found in [I-D.ietf-sacm-coswid]:

* the COSE envelope for CoRIM: signed-coswid

* the CoSWID document: concise-swid-tag

<CODE BEGINS>

corim = #6.500(\$concise-reference-integrity-manifest-type-choice)

\$concise-reference-integrity-manifest-type-choice /= #6.501(unsigned-corim-map
)

\$concise-reference-integrity-manifest-type-choice /= #6.502(signed-corim)

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

protected-signed-corim-header-map = {
 corim.alg-id => int
 corim.content-type => "application/rim+cbor"
 corim.issuer-key-id => bstr
 corim.meta => corim-meta-map
 * cose-label => cose-values
}

corim-meta-map = {
 corim.signer => [+ corim-entity-map]
 ? corim.validity => validity-map
}

corim-entity-map = {
 corim.entity-name => \$entity-name-type-choice
 ? corim.reg-id => uri
 corim.role => \$corim-role-type-choice
 * \$\$corim-entity-map-extension
}

\$corim-role-type-choice /= corim.manifest-creator
\$corim-role-type-choice /= corim.manifest-signer

validity-map = {
 ? corim.not-before => time
 corim.not-after => time
}

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

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

unsigned-corim-map = {
  corim.id => $corim-id-type-choice
  corim.tags => [ + $concise-tag-type-choice ]
  ? corim.dependent-rims => [ + corim-locator-map ]
  ? corim.profile => [ + profile-type-choice ]
  * $$unsigned-corim-map-extension
}

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

corim-locator-map = {
  corim.href => uri
  ? corim.thumbprint => hash-entry
}

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

concise-mid-tag = {
  ? comid.language => language-type
  comid.tag-identity => tag-identity-map
  ? comid.entity => [ + entity-map ]
  ? comid.linked-tags => [ + linked-tag-map ]
  comid.triples => triples-map
  * $$concise-mid-tag-extension
}

language-type = text

tag-identity-map = {
  comid.tag-id => $tag-id-type-choice
  ? comid.tag-version => tag-version-type
}

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

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

```
entity-map = {
  comid.entity-name => $entity-name-type-choice
  ? comid.reg-id => uri
  comid.role => [ + $comid-role-type-choice ]
  * $$entity-map-extension
}

$comid-role-type-choice /= comid.tag-creator
$comid-role-type-choice /= comid.creator
$comid-role-type-choice /= comid.maintainer

linked-tag-map = {
  comid.linked-tag-id => $tag-id-type-choice
  comid.tag-rel => $tag-rel-type-choice
}

$tag-rel-type-choice /= comid.supplements
$tag-rel-type-choice /= comid.replaces

triples-map = non-empty<{
  ? comid.reference-triples => [ + reference-triple-record ]
  ? comid.endorsed-triples => [ + endorsed-triple-record ]
  ? comid.attest-key-triples => [ + attest-key-triple-record ]
  ? comid.identity-triples => [ + identity-triple-record ]
  * $$triples-map-extension
}>

reference-triple-record = [
  environment-map ; target environment
  [ + measurement-map ] ; reference measurements
]

endorsed-triple-record = [
  environment-map ; (target or attesting) environment
  [ + measurement-map ] ; endorsed measurements
]

attest-key-triple-record = [
  environment-map ; attesting environment
  [ + verification-key-map ] ; attestation verification key(s)
]

identity-triple-record = [
  environment-map ; device identifier (instance or class)
  [ + verification-key-map ] ; DevID, or semantically equivalent
]

pkix-base64-key-type = tstr
```



```
pkix-base64-cert-type = tstr

verification-key-map = {
  ; Verification key in DER format base64-encoded.
  ; Typically, but not necessarily a public key.
  comid.key => pkix-base64-key-type
  ; Optional X.509 certificate chain corresponding to the public key
  ; in comid.key, encoded as an array of one or more base64-encoded
  ; DER PKIX certificates. The certificate containing the public key
  ; in comid.key MUST be the first certificate. This MAY be followed
  ; by additional certificates, with each subsequent certificate
  ; being the one used to certify the previous one.
  ? comid.keychain => [ + pkix-base64-cert-type ]
}

environment-map = non-empty<{
  ? comid.class => class-map
  ? comid.instance => $instance-id-type-choice
  ? comid.group => $group-id-type-choice
}>

class-map = non-empty<{
  ? comid.class-id => $class-id-type-choice
  ? comid.vendor => tstr
  ? comid.model => tstr
  ? comid.layer => uint
  ? comid.index => uint
}>

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

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

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

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

tagged-uuid-type = #6.37(uuid-type)

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

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

```
measurement-map = {
  ? comid.mkey => $measured-element-type-choice
  comid.mval => measurement-values-map
}

measurement-values-map = non-empty<{
  ? comid.ver => version-map
  ? comid.svn => svn-type-choice
  ? comid.digests => digests-type
  ? comid.flags => flags-type
  ? raw-value-group
  ? comid.mac-addr => mac-addr-type-choice
  ? comid.ip-addr => ip-addr-type-choice
  ? comid.serial-number => serial-number-type
  ? comid.ueid => ueid-type
  ? comid.uuid => uuid-type
  * $$measurement-values-map-extension
}>

version-map = {
  comid.version => version-type
  ? comid.version-scheme => $version-scheme
}
version-type = text .default '0.0.0'

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

flags-type = bytes .bits operational-flags

operational-flags = &(
  not-configured: 0
  not-secure: 1
  recovery: 2
  debug: 3
)

raw-value-group = (
  comid.raw-value => raw-value-type
  ? comid.raw-value-mask => raw-value-mask-type
)

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

```
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

serial-number-type = text

digests-type = [ + hash-entry ]

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

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

any-uri = uri
label = text / int

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

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

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

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

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

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

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

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

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

$rel /= ancestor
$rel /= component
```

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

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

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

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

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

file-entry = {
```

```
filesystem-item,  
? size => uint,  
? file-version => text,  
? hash => hash-entry,  
* $$file-extension,  
global-attributes,  
}  
  
directory-entry = {  
  filesystem-item,  
  ? path-elements => { path-elements-group },  
  * $$directory-extension,  
  global-attributes,  
}  
  
process-entry = {  
  process-name => text,  
  ? pid => integer,  
  * $$process-extension,  
  global-attributes,  
}  
  
resource-entry = {  
  type => text,  
  * $$resource-extension,  
  global-attributes,  
}  
  
filesystem-item = (  
  ? key => bool,  
  ? location => text,  
  fs-name => text,  
  ? root => text,  
)  
  
payload-entry = {  
  resource-collection,  
  * $$payload-extension,  
  global-attributes,  
}  
  
evidence-entry = {  
  resource-collection,  
  ? date => integer-time,  
  ? device-id => text,  
  * $$evidence-extension,  
  global-attributes,  
}
```

```
integer-time = #6.1(int)

tag-id = 0
software-name = 1
entity = 2
evidence = 3
link = 4
software-meta = 5
payload = 6
hash = 7
corpus = 8
patch = 9
media = 10
supplemental = 11
tag-version = 12
software-version = 13
version-scheme = 14
lang = 15
directory = 16
file = 17
process = 18
resource = 19
size = 20
file-version = 21
key = 22
location = 23
fs-name = 24
root = 25
path-elements = 26
process-name = 27
pid = 28
type = 29
entity-name = 31
reg-id = 32
role = 33
thumbprint = 34
date = 35
device-id = 36
artifact = 37
href = 38
ownership = 39
rel = 40
media-type = 41
use = 42
activation-status = 43
channel-type = 44
colloquial-version = 45
description = 46
```

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

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

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

shared=1
private=2
abandon=3

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

optional=1
required=2
recommended=3

comid.language = 0
comid.tag-identity = 1
comid.entity = 2
comid.linked-tags = 3


```
comid.triples = 4

comid.tag-id = 0
comid.tag-version = 1

comid.entity-name = 0
comid.reg-id = 1
comid.role = 2

comid.linked-tag-id = 0
comid.tag-rel = 1

comid.reference-triples = 0
comid.endorsed-triples = 1
comid.identity-triples = 2
comid.attest-key-triples = 3

comid.class = 0
comid.instance = 1
comid.group = 2

comid.class-id = 0
comid.vendor = 1
comid.model = 2
comid.layer = 3
comid.index = 4

comid.mkey = 0
comid.mval = 1

comid.ver = 0
comid.svn = 1
comid.digests = 2
comid.flags = 3
comid.raw-value = 4
comid.raw-value-mask = 5
comid.mac-addr = 6
comid.ip-addr = 7
comid.serial-number = 8
comid.ueid = 9
comid.uuid = 10

comid.key = 0
comid.keychain = 1

comid.version = 0
comid.version-scheme = 1
```

```
comid.supplements = 0

comid.replaces = 1

comid.tag-creator = 0
comid.creator = 1
comid.maintainer = 2


corim.id = 0
corim.tags = 1
corim.dependent-rims = 2
corim.profile = 3


corim.href = 0
corim.thumbprint = 1


corim.alg-id = 1
corim.content-type = 3
corim.issuer-key-id = 4
corim.meta = 8


corim.not-before = 0
corim.not-after = 1


corim.signer = 0
corim.validity = 1


corim.entity-name = 0
corim.reg-id = 1
corim.role = 2


corim.manifest-creator = 1

corim.manifest-signer = 2


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


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

$entity-name-type-choice /= text

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

uuid-type = bytes .size 16

<CODE ENDS>

5. Privacy Considerations

Privacy Considerations

6. Security Considerations

Security Considerations

7. IANA Considerations

This document has a number of IANA considerations, as described in the following subsections. In summary, 6 new registries are established with this request, with initial entries provided for each registry. New values for 5 other registries are also requested.

7.1. COSE Header Parameters Registry

The 'corim metadata' parameter has been added to the "COSE Header Parameters" registry:

- * Name: 'corim metadata'
- * Label: 11
- * Value: corim-meta-map
- * Description: Provides a map of additional metadata for a CoRIM payload composed of (1) one or more entities that created or signed the corresponding CoRIM and (2) its period of validity
- * Reference: 'corim-meta-map' in {model-corim-meta-map} of this document

7.2. CoRIM Map Items Registry

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

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

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

Table 3: CoRIM Map Items
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	corim.id	RFC-AAAA
1	corim.tags	RFC-AAAA
2	corim.dependent-rims	RFC-AAAA
3-255	Unassigned	

Table 4: CoRIM Map Items Initial Registrations

7.3. CoRIM Entity-Map Items Registry

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

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

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

Table 5: CoRIM Entity Map Items
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	corim.entity-name	RFC-AAAA
1	corim.reg-id	RFC-AAAA
2	corim.role	RFC-AAAA
3-255	Unassigned	

Table 6: CoRIM Entity Map Items Initial
Registrations

7.4. CoRIM Entity-Types Registry

This document defines a new registry titled "CoRIM Entity Types". The registry maintains well-defined integer values as choices for '§entity-name-type-choice' CBOR uints.

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

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

Table 7: CoRIM Entity Types
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	corim.manifest-creator	RFC-AAAA
1	corim.manifest-signer	RFC-AAAA
2-255	Unassigned	

Table 8: CoRIM Entity Types Initial Registrations

7.5. CoMID Map Items Registry

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

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

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

Table 9: CoMID Map Items
Registration Procedures

All negative values are reserved for Private Use.

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

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

Table 10: CoMID Map Items Initial Registrations

7.6. CoMID Entity-Map Items Registry

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

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

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

Table 11: CoMID Entity Map Items Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.entity-name	RFC-AAAA
1	comid.reg-id	RFC-AAAA
2	comid.role	RFC-AAAA
3-255	Unassigned	

Table 12: CoMID Entity Map Items Initial Registrations

7.7. CoMID Triples-Map Items Registry

This document defines a new registry titled "CoMID Triples Map". The registry uses integer values as index values for items in 'comid-triples-map' CBOR maps.

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

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

Table 13: CoMID triples Map Items Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.reference-triples	RFC-AAAA
1	comid.endorsed-triples	RFC-AAAA
2	comid.identity-triples	RFC-AAAA
3	comid.attest-key-triples	RFC-AAAA
4-255	Unassigned	

Table 14: CoMID Triples Map Items Initial Registrations

7.8. CoMID Measurement-Values-Map Items Registry

This document defines a new registry titled "CoMID Measurement-Values Map". The registry uses integer values as index values for items in 'comid-measurement-values-map' CBOR maps.

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

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

Table 15: CoMID Measurement-Values Map Items Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.ver	RFC-AAAA
1	comid.svn	RFC-AAAA
2	comid.digests	RFC-AAAA
3	comid.flags	RFC-AAAA
4	comid.raw-value	RFC-AAAA
5	comid.raw-value-mask	RFC-AAAA
6	comid.mac-addr	RFC-AAAA
7	comid.ip-addr	RFC-AAAA
8	comid.serial-number	RFC-AAAA
9	comid.ueid	RFC-AAAA
10	comid.uuid	RFC-AAAA
11-255	Unassigned	

Table 16: CoMID Measurement-Values Map Items
Initial Registrations

7.9. CoMID Tag-Relationship-Types Registry

This document defines a new registry titled "CoMID Tag-Relationship Types". The registry maintains well-defined integer values as choices for '\$tag-rel-type-choice' CBOR uints.

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

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

Table 17: CoMID Tag-Relationship
Types Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.supplements	RFC-AAAA
1	comid.replaces	RFC-AAAA
2-255	Unassigned	

Table 18: CoMID Tag-Relationship Types
Initial Registrations

7.10. CoMID Role-Types Registry

This document defines a new registry titled "CoMID Role Types". The registry maintains well-defined integer values as choices for '\$comid-role-type-choice' CBOR uints.

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

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

Table 19: CoMID Role Types
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.tag-creator	RFC-AAAA
1	comid.creator	RFC-AAAA
2	comid.maintainer	RFC-AAAA
3-255	Unassigned	

Table 20: CoMID Role Types Initial
Registrations

7.11. rim+cbor Media Type Registration

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

Type name: application

Subtype name: rim+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC8949]. See RFC-AAAA for details.

Security considerations: See Section 6 of RFC-AAAA.

Interoperability considerations: Applications MAY ignore any key value pairs that they do not understand. This allows backwards compatible extensions to this specification.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by remote attestation procedures, supply chain integrity management systems, vulnerability assessment systems, and in applications that rely on trustworthy endorsements and reference values describing the intended operational state of a system.

Fragment identifier considerations: Fragment identification for application/rim+cbor is supported by using fragment identifiers as specified by Section 9.5 of [RFC8949].

Additional information:

Magic number(s): first five bytes in hex: 43 4f 52 49 4d

File extension(s): corim

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.corim conforms to public.data

Person & email address to contact for further information: Henk Birkholz <henk.birkholz@sit.fraunhofer.de>

Intended usage: COMMON

Restrictions on usage: None

Author: Henk Birkholz <henk.birkholz@sit.fraunhofer.de>

Change controller: IESG

7.12. CoAP Content-Format Registration

IANA is requested to assign a CoAP Content-Format ID for the CoRIM media type in the "CoAP Content-Formats" sub-registry, from the "IETF Review or IESG Approval" space (256..999), within the "CoRE Parameters" registry [RFC7252] [IANA.core-parameters]:

Media type	Encoding	ID	Reference
application/rim+cbor	-	TBD1	RFC-AAAA

Table 21: CoAP Content-Format IDs

7.13. 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
500	tagged array or tagged map	Concise Reference Integrity Manifest (CoRIM) [RFC-AAAA]
501	map	unsigned CoRIM [RFC-AAAA]
502	array	signed CoRIM [RFC-AAAA]
505	bstr	byte string with CBOR-encoded Concise SWID tag [RFC-AAAA]
506	bstr	byte string with CBOR-encoded Concise MID tag [RFC-AAAA]

Table 22: CoRIM CBOR Tags

7.14. CoMID 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
550	bstr	UEID with max size of 33 bytes [RFC-AAAA]
551	int	Security Version Number [RFC-AAAA]
552	int	lower bound of allowed Security Version Number [RFC-AAAA]

Table 23: CoMID CBOR Tags

8. References

8.1. 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-12, 23 April 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-12>>.

[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-18, 12 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-sacm-coswid-18>>.

[IANA.cbor-tags]

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

[IANA.core-parameters]

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

[IANA.language-subtag-registry]

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

[IANA.media-types]

IANA, "Media Types", <<http://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>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/rfc/rfc7231>>.
- [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/rfc/rfc7252>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.
- [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>>.
- [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

- [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>>.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Thomas Fossati
Arm Limited
United Kingdom

Email: Thomas.Fossati@arm.com

Yogesh Deshpande
Arm Limited
United Kingdom

Email: yogesh.deshpande@arm.com

Ned Smith
Intel Corporation
United States of America

Email: ned.smith@intel.com

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 30 July 2022

H. Birkholz
Fraunhofer SIT
T. Fossati
Y. Deshpande
Arm Limited
N. Smith
Intel
W. Pan
Huawei Technologies
26 January 2022

Concise Reference Integrity Manifest
draft-birkholz-rats-corim-02

Abstract

Remote Attestation Procedures (RATS) enable Relying Parties to put trust in the trustworthiness of a remote Attester and therefore to decide if to engage in secure interactions with it - or not. Evidence about trustworthiness can be rather complex, voluminous or Attester-specific. As it is deemed unrealistic that every Relying Party is capable of the appraisal of Evidence, that burden is taken on by a Verifier. In order to conduct Evidence appraisal procedures, a Verifier requires not only fresh Evidence from an Attester, but also trusted Endorsements and Reference Values from Endorsers, such as manufacturers, distributors, or 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.

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 30 July 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	3
1.1. Requirements Notation	4
2. Concise Reference Integrity Manifests	4
2.1. Typographical Conventions	5
2.2. Prefixes and Namespaces	6
2.3. Extensibility	7
2.4. Concise RIM Extension Points	7
2.5. CDDL Generic Types	8
2.5.1. Non-Empty	8
2.5.2. One-Or-More	8
3. Concise RIM Data Definition	9
3.1. The signed-corim Container	9
3.1.1. The corim-meta-map Container	9
3.1.2. The corim-signer-map Container	10
3.1.3. The validity-map Container	10
3.2. The corim-map Container	11
3.2.1. The corim-entity-map Container	12
3.2.2. The corim-locator-map Container	12
3.3. The concise-mid-tag Container	13
3.4. The tag-identity-map Container	13
3.5. The entity-map Container	14
3.6. The linked-tag-map Container	15
3.7. The triples-map Container	15
3.8. The environment-map Container	16
3.9. The class-map Container	17
3.10. The measurement-map and measurement-values-map Containers	17
3.10.1. The version-map Container	19
3.10.2. The svn-type-choice Enumeration	20
3.10.3. The raw-value-group Container	20

3.10.4. The ip-addr-type-choice Enumeration	21
3.10.5. The mac-addr-type-choice Enumeration	21
3.11. The verification-key-map Container	21
4. Full CDDL Definition	22
5. Privacy Considerations	35
6. Security Considerations	35
7. IANA Considerations	35
7.1. COSE Header Parameters Registry	35
7.2. CoRIM Map Items Registry	36
7.3. CoRIM Entity-Map Items Registry	36
7.4. CoRIM Entity-Types Registry	37
7.5. CoMID Map Items Registry	38
7.6. CoMID Entity-Map Items Registry	39
7.7. CoMID Triples-Map Items Registry	40
7.8. CoMID Measurement-Values-Map Items Registry	41
7.9. CoMID Tag-Relationship-Types Registry	42
7.10. CoMID Role-Types Registry	43
7.11. rim+cbor Media Type Registration	44
7.12. CoAP Content-Format Registration	45
7.13. CoRIM CBOR Tag Registration	46
7.14. CoMID CBOR Tag Registration	46
8. References	47
8.1. Normative References	47
8.2. Informative References	48
Authors' Addresses	48

1. Introduction

The Remote Attestation Procedures (RATS) architecture [I-D.ietf-rats-architecture] describes appraisal procedures for attestation Evidence and Attestation Results. Appraisal procedures for Evidence are conducted by Verifiers and are intended to assess the trustworthiness of a remote peer. Appraisal procedures for Attestation Results are conducted by Relying Parties and are intended to operationalize the assessment about a remote peer and to act appropriately based on the assessment. In order to enable their intent, appraisal procedures consume Appraisal Policies, Reference Values, and Endorsements.

This documents specifies a binary encoding for Reference Values using the Concise Binary Object Representation (CBOR). The encoding is based on three parts that are defined using the Concise Data Definition Language (CDDL):

- * Concise Reference Integrity Manifests (CoRIM),
- * Concise Module Identifiers (CoMID), and

* Concise Software Identifier (CoSWID).

CoRIM and CoMID tags are defined in this document, CoSWID tags are defined in [I-D.ietf-sacm-coswid]. CoRIM provide a wrapper structure, in which CoMID tags, CoSWID tags, as well as corresponding metadata can be bundled and signed as a whole. CoMID tags represent hardware components and provide a counterpart to CoSWID tags, which represent software components.

In accordance to [RFC4949], software components that are stored in hardware modules are referred to as firmware. While firmware can be represented as a software component, it is also very hardware-specific and often resides directly on block devices instead of a file system. In this specification, firmware and their Reference Values are represented via CoMID tags. Reference Values for any other software components stored on a file system are represented via CoSWID tags.

In addition to CoRIM - and respective CoMID tags - this specification defines a Concise Manifest Revocation that represents a list of reference to CoRIM that are actively marked as invalid before their expiration time.

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.

2. Concise Reference Integrity Manifests

This section specifies the Concise RIM (CoRIM) format, the Concise MID format (CoMID), and the extension to the CoSWID specification that augments CoSWID tags to express specific relationships to CoMID tags.

While each specification defines its own start rule, only CoMID and CoSWID are stand-alone specifications. The CoRIM specification - as the bundling format - has a dependency on CoMID and CoSWID and is not a stand-alone specification.

While stand-alone CoSWID tags may be signed [I-D.ietf-sacm-coswid], CoMID tags are not intended to be stand-alone and are always part of a CoRIM that must be signed. [I-D.ietf-sacm-coswid] specifies the use of COSE [RFC7231] for signing. This specification defines how to generate signed CoRIM tags with COSE to enable proof of authenticity and temper-evidence.

This document uses the Concise Data Definition Language (CDDL [RFC8610]) to define the data structure of CoRIM and CoMID tags, as well as the extensions to CoSWID. The CDDL definitions provided define nested containers. Typically, the CDDL types used for nested containers are maps. Every key used in the maps is a named type that is associated with an corresponding uint via a block of rules appended at the end of the CDDL definition.

Every set of uint keys that is used in the context of the "collision domain" of map is intended to be collision-free (each key is intended to be unique in the scope of a map, not a multimap). To accomplish that, for each map there is an IANA registry for the map members of maps.

2.1. Typographical Conventions

Type names in the following CDDL definitions follow the naming convention illustrated in table Table 1.

type trait	example	typo 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 Convention

2.2. Prefixes and Namespaces

The semantics of the information elements (attributes) defined for CoRIM, CoMID tags, and CoSWID tags are sometimes very similar, but often do not share the same scope or are actually quite different. In order to not overload the already existing semantics of the software-centric IANA registries of CoSWID tags with, for example, hardware-centric semantics of CoMID tags, new type names are introduced. For example: both CoSWID tags and CoMID tags define a tag-id. As CoSWID already specifies tag-id, the tag-id in CoMID tags is prefixed with `comid.` to disambiguate the context, resulting in `comid.tag-id`. This prefixing provides a well-defined scope for the use of the types defined in this document and guarantees interoperability (no type name collisions) with the CoSWID CDDL definition. Effectively, the prefixes used in this specification enable simple hierarchical namespaces. The prefixing introduced is also based on the anticipated namespace features for CDDL.

2.3. Extensibility

Both the CoRIM and the CoMID tag specification include extension points using CDDL sockets (see [RFC8610] Section 3.9). The use of CDDL sockets allows for well-formed extensions to be defined in supplementary CDDL definitions that support additional uses of CoRIM and CoMID tags.

There are two types of extensibility supported via the extension points defined in this document. Both types allow for the addition of keys in the scope of a map.

Custom Keys: The CDDL definition allows for the use of negative integers as keys. These keys cannot take on a well-defined global semantic. They can take on custom-defined semantics in a limited or local scope, e.g. vendor-defined scope.

Registered Keys: Additional keys can be registered at IANA via separate specifications.

Both types of extensibility also allow for the definition of new nested maps that again can include additional defined keys.

2.4. Concise RIM Extension Points

The following CDDL sockets (extension points) are defined in the CoRIM specification, which allow the addition of new information structures to their respective CDDL groups.

Map Name	CDDL Socket	Defined in
corim-entity-map	\$\$corim-entity-map-extension	Section 3.2.1
unsigned-corim-map	\$\$unsigned-corim-map-extension	Section 3.2
concise-mid-tag	\$\$comid-extension	Section 3.3
tag-identity-map	\$\$tag-identity-map-extension	Section 3.4
entity-map	\$\$entity-map-extension	Section 3.5
triples-map	\$\$triples-map-extension	Section 3.7
measurement-values-map	\$\$measurement-values-map-extension	Section 3.10

Table 2: CoMID CDDL Group Extension Points

2.5. CDDL Generic Types

The CDDL definitions for CoRIM and CoMID tags use the two following generic types.

2.5.1. 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 optional members.

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

2.5.2. One-Or-More

The one-or-more generic type allows to omit an encapsulating array, if only one member would be present.

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

3. Concise RIM Data Definition

A CoRIM is a bundle of CoMID tags and/or CoSWID tags that can reference each other and that includes additional metadata about that bundle.

The root of the CDDL specification provided for CoRIM is the rule `corim` :

```
start = corim
```

3.1. The signed-corim Container

A CoRIM is signed using [RFC7231]. The additional CoRIM-specific COSE header member label `corim-meta` is defined as well as the corresponding type `corim-meta-map` as its value. This rule and its constraints MUST be followed when generating or validating a signed CoRIM tag.

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

```
protected-corim-header-map = {  
  corim.alg-id => int  
  corim.content-type => "application/corim-unsigned+cbor"  
  corim.issuer-key-id => bstr  
  corim.meta => bstr .cbor corim-meta-map  
  * cose-label => cose-values  
}
```

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

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

3.1.1. The corim-meta-map Container

This map contains the two additionally defined attributes `corim-signer-map` and `validity-map` that are used to annotate a CoRIM with metadata.

```
corim-meta-map = {  
  corim.signer => corim-signer-map  
  ? corim.signature-validity => validity-map  
}
```

corim.signer: One or more entities that created and/or signed the issued CoRIM.

corim.signature-validity: A time period defining the validity span of the signature over the CoRIM.

3.1.2. The corim-signer-map Container

This map is used to identify the signer of a CoRIM via a name and an optional URI.

```
corim-signer-map = {  
  corim.signer-name => $entity-name-type-choice  
  ? corim.signer-uri => uri  
  * $$corim-signer-map-extension  
}
```

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

corim.signer-name: The name of the organization that signs this CoRIM

corim.signer-uri: An URI uniquely linked to the organization that signs this CoRIM

\$\$corim-signer-map-extension: This CDDL socket is used to add new information elements to the corim-signer-map container. See **FIXME**.

3.1.3. The validity-map Container

The members of this map indicate the life-span or period of validity of a CoRIM that is baked into the protected header at the time of signing.

```
validity-map = {  
  ? corim.not-before => time  
  corim.not-after => time  
}
```

corim.not-before: The timestamp indicating the CoRIM's begin of its validity period.

corim.not-after: The timestamp indicating the CoRIM's end of its validity period.

3.2. The corim-map Container

This map contains the payload of the COSE envelope that is used to sign the CoRIM. This rule and its constraints MUST be followed when generating or validating an unsigned Concise RIM.

```
corim-map = {  
  corim.id => $corim-id-type-choice  
  corim.tags => [ + $concise-tag-type-choice ]  
  ? corim.dependent-rims => [ + corim-locator-map ]  
  ? corim.profile => [ + profile-type-choice ]  
  ? corim.rim-validity => validity-map  
  ? corim.entities => [ + corim-entity-map ]  
  * $$corim-map-extension  
}
```

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

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

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

corim.id: Typically a UUID or a text string that MUST uniquely identify a CoRIM in a given scope.

corim.tags: A collection of one or more CoMID tags and/or CoSWID tags.

corim.dependent-rims: One or more services available via the Internet that can supply additional, possibly dependent manifests (or other associated resources).

corim.profile: One or more profiles that define the domain of interpretation of the CoMID and/or CoSWID tags.

corim.rim-validity: The validity of the CoRIM expressed as a validity-map.

corim.entities: One or more entities involved in the creation of this CoRIM.

\$\$corim-map-extension: This CDDL socket is used to add new information elements to the corim-map container. See FIXME.

3.2.1. The corim-entity-map Container

This Container contains qualifying attributes that provide more context information about the RIM as well its origin and purpose. This rule and its constraints MUST be followed when generating or validating a CoRIM tag

```
corim-entity-map = {  
  corim.entity-name => $entity-name-type-choice  
  ? corim.reg-id => uri  
  corim.role => $corim-role-type-choice  
  * $$corim-entity-map-extension  
}
```

\$corim-role-type-choice /= corim.manifest-creator

corim.entity-name: The name of an organization that performs the roles as indicated by comid.role.

corim.reg-id: The registration identifier of the organization that has authority over the namespace for comid.entity-name.

corim.role: The list of roles the entity is associated with. The entity that generates the CoRIM SHOULD include a \$comid-role-type-choice value of corim.manifest-creator.

\$\$corim-entity-map-extension: This CDDL socket is used to add new information elements to the corim-entity-map container. See FIXME.

3.2.2. The corim-locator-map Container

This map is used to locate and verify the integrity of resources provided by external services, e.g. the CoRIM provider.

```
corim-locator-map = {  
  corim.href => uri  
  ? corim.thumbprint => hash-entry  
}
```

corim.href: A pointer to a services that supplies dependent files or records.

corim.thumbprint: A digest of the reference resource.

3.3. The concise-mid-tag Container

The CDDL specification for the root concise-mid-tag map is as follows. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
concise-mid-tag = {  
  ? comid.language => language-type  
  comid.tag-identity => tag-identity-map  
  ? comid.entities => [ + entity-map ]  
  ? comid.linked-tags => [ + linked-tag-map ]  
  comid.triples => triples-map  
  * $$concise-mid-tag-extension  
}
```

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

comid.language: A textual language tag that conforms with the IANA Language Subtag Registry [IANA.language-subtag-registry].

comid.tag-identity: A composite identifier containing identifying attributes that enable global unique identification of a CoMID tag across versions.

comid.entity: A list of entities that contributed to the CoMID tag.

comid.linked-tags: A list of tags that are linked to this CoMID tag.

comid.triples: A set of relationships in the form of triples, representing a graph-like and semantic reference structure between tags.

\$\$comid-mid-tag-extension: This CDDL socket is used to add new information elements to the concise-mid-tag root container. See FIXME.

3.4. The tag-identity-map Container

The CDDL specification for the tag-identity-map includes all identifying attributes that enable a consumer of information to anticipate required capabilities to process the corresponding tag that map is included in. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
tag-identity-map = {  
  comid.tag-id => $tag-id-type-choice  
  comid.tag-version => tag-version-type  
}
```

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

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

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

comid.tag-id: An identifier for a CoMID that MUST be globally unique.

comid.tag-version: An unsigned integer used as a version identifier.

\$\$tag-identity-map-extension: This CDDL socket is used to add new information elements to the tag-identity-map container. See FIXME.

3.5. The entity-map Container

This Container provides qualifying attributes that provide more context information describing the module as well its origin and purpose. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
entity-map = {  
  comid.entity-name => $entity-name-type-choice  
  ? comid.reg-id => uri  
  comid.role => one-or-more<$comid-role-type-choice>  
  * $$entity-map-extension  
}
```

```
$comid-role-type-choice /= comid.tag-creator  
$comid-role-type-choice /= comid.creator  
$comid-role-type-choice /= comid.maintainer
```

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

comid.entity-name: The name of an organization that performs the roles as indicated by comid.role.

comid.reg-id: The registration identifier of the organization that has authority over the namespace for comid.entity-name.

comid.role: The list of roles a CoMID entity is associated with. The entity that generates the concise-mid-tag SHOULD include a \$comid-role-type-choice value of comid.tag-creator.

\$\$entity-map-extension: This CDDL socket is used to add new information elements to the entity-map container. See FIXME.

3.6. The linked-tag-map Container

A list of tags that are linked to this CoMID tag.

```
linked-tag-map = {  
  comid.linked-tag-id => $tag-id-type-choice  
  comid.tag-rel => $tag-rel-type-choice  
}
```

```
$tag-rel-type-choice /= comid.supplements  
$tag-rel-type-choice /= comid.replaces
```

The following describes each member of the linked-tag-map container.

comid.linked-tag-id: The tag-id of the linked tag. A linked tag MAY be a CoMID tag or a CoSWID tag.

comid.tag-rel: The relationship type with the linked tag. The relationship type MAY be supplements or replaces, as well as other types well-defined by additional specifications.

3.7. The triples-map Container

A set of directed properties that associate sets of data to provide reference values, endorsed values, verification key material or identifying key material for a specific hardware module that is a component of a composite device. The map provides the core element of CoMID tags that associate remote attestation relevant data with a distinct hardware component that runs an execution environment (a module that is either a Target Environment and/or an Attesting Environment). This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
triples-map = non-empty<{  
  ? comid.reference-triples => one-or-more<reference-triple-record>  
  ? comid.endorsed-triples => one-or-more<endorsed-triple-record>  
  ? comid.attest-key-triples => one-or-more<attest-key-triple-record>  
  ? comid.identity-triples => one-or-more<identity-triple-record>  
  * $$triples-map-extension  

```


The following describes each member of the triple-map container.

`comid.reference-triples`: A directed property that associates reference measurements with a module that is a Target Environment.

`comid.endorsed-triples`: A directed property that associates endorsed measurements with a module that is a Target Environment or Attesting Environment.

`comid.attest-key-triples`: A directed property that associates key material used to verify evidence generated from a module that is an attesting environment.

`comid.identity-triples`: A directed property that associates key material used to identify a module instance or a module class that is an identifying part of a device(-set).

`$$triples-map-extension`: This CDDL socket is used to add new information elements to the triples-map container. See FIXME.

3.8. The environment-map Container

This map represents the module(s) that a triple-map can point directed properties (relationships) from in order to associate them with external information for remote attestation, such as reference values, endorsement and endorsed values, verification key material for evidence, or identifying key material for module (re-)identification. This map can identify a single module instance via `comid.instance` or groups of modules via `comid.group`. Referencing classes of modules requires the use of the more complex class-map container. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
environment-map = non-empty<{
  ? comid.class => class-map
  ? comid.instance => $instance-id-type-choice
  ? comid.group => $group-id-type-choice
}>
```

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

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

The following describes each member of the environment-map container.

`comid-class`: A composite identifier for classes of environments/modules.

comid.instance: An identifier for distinct instances of environments/modules that is either a UEID or a UUID.

comid.group: An identifier for a group of environments/modules that is a UUID.

3.9. The class-map Container

This map enables a composite identifier intended to uniquely identify modules that are of a distinct class of devices. Effectively, all provided members in combination are a composite module class identifier. This rule and its constraints MUST be followed when generating or validating a CoMID tag. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
class-map = non-empty<{  
  ? comid.class-id => $class-id-type-choice  
  ? comid.vendor => tstr  
  ? comid.model => tstr  
  ? comid.layer => uint  
  ? comid.index => 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 container.

comid.class-id: TODO

comid.vendor TODO

comid.model TODO

comid.layer TODO

comid.index TODO

3.10. The measurement-map and measurement-values-map Containers

One of the targets (range) that a triple-map can point to in order to associate it with a module (domain) is the measurement-map. This map is used to provide reference measurements values that can be compared with Evidence Claim values or Endorsements and endorsed values from other sources than the corresponding CoRIM. measurement-map comes with a measurement key that identifies the measured element with via a OID reference or a UUID. measurement-values-map contains the actual

measurements associated with the module(s). Byte strings with corresponding bit masks that highlights which bits in the byte string are used as reference measurements or endorsement are located in raw-value-group. The members of measurement-values-map provide well-defined and well-scoped semantics for reference measurement or endorsements with respect to a given module instance, class, or group. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
measurement-map = {
  ? comid.mkey => $measured-element-type-choice
  comid.mval => measurement-values-map
}
```

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

```
measurement-values-map = non-empty<{
  ? comid.ver => version-map
  ? comid.svn => svn-type-choice
  ? comid.digests => digests-type
  ? comid.flags => flags-type
  ? raw-value-group
  ? comid.mac-addr => mac-addr-type-choice
  ? comid.ip-addr => ip-addr-type-choice
  ? comid.serial-number => serial-number-type
  ? comid.ueid => ueid-type
  ? comid.uuid => uuid-type
  ? comid.name => tstr
  * $$measurement-values-map-extension
}>
```

```
flags-type = bytes .bits operational-flags
```

```
$operational-flags /= &( not-configured: 0 )
$operational-flags /= &( not-secure: 1 )
$operational-flags /= &( recovery: 2 )
$operational-flags /= &( debug: 3 )
$operational-flags /= &( not-replay-protected: 4 )
$operational-flags /= &( not-integrity-protected: 5 )
```

```
serial-number-type = text
```

```
digests-type = [ + hash-entry ]
```

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

comid.mkey: An identifier for the set of measurements expressed in measurement-values-map that is either an OID or a UUID.

comid.ver: A version number measurement.

comid.svn: A security related version number measurement.

comid.digests: A digest (typically a hash value) measurement.

comid.flags: Measurements that reflect operational modes that are made permanent at manufacturing time such that they are not expected to change during normal operation of the Attester.

raw-value-group: A measurement in the form of a byte string that can come with a corresponding bit mask defining the relevance of each bit in the byte string as a measurement.

comid.mac-addr: An EUI-48 or EUI-64 MAC address measurement.

comid.ip-addr: An Ipv4 or Ipv6 address measurement.

comid.serial-number: A measurement of a serial number in text.

comid.ueid: A measurement of a Unique Enough Identifier (UEID).

comid.uuid: A measurement of a Universally Unique Identifier (UUID).

comid.name: TODO

\$\$measurement-values-map-extension: This CDDL socket is used to add new information elements to the measurement-values-map container. See FIXME.

3.10.1. The version-map Container

This map expresses reference values about version information.

```
version-map = {  
  comid.version => version-type  
  ? comid.version-scheme => $version-scheme  
}
```

```
version-type = text .default '0.0.0'
```

The following describes each member of the version-map container.

comid.version: The version in the form of a text string.

comid-version-scheme: The version-scheme of the text string value as defined in [I-D.ietf-sacm-coswid]

3.10.2. The svn-type-choice Enumeration

This choice defines the CBOR tagged Security Version Numbers (SVN) that can be used as reference values for Evidence and Endorsements.

```
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
```

The following describes the types in the svn-type-choice enumeration.

tagged-svn: A specific SVN.

tagged-min-svn: A lower bound for allowed SVN.

3.10.3. The raw-value-group Container

FIXME This group can express a single raw byte value and can come with an optional bit mask that defines which bits in the byte string is used as a reference value, by setting corresponding position in the bit mask to 1.

```
raw-value-group = (
  comid.raw-value => $raw-value-type-choice
  ? comid.raw-value-mask => raw-value-mask-type
)
```

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

raw-value-mask-type = bytes

The following describes the types in the raw-value-group Container.

comid.raw-value: FIXME Bit positions in raw-value-type that correspond to bit positions in raw-value-mask-type.

comid.raw-value-mask: A raw-value-mask-type bit corresponding to a bit in raw-value-type MUST be 1 to evaluate the corresponding raw-value-type bit.

3.10.4. The ip-addr-type-choice Enumeration

This type choice expresses IP addresses as reference values.

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

3.10.5. The mac-addr-type-choice Enumeration

This type choice expresses MAC addresses as reference values.

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

3.11. The verification-key-map Container

One of the targets (range) that a triple-map can point to in order to associate it with a module (domain). This map is used to provide the key material for evidence verification (effectively signature checking or a lightweight proof-of-possession of private signing key material) or for identity assertion/check (where a proof-of-possession implies a certain device identity). In support of informed trust decisions, an optional trust anchor in the form a PKIX certification path that is associated with the provided key material can be included. This rule and its constraints MUST be followed when generating or validating a CoMID tag.

```
verification-key-map = {
  comid.key => pkix-base64-key-type
  ? comid.keychain => [ + pkix-base64-cert-type ]
}
```

```
pkix-base64-key-type = tstr
pkix-base64-cert-type = tstr
```

The following describes each member of the verification-key-map container.

comid.key: Verification key material in DER format base64 encoded. Typically, but not necessarily, a public key.

comid.keychain: One or more base64 encoded PKIX certificates. The certificate containing the public key in comid.key MUST be the first certificate. Additional certificates MAY follow. Each subsequent certificate SHOULD certify the previous certificate.

4. Full CDDL Definition

This section aggregates the CDDL definitions specified in this document in a full CDDL definitions including:

- * the COSE envelope for CoRIM: signed-corim
- * the CoRIM document: unsigned-corim
- * the CoMID document: concise-mid-tag

Not included in the full CDDL definition are CDDL dependencies to CoSWID. The following CDDL definitions can be found in [I-D.ietf-sacm-coswid]:

- * the COSE envelope for CoRIM: signed-coswid
- * the CoSWID document: concise-swid-tag

<CODE BEGINS>

```
corim = #6.500($concise-reference-integrity-manifest-type-choice)
```

```
$concise-reference-integrity-manifest-type-choice /= #6.501(unsigned-corim-map)
```

```
$concise-reference-integrity-manifest-type-choice /= #6.502(signed-corim)
```

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

```
protected-signed-corim-header-map = {
  corim.alg-id => int
  corim.content-type => "application/rim+cbor"
  corim.issuer-key-id => bstr
  corim.meta => corim-meta-map
  * cose-label => cose-values
}
```

```
corim-meta-map = {
  corim.signer => [ + corim-entity-map ]
  ? corim.validity => validity-map
}
```

```
corim-entity-map = {
  corim.entity-name => $entity-name-type-choice
  ? corim.reg-id => uri
  corim.role => $corim-role-type-choice
  * $$corim-entity-map-extension
}
```

```
$corim-role-type-choice /= corim.manifest-creator
$corim-role-type-choice /= corim.manifest-signer

validity-map = {
  ? corim.not-before => time
  corim.not-after => time
}

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

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

unsigned-corim-map = {
  corim.id => $corim-id-type-choice
  corim.tags => [ + $concise-tag-type-choice ]
  ? corim.dependent-rims => [ + corim-locator-map ]
  ? corim.profile => [ + profile-type-choice ]
  * $$unsigned-corim-map-extension
}

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

corim-locator-map = {
  corim.href => uri
  ? corim.thumbprint => hash-entry
}

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

concise-mid-tag = {
  ? comid.language => language-type
  comid.tag-identity => tag-identity-map
  ? comid.entity => [ + entity-map ]
  ? comid.linked-tags => [ + linked-tag-map ]
  comid.triples => triples-map
  * $$concise-mid-tag-extension
}
```



```
language-type = text

tag-identity-map = {
  comid.tag-id => $tag-id-type-choice
  ? comid.tag-version => tag-version-type
}

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

tag-version-type = uint .default 0

entity-map = {
  comid.entity-name => $entity-name-type-choice
  ? comid.reg-id => uri
  comid.role => [ + $comid-role-type-choice ]
  * $$entity-map-extension
}

$comid-role-type-choice /= comid.tag-creator
$comid-role-type-choice /= comid.creator
$comid-role-type-choice /= comid.maintainer

linked-tag-map = {
  comid.linked-tag-id => $tag-id-type-choice
  comid.tag-rel => $tag-rel-type-choice
}

$tag-rel-type-choice /= comid.supplements
$tag-rel-type-choice /= comid.replaces

triples-map = non-empty<{
  ? comid.reference-triples => [ + reference-triple-record ]
  ? comid.endorsed-triples => [ + endorsed-triple-record ]
  ? comid.attest-key-triples => [ + attest-key-triple-record ]
  ? comid.identity-triples => [ + identity-triple-record ]
  * $$triples-map-extension
}>

reference-triple-record = [
  environment-map ; target environment
  [ + measurement-map ] ; reference measurements
]

endorsed-triple-record = [
  environment-map ; (target or attesting) environment
  [ + measurement-map ] ; endorsed measurements
]
```

```
attest-key-triple-record = [  
  environment-map ; attesting environment  
  [ + verification-key-map ] ; attestation verification key(s)  
]  
  
identity-triple-record = [  
  environment-map ; device identifier (instance or class)  
  [ + verification-key-map ] ; DevID, or semantically equivalent  
]  
  
pkix-base64-key-type = tstr  
pkix-base64-cert-type = tstr  
  
verification-key-map = {  
  ; Verification key in DER format base64-encoded.  
  ; Typically, but not necessarily a public key.  
  comid.key => pkix-base64-key-type  
  ; Optional X.509 certificate chain corresponding to the public key  
  ; in comid.key, encoded as an array of one or more base64-encoded  
  ; DER PKIX certificates. The certificate containing the public key  
  ; in comid.key MUST be the first certificate. This MAY be followed  
  ; by additional certificates, with each subsequent certificate  
  ; being the one used to certify the previous one.  
  ? comid.keychain => [ + pkix-base64-cert-type ]  
}  
  
environment-map = non-empty<{  
  ? comid.class => class-map  
  ? comid.instance => $instance-id-type-choice  
  ? comid.group => $group-id-type-choice  
  
class-map = non-empty<{  
  ? comid.class-id => $class-id-type-choice  
  ? comid.vendor => tstr  
  ? comid.model => tstr  
  ? comid.layer => uint  
  ? comid.index => uint  
  
$class-id-type-choice /= tagged-oid-type  
$class-id-type-choice /= tagged-uuid-type  
  
$instance-id-type-choice /= tagged-uuid-type  
$instance-id-type-choice /= tagged-uuid-type  
  
$group-id-type-choice /= tagged-uuid-type
```

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

tagged-uuid-type = #6.37(uuid-type)

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

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

measurement-map = {
  ? comid.mkey => $measured-element-type-choice
  comid.mval => measurement-values-map
}

measurement-values-map = non-empty<{
  ? comid.ver => version-map
  ? comid.svn => svn-type-choice
  ? comid.digests => digests-type
  ? comid.flags => flags-type
  ? raw-value-group
  ? comid.mac-addr => mac-addr-type-choice
  ? comid.ip-addr => ip-addr-type-choice
  ? comid.serial-number => serial-number-type
  ? comid.ueid => ueid-type
  ? comid.uuid => uuid-type
  * $$measurement-values-map-extension
}>

version-map = {
  comid.version => version-type
  ? comid.version-scheme => $version-scheme
}
version-type = text .default '0.0.0'

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

flags-type = bytes .bits operational-flags

operational-flags = &(
  not-configured: 0
  not-secure: 1
  recovery: 2
```

```
    debug: 3
  )

raw-value-group = (
  comid.raw-value => raw-value-type
  ? comid.raw-value-mask => raw-value-mask-type
)

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

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

serial-number-type = text

digests-type = [ + hash-entry ]

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

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

any-uri = uri
label = text / int
```

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

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

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

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

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

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

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

link-entry = {
  ? artifact => text,
  href => any-uri,
  ? media => text,
  ? ownership => $ownership,
  rel => $rel,
  ? media-type => text,
  ? use => $use,
```

```
    * $$link-extension,
    global-attributes,
  }

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

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

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

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

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

```
        ? file => one-or-more<file-entry>,
    )

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

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

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

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

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

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

payload-entry = {
    resource-collection,
```

```
    * $$payload-extension,  
    global-attributes,  
  }  
  
  evidence-entry = {  
    resource-collection,  
    ? date => integer-time,  
    ? device-id => text,  
    * $$evidence-extension,  
    global-attributes,  
  }  
  
  integer-time = #6.1(int)  
  
  tag-id = 0  
  software-name = 1  
  entity = 2  
  evidence = 3  
  link = 4  
  software-meta = 5  
  payload = 6  
  hash = 7  
  corpus = 8  
  patch = 9  
  media = 10  
  supplemental = 11  
  tag-version = 12  
  software-version = 13  
  version-scheme = 14  
  lang = 15  
  directory = 16  
  file = 17  
  process = 18  
  resource = 19  
  size = 20  
  file-version = 21  
  key = 22  
  location = 23  
  fs-name = 24  
  root = 25  
  path-elements = 26  
  process-name = 27  
  pid = 28  
  type = 29  
  entity-name = 31  
  reg-id = 32  
  role = 33  
  thumbprint = 34
```


date = 35
device-id = 36
artifact = 37
href = 38
ownership = 39
rel = 40
media-type = 41
use = 42
activation-status = 43
channel-type = 44
colloquial-version = 45
description = 46
edition = 47
entitlement-data-required = 48
entitlement-key = 49
generator = 50
persistent-id = 51
product = 52
product-family = 53
revision = 54
summary = 55
unspsc-code = 56
unspsc-version = 57

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

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

shared=1
private=2
abandon=3

ancestor=1
component=2
feature=3
installationmedia=4
packageinstaller=5
parent=6
patches=7

```
requires=8
see-also=9
supersedes=10

optional=1
required=2
recommended=3

comid.language = 0
comid.tag-identity = 1
comid.entity = 2
comid.linked-tags = 3
comid.triples = 4

comid.tag-id = 0
comid.tag-version = 1

comid.entity-name = 0
comid.reg-id = 1
comid.role = 2

comid.linked-tag-id = 0
comid.tag-rel = 1

comid.reference-triples = 0
comid.endorsed-triples = 1
comid.identity-triples = 2
comid.attest-key-triples = 3

comid.class = 0
comid.instance = 1
comid.group = 2

comid.class-id = 0
comid.vendor = 1
comid.model = 2
comid.layer = 3
comid.index = 4

comid.mkey = 0
comid.mval = 1

comid.ver = 0
comid.svn = 1
comid.digests = 2
comid.flags = 3
comid.raw-value = 4
comid.raw-value-mask = 5
```

```
comid.mac-addr = 6
comid.ip-addr = 7
comid.serial-number = 8
comid.ueid = 9
comid.uuid = 10

comid.key = 0
comid.keychain = 1

comid.version = 0
comid.version-scheme = 1

comid.supplements = 0

comid.replaces = 1

comid.tag-creator = 0
comid.creator = 1
comid.maintainer = 2

corim.id = 0
corim.tags = 1
corim.dependent-rims = 2
corim.profile = 3

corim.href = 0
corim.thumbprint = 1

corim.alg-id = 1
corim.content-type = 3
corim.issuer-key-id = 4
corim.meta = 8

corim.not-before = 0
corim.not-after = 1

corim.signer = 0
corim.validity = 1

corim.entity-name = 0
corim.reg-id = 1
corim.role = 2

corim.manifest-creator = 1

corim.manifest-signer = 2
```

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

```
cose-label = int / tstr  
cose-values = any
```

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

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

```
uuid-type = bytes .size 16
```

```
<CODE ENDS>
```

5. Privacy Considerations

Privacy Considerations

6. Security Considerations

Security Considerations

7. IANA Considerations

This document has a number of IANA considerations, as described in the following subsections. In summary, 6 new registries are established with this request, with initial entries provided for each registry. New values for 5 other registries are also requested.

7.1. COSE Header Parameters Registry

The 'corim metadata' parameter has been added to the "COSE Header Parameters" registry:

- * Name: 'corim metadata'
- * Label: 11
- * Value: corim-meta-map
- * Description: Provides a map of additional metadata for a CoRIM payload composed of (1) one or more entities that created or signed the corresponding CoRIM and (2) its period of validity

- * Reference: 'corim-meta-map' in {model-corim-meta-map} of this document

7.2. CoRIM Map Items Registry

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

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

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

Table 3: CoRIM Map Items
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	corim.id	RFC-AAAA
1	corim.tags	RFC-AAAA
2	corim.dependent-rims	RFC-AAAA
3-255	Unassigned	

Table 4: CoRIM Map Items Initial Registrations

7.3. CoRIM Entity-Map Items Registry

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

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

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

Table 5: CoRIM Entity Map Items
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	corim.entity-name	RFC-AAAA
1	corim.reg-id	RFC-AAAA
2	corim.role	RFC-AAAA
3-255	Unassigned	

Table 6: CoRIM Entity Map Items Initial
Registrations

7.4. CoRIM Entity-Types Registry

This document defines a new registry titled "CoRIM Entity Types". The registry maintains well-defined integer values as choices for '\$entity-name-type-choice' CBOR uints.

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

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

Table 7: CoRIM Entity Types
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	corim.manifest-creator	RFC-AAAA
1	corim.manifest-signer	RFC-AAAA
2-255	Unassigned	

Table 8: CoRIM Entity Types Initial Registrations

7.5. CoMID Map Items Registry

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

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

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

Table 9: CoMID Map Items
Registration Procedures

All negative values are reserved for Private Use.

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

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

Table 10: CoMID Map Items Initial Registrations

7.6. CoMID Entity-Map Items Registry

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

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

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

Table 11: CoMID Entity Map Items Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.entity-name	RFC-AAAA
1	comid.reg-id	RFC-AAAA
2	comid.role	RFC-AAAA
3-255	Unassigned	

Table 12: CoMID Entity Map Items Initial Registrations

7.7. CoMID Triples-Map Items Registry

This document defines a new registry titled "CoMID Triples Map". The registry uses integer values as index values for items in 'comid-triples-map' CBOR maps.

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

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

Table 13: CoMID triples Map Items Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.reference-triples	RFC-AAAA
1	comid.endorsed-triples	RFC-AAAA
2	comid.identity-triples	RFC-AAAA
3	comid.attest-key-triples	RFC-AAAA
4-255	Unassigned	

Table 14: CoMID Triples Map Items Initial Registrations

7.8. CoMID Measurement-Values-Map Items Registry

This document defines a new registry titled "CoMID Measurement-Values Map". The registry uses integer values as index values for items in 'comid-measurement-values-map' CBOR maps.

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

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

Table 15: CoMID Measurement-Values Map Items Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.ver	RFC-AAAA
1	comid.svn	RFC-AAAA
2	comid.digests	RFC-AAAA
3	comid.flags	RFC-AAAA
4	comid.raw-value	RFC-AAAA
5	comid.raw-value-mask	RFC-AAAA
6	comid.mac-addr	RFC-AAAA
7	comid.ip-addr	RFC-AAAA
8	comid.serial-number	RFC-AAAA
9	comid.ueid	RFC-AAAA
10	comid.uuid	RFC-AAAA
11-255	Unassigned	

Table 16: CoMID Measurement-Values Map Items
Initial Registrations

7.9. CoMID Tag-Relationship-Types Registry

This document defines a new registry titled "CoMID Tag-Relationship Types". The registry maintains well-defined integer values as choices for '\$tag-rel-type-choice' CBOR uints.

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

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

Table 17: CoMID Tag-Relationship
Types Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.supplements	RFC-AAAA
1	comid.replaces	RFC-AAAA
2-255	Unassigned	

Table 18: CoMID Tag-Relationship Types
Initial Registrations

7.10. CoMID Role-Types Registry

This document defines a new registry titled "CoMID Role Types". The registry maintains well-defined integer values as choices for '\$comid-role-type-choice' CBOR uints.

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

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

Table 19: CoMID Role Types
Registration Procedures

All negative values are reserved for Private Use.

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

Index	Item Name	Specification
0	comid.tag-creator	RFC-AAAA
1	comid.creator	RFC-AAAA
2	comid.maintainer	RFC-AAAA
3-255	Unassigned	

Table 20: CoMID Role Types Initial
Registrations

7.11. rim+cbor Media Type Registration

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

Type name: application

Subtype name: rim+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC8949]. See RFC-AAAA for details.

Security considerations: See Section 6 of RFC-AAAA.

Interoperability considerations: Applications MAY ignore any key value pairs that they do not understand. This allows backwards compatible extensions to this specification.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by remote attestation procedures, supply chain integrity management systems, vulnerability assessment systems, and in applications that rely on trustworthy endorsements and reference values describing the intended operational state of a system.

Fragment identifier considerations: Fragment identification for application/rim+cbor is supported by using fragment identifiers as specified by Section 9.5 of [RFC8949].

Additional information:

Magic number(s): first five bytes in hex: 43 4f 52 49 4d

File extension(s): corim

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.corim conforms to public.data

Person & email address to contact for further information: Henk Birkholz <henk.birkholz@sit.fraunhofer.de>

Intended usage: COMMON

Restrictions on usage: None

Author: Henk Birkholz <henk.birkholz@sit.fraunhofer.de>

Change controller: IESG

7.12. CoAP Content-Format Registration

IANA is requested to assign a CoAP Content-Format ID for the CoRIM media type in the "CoAP Content-Formats" sub-registry, from the "IETF Review or IESG Approval" space (256..999), within the "CoRE Parameters" registry [RFC7252] [IANA.core-parameters]:

Media type	Encoding	ID	Reference
application/rim+cbor	-	TBD1	RFC-AAAA

Table 21: CoAP Content-Format IDs

7.13. 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
500	tagged array or tagged map	Concise Reference Integrity Manifest (CoRIM) [RFC-AAAA]
501	map	unsigned CoRIM [RFC-AAAA]
502	array	signed CoRIM [RFC-AAAA]
505	bstr	byte string with CBOR-encoded Concise SWID tag [RFC-AAAA]
506	bstr	byte string with CBOR-encoded Concise MID tag [RFC-AAAA]

Table 22: CoRIM CBOR Tags

7.14. CoMID 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
550	bstr	UEID with max size of 33 bytes [RFC-AAAA]
551	int	Security Version Number [RFC-AAAA]
552	int	lower bound of allowed Security Version Number [RFC-AAAA]

Table 23: CoMID CBOR Tags

8. References

8.1. 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-14, 9 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-14.txt>>.

[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-20, 26 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-20.txt>>.

[IANA.cbor-tags]

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

[IANA.core-parameters]

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

[IANA.language-subtag-registry]

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

[IANA.media-types]

IANA, "Media Types", <<http://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/info/rfc2119>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [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>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [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>>.
- [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>>.
- [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>>.

8.2. Informative References

- [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>>.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Thomas Fossati
Arm Limited
United Kingdom

Email: Thomas.Fossati@arm.com

Yogesh Deshpande
Arm Limited
United Kingdom

Email: yogesh.deshpande@arm.com

Ned Smith
Intel Corporation
United States of America

Email: ned.smith@intel.com

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 28 April 2022

H. Birkholz
Fraunhofer SIT
C. Newton
L. Chen
University of Surrey
D. Thaler
Microsoft
25 October 2021

Direct Anonymous Attestation for the Remote Attestation Procedures
Architecture
draft-birkholz-rats-daa-02

Abstract

This document maps the concept of Direct Anonymous Attestation (DAA) to the Remote Attestation Procedures (RATS) Architecture. The role DAA Issuer is introduced and its interactions with existing RATS roles is specified.

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 28 April 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Direct Anonymous Attestation	3
4. DAA changes to the RATS Architecture	5
5. Additions to Remote Attestation principles	6
6. Privacy Considerations	7
7. Security Considerations	7
8. Implementation Considerations	7
9. IANA Considerations	8
10. References	8
10.1. Normative References	8
10.2. Informative References	8
Authors' Addresses	8

1. Introduction

Remote Attestation procedures (RATS, [I-D.ietf-rats-architecture]) describe interactions between well-defined architectural constituents in support of Relying Parties that require an understanding about the trustworthiness of a remote peer. The identity of an Attester and its corresponding Attesting Environments play a vital role in RATS. A common way to refer to such an identity is the Authentication Secret ID as defined in the Reference Interaction Models for RATS [I-D.ietf-rats-reference-interaction-models]. The fact that every Attesting Environment can be uniquely identified in the context of the RATS architecture is not suitable for every application of remote attestation. Additional issues may arise when Personally identifiable information (PII) -- whether obfuscated or in clear text -- are included in attestation Evidence or even corresponding Attestation Results. This document illustrates how Direct Anonymous Attestation (DAA) can mitigate the issue of uniquely (re-)identifiable Attesting Environments. To accomplish that goal, a new RATS role -- the DAA Issuer -- is introduced and its duties as well as its interactions with other RATS roles are specified.

2. Terminology

This document uses the following set of terms, roles, and concepts as defined in [I-D.ietf-rats-architecture]: Attester, Verifier, Relying Party, Conceptual Message, Evidence, Attestation Result, Attesting Environment. The role of Endorser, also defined in [I-D.ietf-rats-architecture], needs to be adapted and details are given below.

Additionally, this document uses and adapts, as necessary, the following concepts and information elements as defined in [I-D.ietf-rats-reference-interaction-models]: Attester Identity, Authentication Secret, Authentication Secret ID

A PKIX Certificate is an X.509v3 format certificate as specified by [RFC5280].

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. Direct Anonymous Attestation

Figure 1 shows the data flows between the different RATS roles involved in DAA.

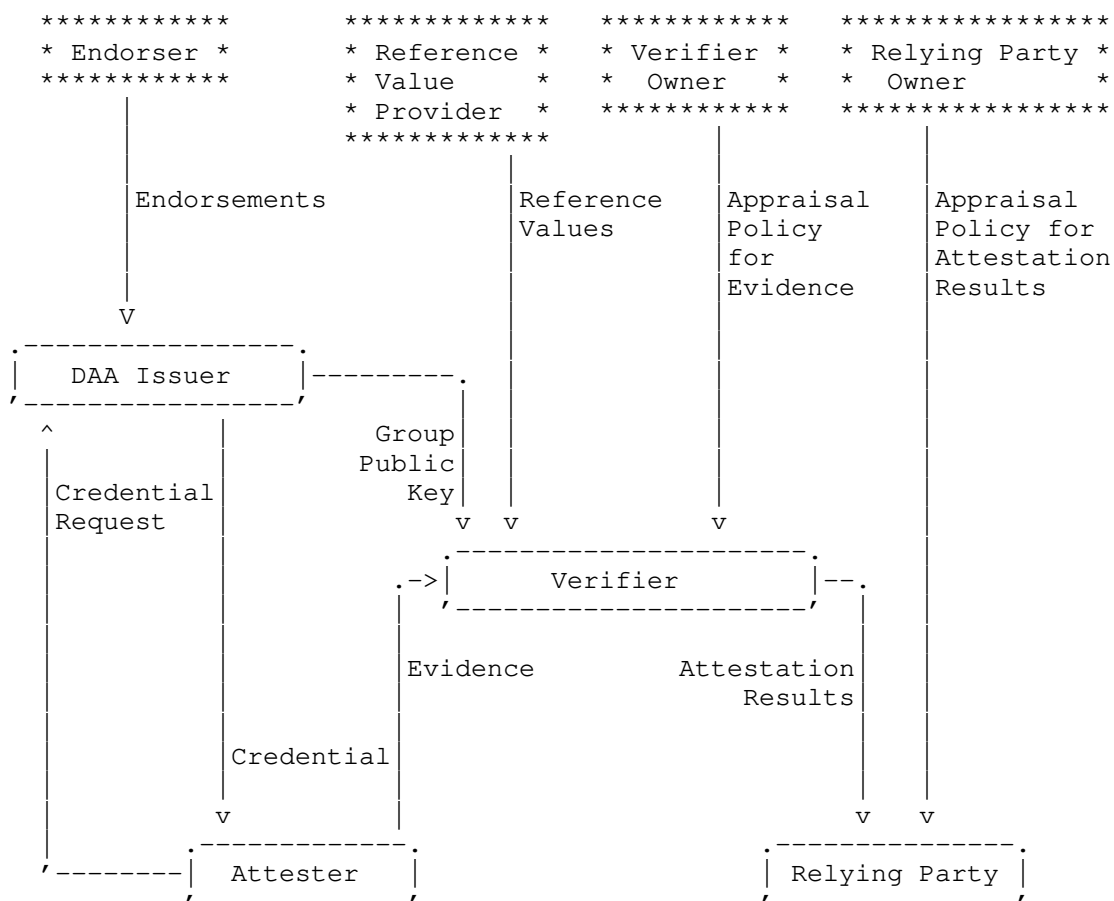


Figure 1: DAA data flows

DAA [DAA] is a signature scheme that allows the privacy of users that are associated with an Attester (e.g. its owner) to be maintained. Essentially, DAA can be seen as a group signature scheme with the feature that given a DAA signature no-one can find out who the signer is, i.e., the anonymity is not revocable. To be able to sign anonymously, an Attester has to obtain a credential from a DAA Issuer. The DAA Issuer uses a private/public key pair to generate credentials for a group of Attesters and makes the public key (in the form of a public key certificate) available to the verifier in order to enable them to validate the Evidence received.

In order to support these DAA signatures, the DAA Issuer MUST associate a single key pair with a group of Attesters and use the same key pair when creating the credentials for all of the Attesters

in this group. The DAA Issuer's group public key certificate replaces the individual Attester Identity documents during authenticity validation as a part of the appraisal of Evidence conducted by a verifier. This is in contrast to intuition that there has to be a unique Attester Identity per device.

For DAA, the role of the Endorser is essentially the same, but they now provide Attester endorsement documents to the DAA Issuer rather than directly to the verifier. These Attester endorsement documents enable the Attester to obtain a credential from the DAA Issuer.

4. DAA changes to the RATS Architecture

In order to enable the use of DAA, a new conceptual message, the Credential Request, is defined and a new role, the DAA Issuer role, is added to the roles defined in the RATS Architecture.

Credential Request: An Attester sends a Credential Request to the DAA Issuer to obtain a credential. This request contains information about the DAA key that the Attester will use to create evidence and together with Attester endorsement information that is provided by the Endorser to confirm that the request came from a valid Attester.

DAA Issuer: A RATS role that offers zero-knowledge proofs based on public-key certificates used for a group of Attesters (Group Public Keys) [DAA]. How this group of Attesters is defined is not specified here, but the group must be large enough for the necessary anonymity to be assured.

Effectively, these certificates share the semantics of Endorsements, with the following exceptions:

- * Upon receiving a Credential Request from an Attester, the associated group private key is used by the DAA Issuer to provide the Attester with a credential that it can use to convince the Verifier that its Evidence is valid. To keep their anonymity the Attester randomizes this credential each time that it is used. Although the DAA Issuer knows the Attester Identity and can associate this with the credential issued, randomisation ensures that the Attester's identity cannot be revealed to anyone, including the Issuer.
- * The Verifier can use the DAA Issuer's group public key certificate, together with the randomized credential from the Attester, to confirm that the Evidence comes from a valid Attester without revealing the Attester's identity.

- * A credential is conveyed from a DAA Issuer to an Attester in combination with the conveyance of the group public key certificate from DAA Issuer to Verifier.

5. Additions to Remote Attestation principles

In order to ensure an appropriate conveyance of Evidence via interaction models in general, the following prerequisite considering Attester Identity MUST be in place to support the implementation of interaction models.

Attestation Evidence Authenticity: Attestation Evidence MUST be correct and authentic.

In order to provide proofs of authenticity, Attestation Evidence SHOULD be cryptographically associated with an identity document that is a randomised DAA credential.

The following information elements define extensions for corresponding information elements defined in [I-D.ietf-rats-reference-interaction-models] and that are vital to all types of reference interaction models. Varying from solution to solution, generic information elements can be either included in the scope of protocol messages (instantiating Conceptual Messages defined by the RATS architecture) or can be included in additional protocol parameters of protocols that facilitate the conveyance of RATS Conceptual Messages. Ultimately, the following information elements are required by any kind of scalable remote attestation procedure using DAA with one of RATS's reference interaction models.

Attester Identity ('attesterIdentity'): _mandatory_

In DAA, the Attester's identity is not revealed to the verifier. The Attester is issued with a credential by the DAA Issuer that is randomised and then used to anonymously confirm the validity of their evidence. The evidence is verified using the DAA Issuer's group public key.

Authentication Secret IDs ('authSecID'): _mandatory_

In DAA, Authentication Secret IDs are represented by the DAA Issuer's group public key that MUST be used to create DAA credentials for the corresponding Authentication Secrets used to protect Evidence.

In DAA, an Authentication Secret ID does not identify a unique

Attesting Environment but is associated with a group of Attesting Environments. This is because an Attesting Environment should not be distinguishable and the DAA credential which represents the Attesting Environment is randomised each time it used.

6. Privacy Considerations

As outlined about for DAA to provide privacy for the Attester the DAA group must be large enough to stop the Verifier identifying the Attester.

Randomization of the DAA credential by the Attester means that collusion between the DAA Issuer and Verifier, will not give them any advantage when trying to identify the Attester.

For DAA, the Attestation Evidence conveyed to the Verifier MUST not uniquely identify the Attester. If the Attestation Evidence is unique to an Attester other cryptographic techniques can be used, for example, property based attestation. (Henk -- reference follows)

Chen L., Loehr H., Manulis M., Sadeghi AR. (2008) Property-Based Attestation without a Trusted Third Party. Information Security. ISC 2008. Lecture Notes in Computer Science, vol 5222. Springer. https://doi.org/10.1007/978-3-540-85886-7_3

7. Security Considerations

The anonymity property of DAA makes revocation difficult. Well known solutions include: 1. Rogue attester revocation -- if the an Attester's private key is compromised and known by the Verifier then any DAA signature from that Attester can be revoked. 2. EPID - Intel's Enhanced Privacy ID -- this requires the Attester to prove (as part of their Attestation) that their credential was not used to generate any signature in a signature revocation list.

There are no other special security conderations for DAA over and above those specifed in the RATS architecture document [I-D.ietf-rats-architecture].

8. Implementation Considerations

The new DAA Issuer role can be implemented in a number of ways, for example: 1. As a stand-alone service like a Certificate Authority, a Privacy CA. 2. As a part of the Attester's manufacture. The Endorser and the DAA Issuer could be the same entity and the manufacturer would then provide a certificate for the group public key to the Verifier.

9. IANA Considerations

We don't yet.

10. References

10.1. Normative References

- [DAA] Brickell, E., Camenisch, J., and L. Chen, "Direct Anonymous Attestation", page 132-145, ACM Proceedings of the 11rd ACM conference on Computer and Communications Security, 2004.
- [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>>.
- [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/info/rfc5280>>.
- [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>>.

10.2. Informative 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-12, 23 April 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-12.txt>>.
- [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-04, 26 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-reference-interaction-models-04.txt>>.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Christopher Newton
University of Surrey

Email: cn0016@surrey.ac.uk

Liqun Chen
University of Surrey

Email: liqun.chen@surrey.ac.uk

Dave Thaler
Microsoft
United States of America

Email: dthaler@microsoft.com

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 25 October 2021

H. Birkholz
Fraunhofer SIT
D. Thaler
Microsoft
M. Richardson
Sandelman Software Works
N. Smith
Intel
W. Pan
Huawei Technologies
23 April 2021

Remote Attestation Procedures Architecture
draft-ietf-rats-architecture-12

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 25 October 2021.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified 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	5
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	7
3. Architectural Overview	8
3.1. Layered Attestation Environments	12
3.2. Composite Device	14
3.3. Implementation Considerations	16
4. Terminology	17
4.1. Roles	17
4.2. Artifacts	18
5. Topological Patterns	19
5.1. Passport Model	20
5.2. Background-Check Model	21
5.3. Combinations	22
6. Roles and Entities	23
7. Trust Model	24
7.1. Relying Party	24
7.2. Attester	25
7.3. Relying Party Owner	25

7.4. Verifier	25
7.5. Endorser, Reference Value Provider, and Verifier Owner	27
8. Conceptual Messages	28
8.1. Evidence	28
8.2. Endorsements	28
8.3. Reference Values	29
8.4. Attestation Results	29
8.5. Appraisal Policies	30
9. Claims Encoding Formats	31
10. Freshness	32
10.1. Explicit Timekeeping using Synchronized Clocks	33
10.2. Implicit Timekeeping using Nonces	33
10.3. Implicit Timekeeping using Epoch IDs	33
10.4. Discussion	35
11. Privacy Considerations	35
12. Security Considerations	36
12.1. Attester and Attestation Key Protection	36
12.1.1. On-Device Attester and Key Protection	37
12.1.2. Attestation Key Provisioning Processes	37
12.2. Integrity Protection	39
12.3. Epoch ID-based Attestation	39
12.4. Trust Anchor Protection	40
13. IANA Considerations	40
14. Acknowledgments	41
15. Notable Contributions	41
16. Appendix A: Time Considerations	41
16.1. Example 1: Timestamp-based Passport Model Example	43
16.2. Example 2: Nonce-based Passport Model Example	44
16.3. Example 3: Epoch ID-based Passport Model Example	46
16.4. Example 4: Timestamp-based Background-Check Model Example	47
16.5. Example 5: Nonce-based Background-Check Model Example	48
17. References	49
17.1. Normative References	49
17.2. Informative References	49
Contributors	51
Authors' Addresses	53

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 taken out of service, or otherwise flagged for repair.

For example:

- * A bank back-end 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 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 a trustworthy report that includes identity and version information about the hardware and software on the machines attached to their network, for purposes such as inventory, audit, anomaly detection, record maintenance and/or trending reports (logging). 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, solutions start with a specific component (called a root of trust) that is intended to provide trustworthy device identity and protected storage for measurements. The system components perform a series of measurements that may be signed via functions provided by a root of trust, considered as Evidence about present 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-tee-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.

Other biometric authentication protocols such as the Chinese IFAA standard and WeChat Pay as well as Google Pay make use of remote attestation in one form or another.

Attester: Every FIDO Authenticator contains an Attester.

Relying Party: Any web site, mobile application back-end, 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.

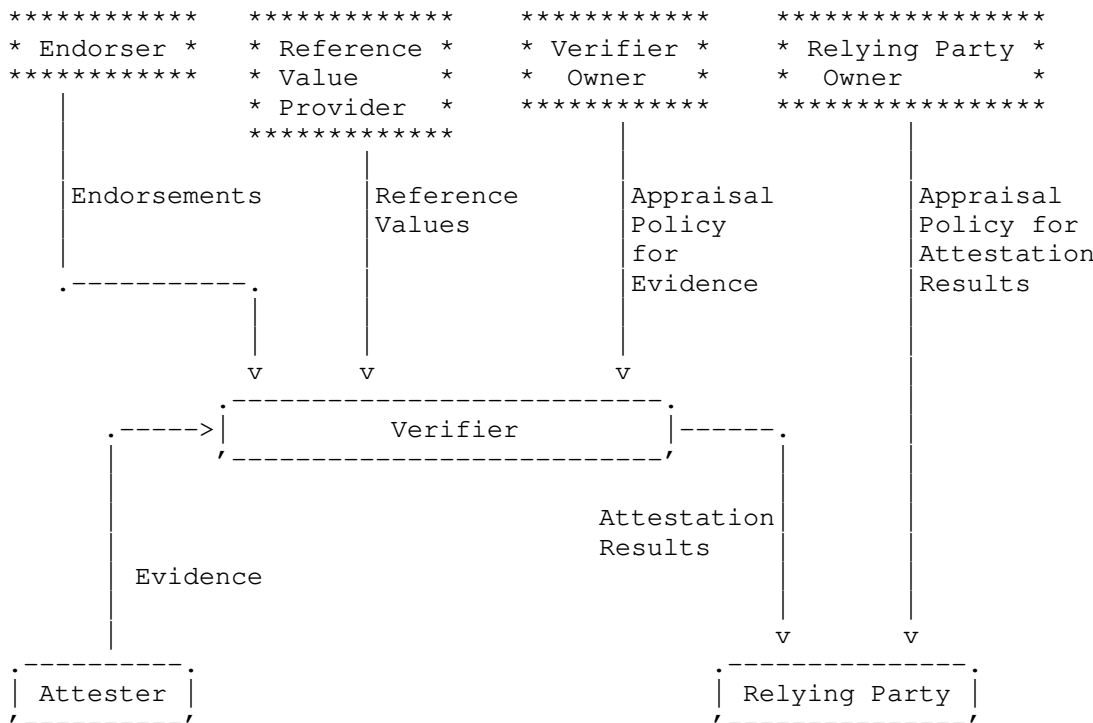


Figure 1: Conceptual Data Flow

The text below summarizes the activities conducted by the roles illustrated in Figure 1.

An Attester creates Evidence that is conveyed to a Verifier.

A Verifier uses 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 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.

A Relying Party 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. ## 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. In some implementations, the Attesting and Target Environments might be combined. Other implementations might have multiple Attesting and Target Environments, such as in the examples described in more detail in Section 3.1 and Section 3.2. Other examples may exist. All compositions of Attesting and Target Environments discussed in this architecture can be combined into more complex implementations.

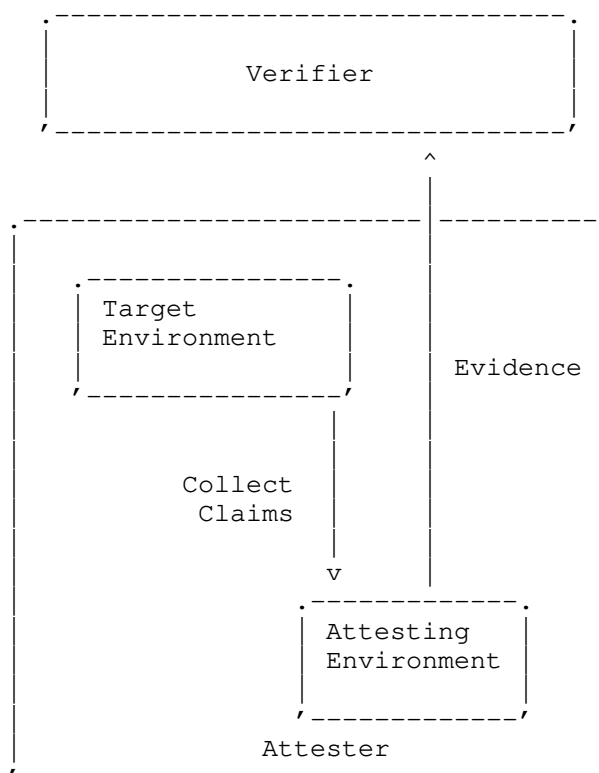


Figure 2: Two Types of Environments

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.

3.1. Layered Attestation Environments

By definition, the Attester role generates Evidence. An Attester may consist of one or more nested environments (layers). The root layer of an Attester includes at least one root of trust. In order to appraise Evidence generated by an Attester, the Verifier needs to trust the Attester's root of trust. Trust in the Attester's root of trust can be established in various ways as discussed in Section 7.4.

In layered attestation, a root of trust is the initial Attesting Environment. Claims can be collected from or about each 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 a root of trust 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.

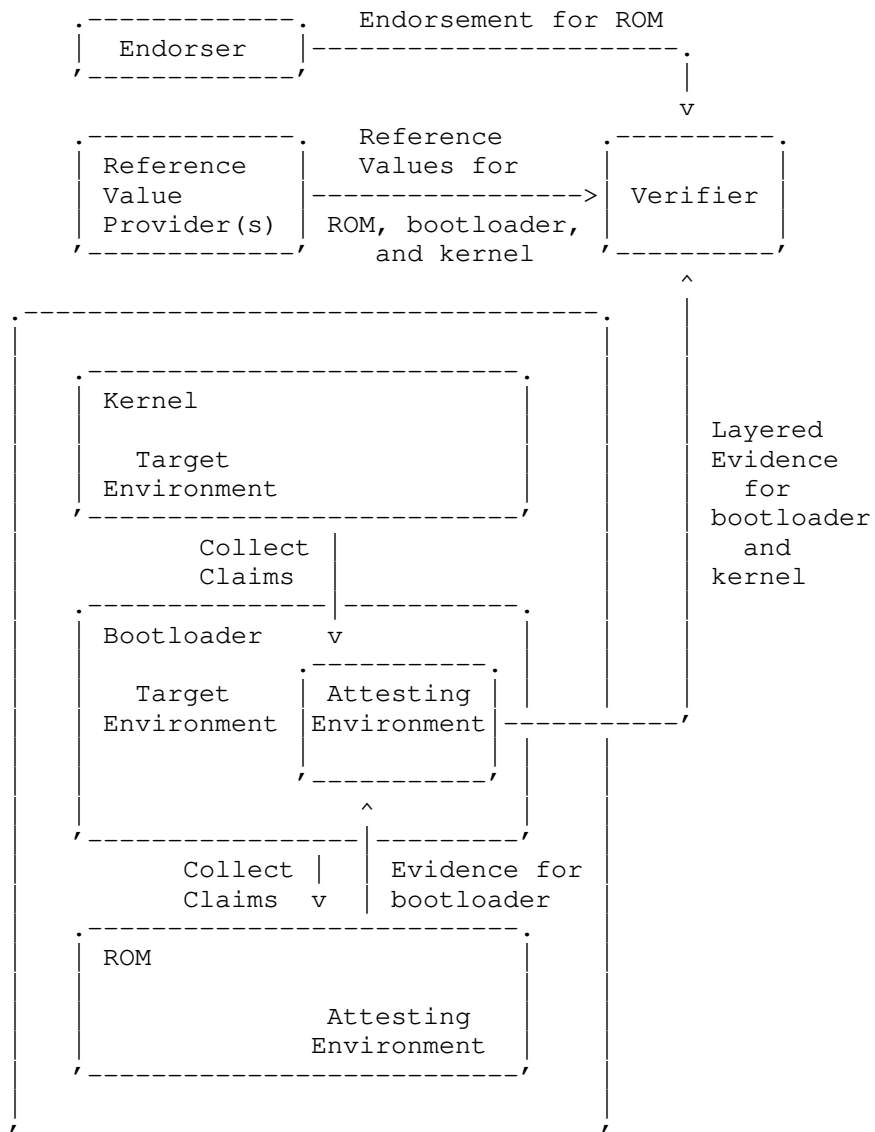


Figure 3: Layered Attester

The first Attesting Environment, the read-only BIOS 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.

3.2. 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. In consequence, 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.

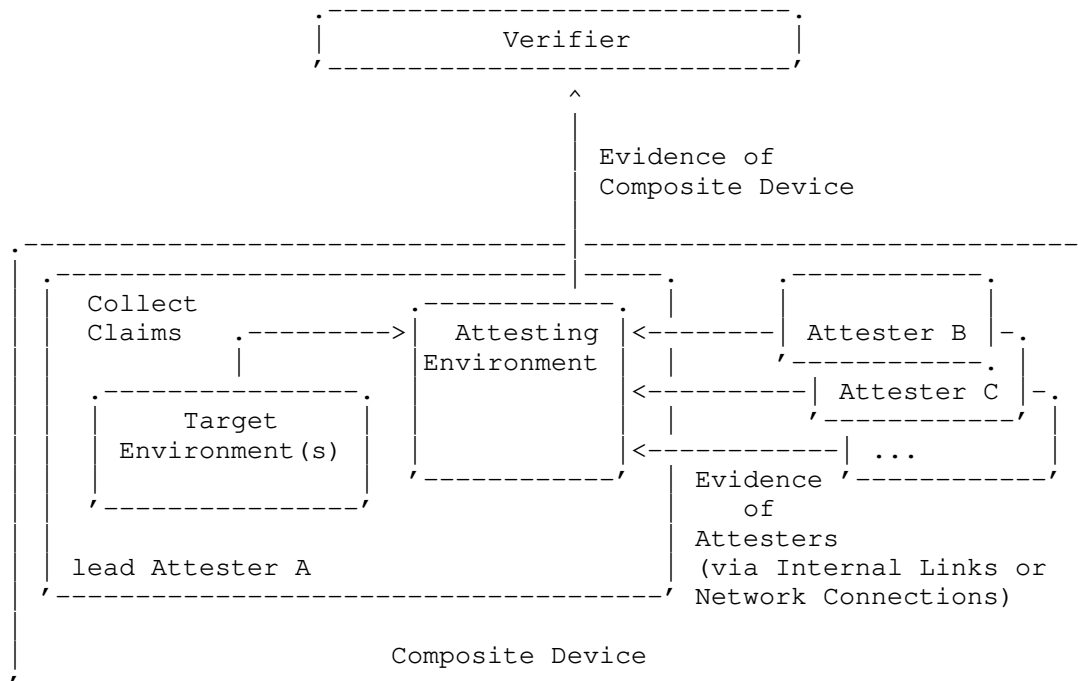


Figure 4: Composite Device

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.

3.3. 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.

4. Terminology

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

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 help Verifiers appraise the authenticity of Evidence.

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 passport issuing agency is a Verifier, 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. If the Attestation Result was a successful one, 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.

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.

Since the resource access protocol between the Attester and Relying Party includes an Attestation Result, in this model the details of that 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.

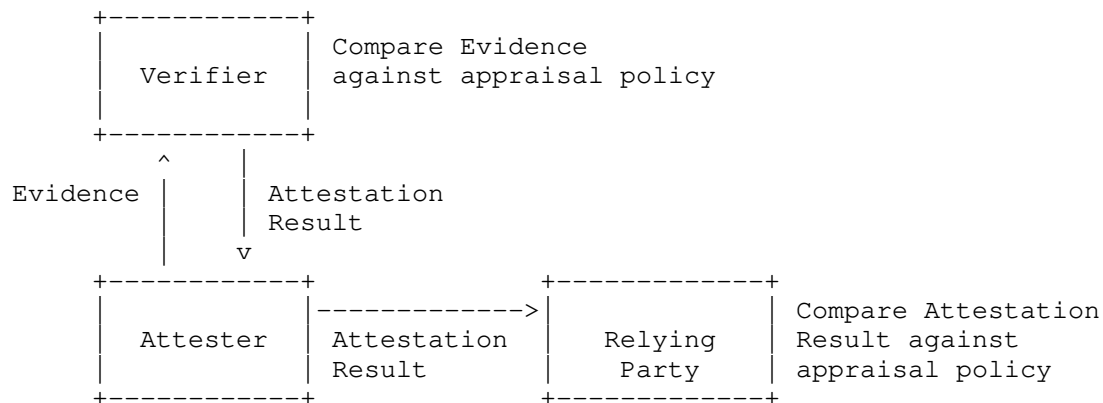


Figure 5: Passport Model

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 simply passes it on to a Verifier. The Verifier then 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.

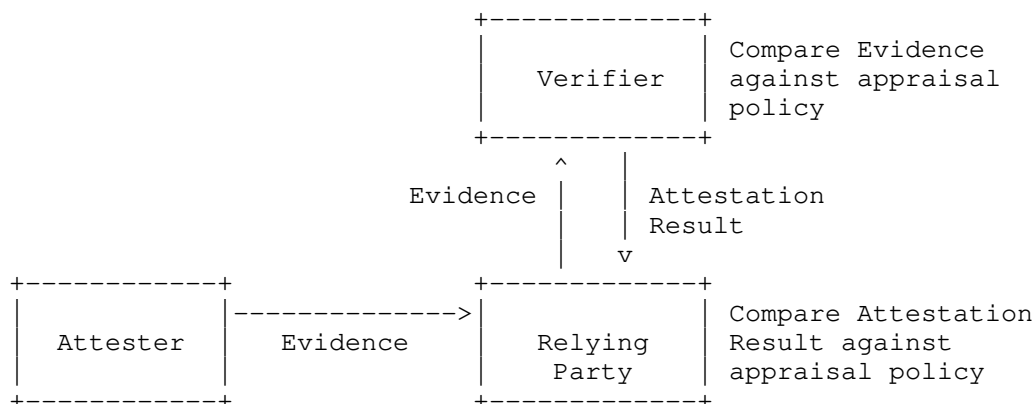


Figure 6: Background-Check Model

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].

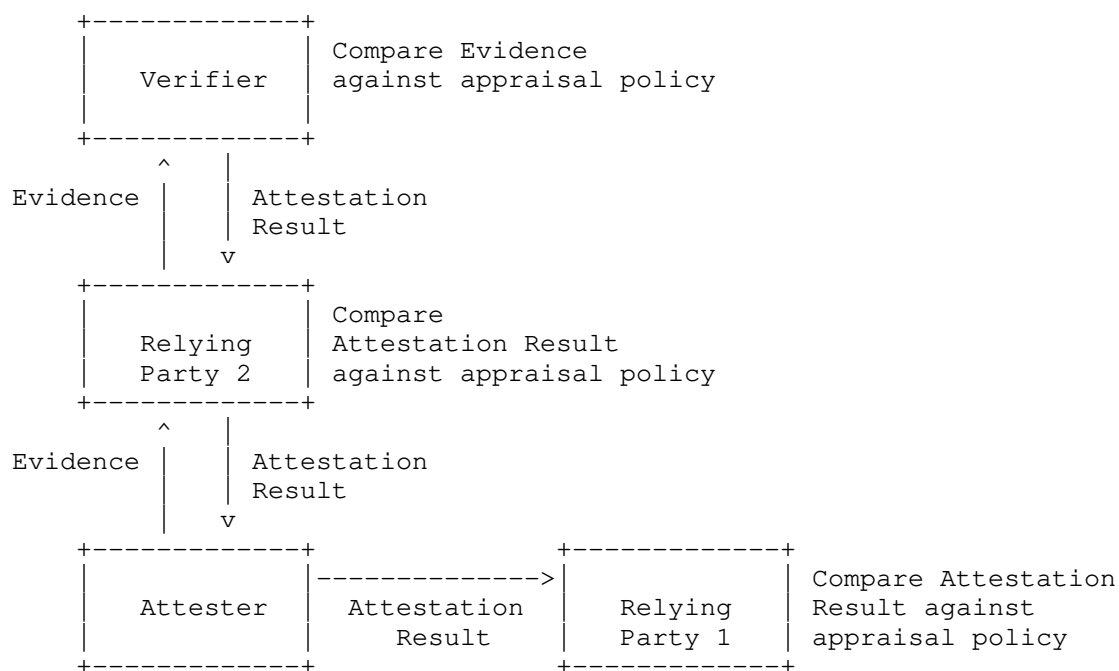


Figure 7: Example Combination

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 might come by the Relying Party trusting the Verifier (or its public key) directly, or might come by trusting an entity (e.g., a Certificate Authority) that is in the Verifier's certificate path. 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.

The Relying Party might implicitly trust a Verifier, such as in a Verifier/Relying Party combination where the Verifier and Relying Party roles are combined. Or, 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.

For example, one explicit way for a Relying Party "A" to establish such trust in 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.

As another example, the Relying Party can establish trust in the Verifier 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.

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.

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, in which case the Endorser's key material is stored in the Verifier's trust anchor store.

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. The root of trust protects both the conveyance channel key material and the Attesting Environment 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 a 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. 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 man-in-the-middle attackers 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 signing capability. For example, if the 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 often need to include detailed information about the Attester, for use by Relying Parties, much like physical passports and drivers licenses include personal information such as name and date of birth. 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:

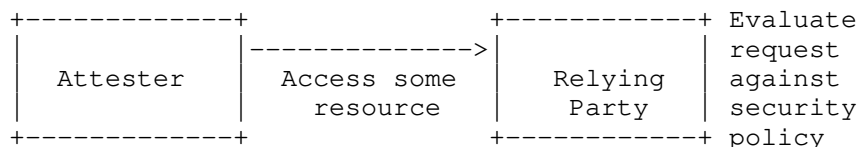


Figure 8: Typical Resource Access

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.

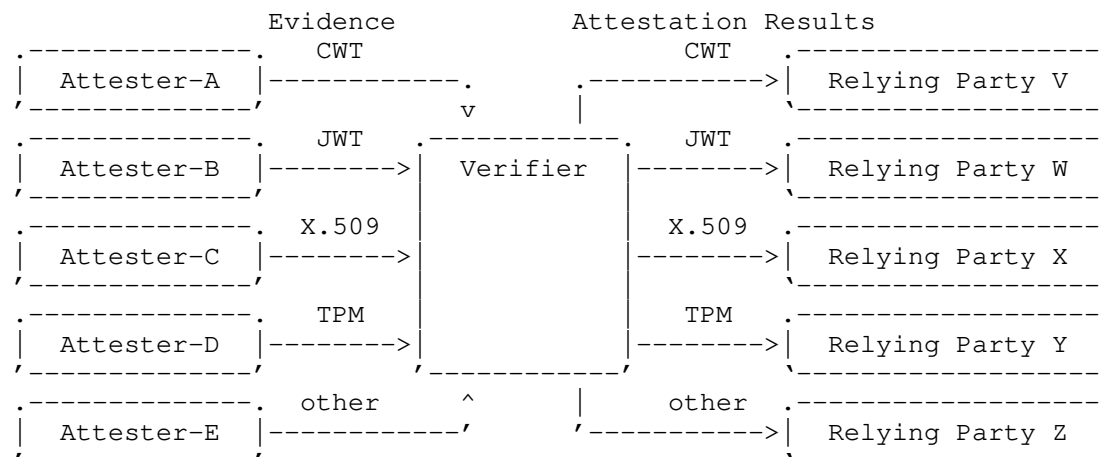


Figure 9: Multiple Attesters and Relying Parties with Different Formats

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 and their values 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.

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 a 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 and are considered trustworthy 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 Section 16.

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.

Many Claims in Evidence and Attestation Results are potentially Personally Identifying Information (PII) depending on the end-to-end use case of the remote attestation procedure. Remote attestation that goes up to include 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.

Evidence and Attestation Results are expected to be integrity protected (i.e., either via signing or a secure channel) and optionally might be confidentiality protected via encryption. If confidentiality protection via signing the conceptual messages is

omitted or unavailable, the protecting protocols that convey Evidence or Attestation Results are responsible for detailing what kinds of information are disclosed, and to whom they are exposed.

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 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).

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. An essential value-add provided by RATS is for the Relying Party to be able to trust the Attester even if the user or owner is not trusted.

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 is procedure is referred to as personalization or customization.

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 device, based upon considerations as to a cost/benefit evaluation of the intended function of the device. 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).

So, this is why, in general, a combination of some physical security measures and some cryptographic measures is 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 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. Integrity Protection

Any solution that conveys information used for security purposes, whether such information is in the form of Evidence, Attestation Results, Endorsements, or appraisal policy 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 appraisal policies 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 (see Section 8) 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.

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 for ever, 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. Specifically, a trust anchor store must resist modification against unauthorized insertion, deletion, and modification.

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. See Section 6 of [RFC5280] for details.

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.

16. 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	A nonce is relayed to an Attester by

	relayed	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.

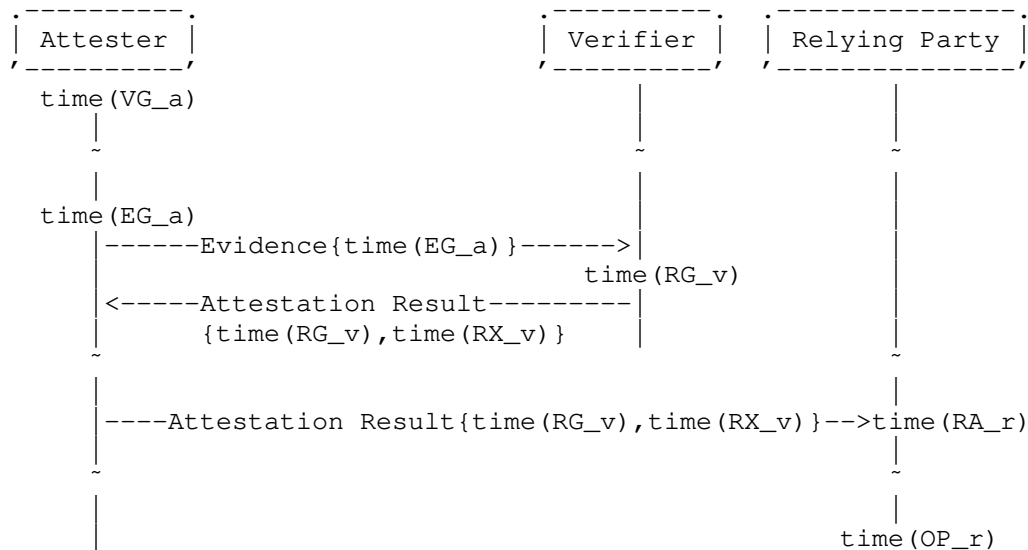
Times with an appended Prime (') indicate a second instance of the same event.

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.

16.1. 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.



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.

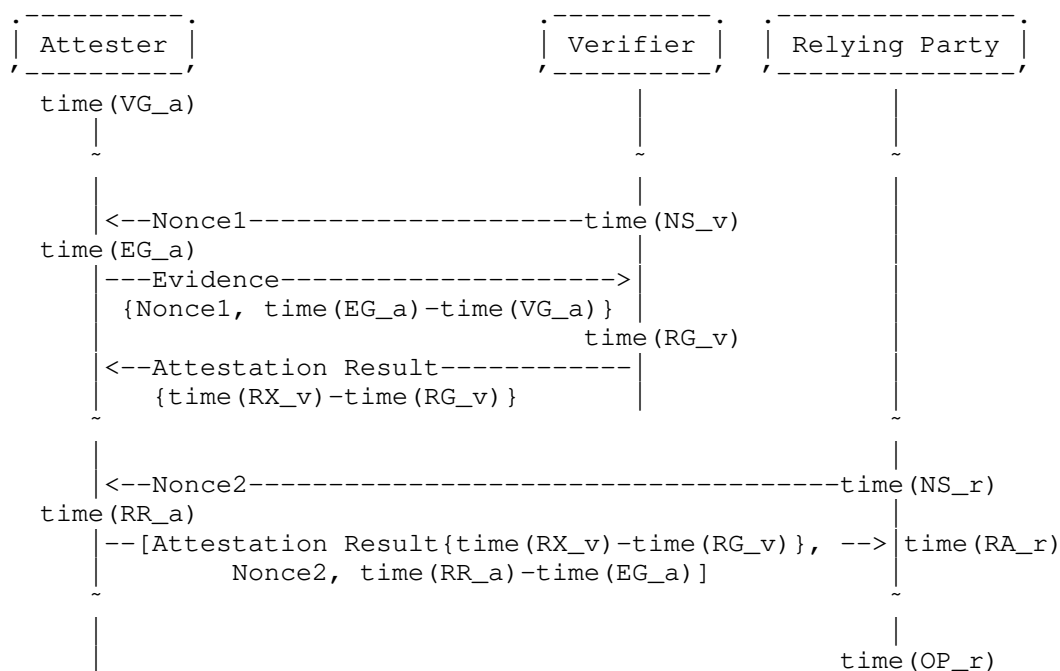
If `time(VG_a)` is also included in the Evidence along with the Claim value generated at that time, and the Verifier decides that it can trust the `time(VG_a)` value, the Verifier can also determine whether the Claim value is recent by checking `"time(RG_v) - time(VG_a) < Threshold"`. The threshold is 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 Relying Party can check whether the Attestation Result is fresh when appraising it at $\text{time}(\text{RA}_r)$ by checking " $\text{time}(\text{RA}_r) - \text{time}(\text{RG}_v) < \text{Threshold}$ ", 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 $\text{time}(\text{RA}_r)$). 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 $\text{time}(\text{OP}_r)$) as long as " $\text{time}(\text{OP}_r) - \text{time}(\text{RG}_v) < \text{Threshold}$ ". However, if the Attestation Result contains an expiry time $\text{time}(\text{RX}_v)$ then it could explicitly check " $\text{time}(\text{OP}_r) < \text{time}(\text{RX}_v)$ ".

16.2. 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.



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)"`.

16.3. 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.

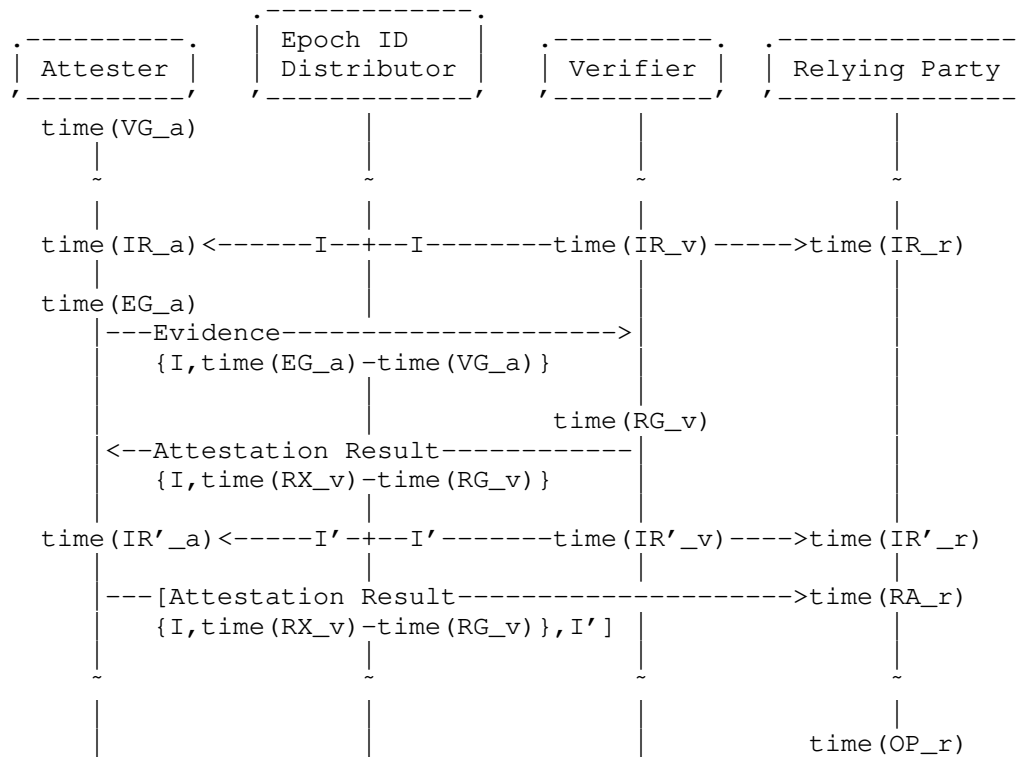
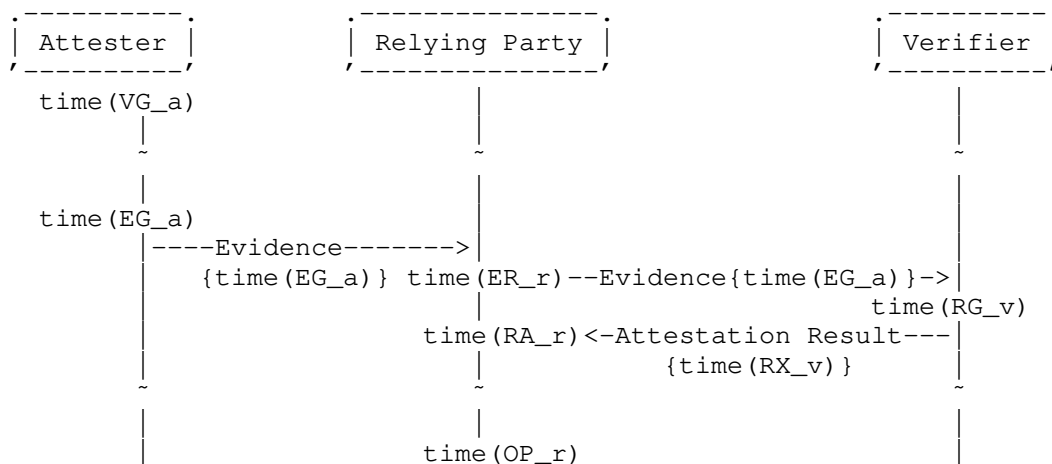


Figure 10: Epoch ID-based Passport Model

16.4. Example 4: Timestamp-based Background-Check Model Example

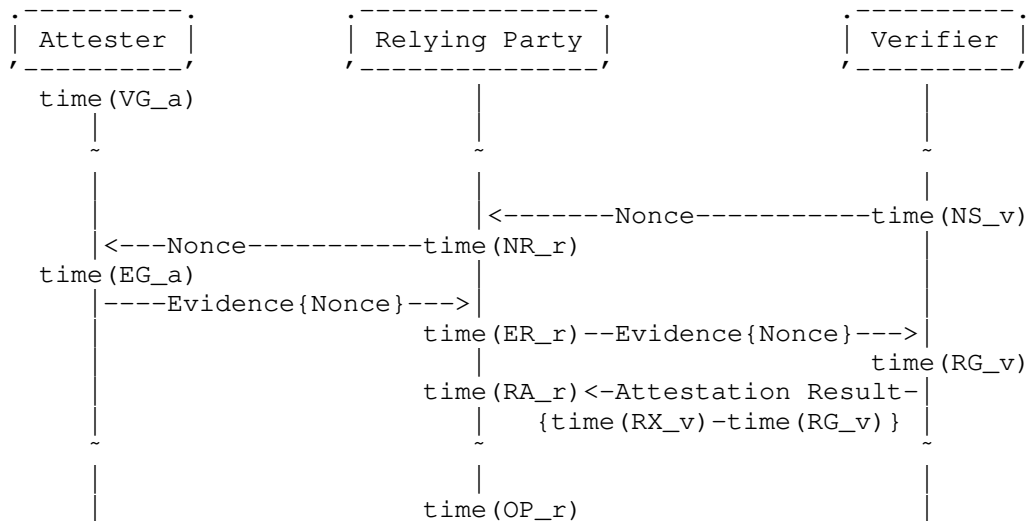
The following example illustrates a hypothetical Background-Check Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party.



The time considerations in this example are equivalent to those discussed under Example 1 above.

16.5. 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.



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)"`.

17. References

17.1. 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>>.

17.2. 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. E., and C. Bormann,
"Time-Based Uni-Directional Attestation", Work in
Progress, Internet-Draft, draft-birkholz-rats-tuda-04, 13
January 2021,
<<https://tools.ietf.org/html/draft-birkholz-rats-tuda-04>>.
- [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://tools.ietf.org/html/draft-birkholz-rats-uccs-03>>.
- [I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler,
"Trusted Execution Environment Provisioning (TEEP)
Architecture", Work in Progress, Internet-Draft, draft-
ietf-teep-architecture-14, 22 February 2021,
<<https://tools.ietf.org/html/draft-ietf-teep-architecture-14>>.
- [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://tools.ietf.org/html/draft-tschofenig-tls-cwt-02>>.
- [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/](https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/)>.
- [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>.

[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/>>.

Contributors

Monty Wiseman

Email: montywiseman32@gmail.com

Liang Xia

Email: frank.xialiang@huawei.com

Laurence Lundblade

Email: lgl@island-resort.com

Eliot Lear

Email: ellear@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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2022

L. Lundblade
Security Theory LLC
G. Mandyam
J. O'Donoghue
Qualcomm Technologies Inc.
October 23, 2021

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-11

Abstract

An Entity Attestation Token (EAT) provides a signed (attested) set of claims that describe state and characteristics of an entity, typically a device like a phone or an IoT device. These claims are used by a Relying Party to determine how much it wishes to trust the entity.

An EAT is either a CWT or JWT with some attestation-oriented claims. To a large degree, all this document does is extend CWT and JWT.

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 April 26, 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	5
1.1. CWT, JWT, UCCS, UJCS and DEB	5
1.2. CDDL, CBOR and JSON	6
1.3. Operating Model and RATS Architecture	7
1.3.1. Use as Attestation Evidence	8
1.3.2. Use as Attestation Results	8
1.4. Entity Overview	9
2. Terminology	9
3. The Claims	10
3.1. Token ID Claim (cti and jti)	11
3.2. Timestamp claim (iat)	11
3.3. Nonce Claim (nonce)	11
3.4. Universal Entity ID Claim (ueid)	12
3.5. Semi-permanent UEIDs (SUEIDs)	14
3.6. Hardware OEM Identification (oemid)	15
3.6.1. Random Number Based	15
3.6.2. IEEE Based	15
3.6.3. IANA Private Enterprise Number	16
3.7. Hardware Version Claims (hardware-version-claims)	16
3.8. Software Name Claim	17
3.9. Software Version Claim	17
3.10. The Security Level Claim (security-level)	17
3.11. Secure Boot Claim (secure-boot)	19
3.12. Debug Status Claim (debug-status)	19
3.12.1. Enabled	20
3.12.2. Disabled	20
3.12.3. Disabled Since Boot	21
3.12.4. Disabled Permanently	21
3.12.5. Disabled Fully and Permanently	21
3.13. Including Keys	21
3.14. The Location Claim (location)	22
3.15. The Uptime Claim (uptime)	23
3.16. The Boot Seed Claim (boot-seed)	23
3.17. The Intended Use Claim (intended-use)	24
3.18. The Profile Claim (profile)	25
3.19. The DLOA (Digital Letter or Approval) Claim (dloas)	26
3.20. The Software Manifests Claim (manifests)	27
3.21. The Software Evidence Claim (swevidence)	28
3.22. The SW Measurement Results Claim (swresults)	29
3.22.1. Scheme	29

3.22.2.	Objective	30
3.22.3.	Results	30
3.22.4.	Objective Name	31
3.23.	Submodules (submods)	33
3.23.1.	Submodule Types	33
3.23.1.1.	Submodule Claims-Set	33
3.23.1.2.	Nested Token	34
3.23.1.3.	Detached Submodule Digest	36
3.23.2.	No Inheritance	37
3.23.3.	Security Levels	37
3.23.4.	Submodule Names	37
3.23.5.	CDDL for submods	38
4.	Unprotected JWT Claims-Sets	38
5.	Detached EAT Bundles	39
6.	Endorsements and Verification Keys	40
6.1.	Identification Methods	41
6.1.1.	COSE/JWS Key ID	41
6.1.2.	JWS and COSE X.509 Header Parameters	41
6.1.3.	CBOR Certificate COSE Header Parameters	42
6.1.4.	Claim-Based Key Identification	42
6.2.	Other Considerations	42
7.	Profiles	42
7.1.	Format of a Profile Document	43
7.2.	List of Profile Issues	43
7.2.1.	Use of JSON, CBOR or both	43
7.2.2.	CBOR Map and Array Encoding	43
7.2.3.	CBOR String Encoding	44
7.2.4.	CBOR Preferred Serialization	44
7.2.5.	COSE/JOSE Protection	44
7.2.6.	COSE/JOSE Algorithms	44
7.2.7.	DEB Support	44
7.2.8.	Verification Key Identification	45
7.2.9.	Endorsement Identification	45
7.2.10.	Freshness	45
7.2.11.	Required Claims	45
7.2.12.	Prohibited Claims	45
7.2.13.	Additional Claims	45
7.2.14.	Refined Claim Definition	45
7.2.15.	CBOR Tags	46
7.2.16.	Manifests and Software Evidence Claims	46
8.	Encoding and Collected CDDL	46
8.1.	Claims-Set and CDDL for CWT and JWT	46
8.2.	Encoding Data Types	47
8.2.1.	Common Data Types	47
8.2.2.	JSON Interoperability	47
8.2.3.	Labels	47
8.3.	CBOR Interoperability	48
8.3.1.	EAT Constrained Device Serialization	48

8.4.	Collected Common CDDL	49
8.5.	Collected CDDL for CBOR	55
8.6.	Collected CDDL for JSON	57
9.	IANA Considerations	59
9.1.	Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries	59
9.2.	Claim Characteristics	59
9.2.1.	Interoperability and Relying Party Orientation	59
9.2.2.	Operating System and Technology Neutral	59
9.2.3.	Security Level Neutral	60
9.2.4.	Reuse of Extant Data Formats	60
9.2.5.	Proprietary Claims	60
9.3.	Claims Registered by This Document	61
9.3.1.	Claims for Early Assignment	61
9.3.2.	To be Assigned Claims	64
9.3.3.	Version Schemes Registered by this Document	64
9.3.4.	UEID URN Registered by this Document	64
9.3.5.	Tag for Detached EAT Bundle	65
10.	Privacy Considerations	65
10.1.	UEID and SUEID Privacy Considerations	65
10.2.	Location Privacy Considerations	66
11.	Security Considerations	66
11.1.	Key Provisioning	66
11.1.1.	Transmission of Key Material	67
11.2.	Transport Security	67
11.3.	Multiple EAT Consumers	67
12.	References	68
12.1.	Normative References	68
12.2.	Informative References	70
Appendix A.	Examples	73
A.1.	Simple TEE Attestation	73
A.2.	EAT Produced by Attestation Hardware Block	74
A.3.	Detached EAT Bundle	75
A.4.	Key / Key Store Attestation	76
A.5.	SW Measurements of an IoT Device	78
A.6.	Attestation Results in JSON format	81
Appendix B.	UEID Design Rationale	82
B.1.	Collision Probability	82
B.2.	No Use of UUID	84
Appendix C.	EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)	85
C.1.	DevID Used With EAT	85
C.2.	How EAT Provides an Equivalent Secure Device Identity	86
C.3.	An X.509 Format EAT	86
C.4.	Device Identifier Permanence	87
Appendix D.	Changes from Previous Drafts	87
D.1.	From draft-rats-eat-01	87
D.2.	From draft-mandyam-rats-eat-00	87

D.3.	From draft-ietf-rats-eat-01	87
D.4.	From draft-ietf-rats-eat-02	88
D.5.	From draft-ietf-rats-eat-03	88
D.6.	From draft-ietf-rats-eat-04	88
D.7.	From draft-ietf-rats-eat-05	89
D.8.	From draft-ietf-rats-eat-06	89
D.9.	From draft-ietf-rats-eat-07	89
D.10.	From draft-ietf-rats-eat-08	89
D.11.	From draft-ietf-rats-eat-09	89
D.12.	From draft-ietf-rats-eat-10	90
Authors' Addresses	91

1. Introduction

Remote device attestation is a fundamental service that allows a remote device such as a mobile phone, an Internet-of-Things (IoT) device, or other endpoint to prove itself to a Relying Party, a server or a service. This allows the Relying Party to know some characteristics about the device and decide whether it trusts the device.

The notion of attestation here is large and may include, but is not limited to the following:

- o Proof of the make and model of the device hardware (HW)
- o Proof of the make and model of the device processor, particularly for security-oriented chips
- o Measurement of the software (SW) running on the device
- o Configuration and state of the device
- o Environmental characteristics of the device such as its GPS location

This document uses the terminology and main operational model defined in [RATS.Architecture]. In particular it is a format that can be used for Attestation Evidence or Attestation Results as defined in the RATS architecture.

1.1. CWT, JWT, UCCS, UJCS and DEB

An EAT is a set of claims about an entity/device based on one of the following:

- o CBOR Web Token (CWT), [RFC8392]

- o Unprotected CWT Claims Sets (UCCS), [UCCS.Draft]
- o JSON Web Token (JWT), [RFC7519]

All definitions, requirements, creation and validation procedures, security considerations, IANA registrations and so on from these carry over to EAT.

This specification extends those specifications by defining additional claims for attestation. This specification also describes the notion of a "profile" that can narrow the definition of an EAT, ensure interoperability and fill in details for specific usage scenarios. This specification also adds some considerations for registration of future EAT-related claims.

The identification of a protocol element as an EAT, whether CBOR or JSON encoded, follows the general conventions used by CWT, JWT and UCCS. Largely this depends on the protocol carrying the EAT. In some cases it may be by content type (e.g., MIME type). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This specification adds two more top-level messages:

- o Unprotected JWT Claims Set (UJCS), Section 4
- o Detached EAT Bundle (DEB), Section 5

A DEB is simple structure to hold a collection of detached claims-sets and the EAT that separately provides integrity and authenticity protection for them. It can be either CBOR or JSON encoded.

1.2. CDDL, CBOR and JSON

An EAT can be encoded in either CBOR or JSON. The definition of each claim is such that it can be encoded either. Each token is either entirely CBOR or JSON, with only an exception for nested tokens.

To implement composite attestation as described in the RATS architecture document, one token has to be nested inside another. It is also possible to construct composite Attestation Results (see below) which may be expressed as one token nested inside another. So as to not force each end-end attestation system to be all JSON or all CBOR, nesting of JSON-encoded tokens in CBOR-encoded tokens and vice versa is accommodated by this specification. This is the only place that CBOR and JSON can be mixed.

This specification formally uses CDDL, [RFC8610], to define each claim. The implementor interprets the CDDL to come to either the CBOR [RFC8949] or JSON [ECMAScript] representation. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 8.2.2 where Appendix E is insufficient.

The CWT and JWT specifications were authored before CDDL was available and did not use CDDL. 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].

The UCCS specification does not include CDDL. This specification provides CDDL for it.

(TODO: The authors are open to modifications to this specification and the UCCS specification to include CDDL for UCCS and UJCS there instead of here.)

1.3. Operating Model and RATS Architecture

While it is not required that EAT be used with the RATS operational model described in Figure 1 in [RATS.Architecture], or even that it be used for attestation, this document is authored with an orientation around that model.

To summarize, an Attester on an entity/device generates Attestation Evidence. Attestation Evidence is a Claims Set describing various characteristics of the entity/device. Attestation Evidence also is usually signed by a key that proves the entity/device and the evidence it produces are authentic. The Claims Set includes a nonce or some other means to provide freshness. EAT is designed to carry Attestation Evidence. The Attestation Evidence goes to a Verifier where the signature is validated. Some of the Claims may also be validated against Reference Values. The Verifier then produces Attestation Results which is also usually a Claims Set. EAT is also designed to carry Attestation Results. The Attestation Results go to the Relying Party which is the ultimate consumer of the "Remote Attestation Procedures", RATS. The Relying Party uses the Attestation Results as needed for the use case, perhaps allowing a device on the network, allowing a financial transaction or such.

Note that sometimes the Verifier and Relying Party are not separate and thus there is no need for a protocol to carry Attestation Results.

1.3.1. Use as Attestation Evidence

Any claim defined in this document or in the IANA CWT or JWT registry may be used in Attestation Evidence.

Attestation Evidence nearly always has to be signed or otherwise have authenticity and integrity protection because the Attester is remote relative to the Verifier. Usually, this is by using COSE/JOSE signing where the signing key is an attestation key provisioned into the entity/device by its manufacturer. The details of how this is achieved are beyond this specification, but see Section 6. If there is already a suitable secure channel between the Attester and Verifier, UCCS may be used.

1.3.2. Use as Attestation Results

Any claim defined in this document or in the IANA CWT or JWT registry may be used in Attestation Results.

It is useful to characterize the relationship of claims in Evidence to those in Attestation Results.

Many claims in Attestation Evidence simply will pass through the Verifier to the Relying Party without modification. They will be verified as authentic from the device by the Verifier just through normal verification of the Attester's signature. The UEID, Section 3.4, and Location, Section 3.14, are examples of claims that may be passed through.

Some claims in Attestation Evidence will be verified by the Verifier by comparison to Reference Values. These claims will not likely be conveyed to the Relying Party. Instead, some claim indicating they were checked may be added to the Attestation Results or it may be tacitly known that the Verifier always does this check. For example, the Verifier receives the Software Evidence claim, Section 3.21, compares it to Reference Values and conveys the results to the Relying Party in a Software Measurement Results Claim, Section 3.22.

In some cases the Verifier may provide privacy-preserving functionality by stripping or modifying claims that do not possess sufficient privacy-preserving characteristics. For example, the data in the Location claim, Section 3.14, may be modified to have a precision of a few kilometers rather than a few meters.

When the Verifier is remote from the Relying Party, the Attestation Results must be protected for integrity, authenticity and possibly confidentiality. Often this will simply be HTTPS as per a normal web

service, but COSE or JOSE may also be used. The details of this protection are beyond the scope of this document.

1.4. Entity Overview

An "entity" can be any device or device subassembly ("submodule") that can generate its own attestation in the form of an EAT. The attestation should be cryptographically verifiable by the EAT consumer. An EAT at the device-level can be composed of several submodule EAT's.

Modern devices such as a mobile phone have many different execution environments operating with different security levels. For example, it is common for a mobile phone to have an "apps" environment that runs an operating system (OS) that hosts a plethora of downloadable apps. It may also have a TEE (Trusted Execution Environment) that is distinct, isolated, and hosts security-oriented functionality like biometric authentication. Additionally, it may have an eSE (embedded Secure Element) - a high security chip with defenses against HW attacks that is used to produce attestations. This device attestation format allows the attested data to be tagged at a security level from which it originates. In general, any discrete execution environment that has an identifiable security level can be considered an entity.

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.

This document reuses terminology from JWT [RFC7519] and CWT [RFC8392].

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.

Claim Value: The value portion of the claim. A claim value can be any CBOR data item or JSON value.

CWT/JWT Claims Set: The CBOR map or JSON object that contains the claims conveyed by the CWT or JWT.

This document reuses terminology from RATS Architecture [RATS.Architecture]

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 Attestation 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].

Attestation Evidence: A Claims Set generated by an Attester to be appraised by a Verifier. Attestation 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 Attestation 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.

3. 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.

This section also describes how several extant CWT and JWT claims apply in EAT.

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 8 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

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.

3.1. Token ID Claim (cti and jti)

CWT defines the "cti" claim. JWT defines the "jti" claim. These are equivalent to each other in EAT and carry a unique token identifier as they do in JWT and CWT. They may be used to defend against re use of the token but are distinct from the nonce that is used by the Relying Party to guarantee freshness and defend against replay.

3.2. Timestamp claim (iat)

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 floating-point for this claim. EAT disallows the use of floating-point. No token may contain an iat claim in float-point format. Any recipient of a token with a floating-point format iat claim may consider it an error. A 64-bit integer representation of epoch time 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.

3.3. Nonce Claim (nonce)

All EATs should have a nonce to prevent replay attacks. The nonce is generated by the Relying Party, the end consumer of the token. It is conveyed to the entity over whatever transport is in use before the token is generated and then included in the token as the nonce claim.

This documents the nonce claim for registration in the IANA CWT claims registry. This is equivalent to the JWT nonce claim that is already registered.

The nonce must be at least 8 bytes (64 bits) as fewer are unlikely to be secure. A maximum of 64 bytes is set to limit the memory a constrained implementation uses. This size range is not set for the

already-registered JWT nonce, but it should follow this size recommendation when used in an EAT.

Multiple nonces are allowed to accommodate multistage verification and consumption.

```
$$claims-set-claims //=  
  (nonce-label => nonce-type / [ 2* nonce-type ])
```

```
nonce-type = bstr .size (8..64)
```

3.4. Universal Entity ID Claim (ueid)

UEID's identify individual manufactured entities / devices such as a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire device or a submodule or subsystem. It does not identify types, models or classes of devices. It is akin to a serial number, though it does not have to be sequential.

UEID's must be universally and globally unique across manufacturers and countries. 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 device to keep devices distinct between manufacturers).

There are privacy considerations for UEID's. See Section 10.1.

The UEID is permanent. It never change for a given device / entity.

UEIDs are variable length. All implementations MUST be able to receive UEIDs that are 33 bytes long (1 type byte and 256 bits). The recommended maximum sent is also 33 bytes.

When the entity constructs the UEID, the first byte is a type and the following bytes the ID for that type. Several types are allowed to accommodate different industries and different manufacturing processes and to give options to avoid paying fees for certain types of manufacturer registrations.

Creation of new types requires a Standards Action [RFC8126].

Type Byte	Type Name	Specification
0x01	RAND	This is a 128, 192 or 256 bit random number generated once and stored in the device. 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 device and stored. It may not be smaller than 128 bits.
0x02	IEEE EUI	This makes use of the IEEE company identification registry. 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-device 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 devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) [IEEE.802-2001], [OUI.Guide]
0x03	IMEI	This is a 14-digit identifier consisting of an 8-digit Type Allocation Code 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. [ThreeGPP.IMEI]

Table 1: UEID Composition Types

UEID's are not designed for direct use by humans (e.g., printing on the case of a device), so no textual representation is defined.

The consumer (the Relying Party) of a UEID MUST treat a UEID as a completely opaque string of bytes and not make any use of its internal structure. For example, they should not use the OUI part of a type 0x02 UEID to identify the manufacturer of the device. Instead they should use the oemid claim that is defined elsewhere. The reasons for this are:

- o UEIDs types may vary freely from one manufacturer to the next.

- o New types of UEIDs may be created. For example, a type 0x07 UEID may be created based on some other manufacturer registration scheme.
- o Device manufacturers are allowed to change from one type of UEID to another anytime they want. For example, they may find they can optimize their manufacturing by switching from type 0x01 to type 0x02 or vice versa. The main requirement on the manufacturer is that UEIDs be universally unique.

A Device Identifier URN is registered for UEIDs. See Section 9.3.4.

```
$$claims-set-claims // = (ueid-label => ueid-type)
```

```
ueid-type = bstr .size (7..33)
```

3.5. Semi-permanent UEIDs (SUEIDs)

An SEUID is of the same format as a UEID, but it may change to a different value on device life-cycle events. Examples of these events are change of ownership, factory reset and on-boarding into an IoT device management system. A device may have both a UEID and SUEIDs, neither, one or the other.

There may be multiple SUEIDs. Each one has a text string label the purpose of which is to distinguish it from others in the token. The label may name the purpose, application or type of the SUEID. Typically, there will be few SUEIDs so there is no need for a formal labeling mechanism like a registry. The EAT profile may describe how SUEIDs should be labeled. If there is only one SUEID, the claim remains a map and there still must be a label. For example, the label for the SUEID used by FIDO Onboarding Protocol could simply be "FIDO".

There are privacy considerations for SUEID's. See Section 10.1.

A Device Identifier URN is registered for SUEIDs. See Section 9.3.4.

```
$$claims-set-claims // = (sueids-label => sueids-type)
```

```
sueids-type = {
  + tstr => ueid-type
}
```

3.6. Hardware OEM Identification (oemid)

This claim identifies the OEM of the hardware. Any of the three forms may be used at the convenience of the attester implementation. The receiver of this claim MUST be able to handle all three forms.

3.6.1. Random Number Based

This format is always 16 bytes in size (128 bits).

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 device they make. This uniquely identifies the OEM on a statistical basis and is large enough should there be ten billion companies.

The OEM may also use a hash like SHA-256 and truncate the output to 128 bits. The input to the hash should be somethings that have at least 96 bits of entropy, but preferably 128 bits of entropy. The input to the hash may be something whose uniqueness is managed by a central registry like a domain name.

This is to be base64url encoded in JSON.

3.6.2. IEEE Based

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 services [OUI.Lookup].

Companies that have more than one of these IDs or MAC address blocks should pick one and prefer that for all their devices.

Commonly, these are expressed in Hexadecimal Representation [IEEE.802-2001] 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. For JSON encoded tokens, this is further base64url encoded.

This format is always 3 bytes in size in CBOR.

3.6.3. IANA Private Enterprise Number

IANA maintains a simple integer-based company registry called the Private Enterprise Number (PEN) [PEN].

PENs are often used to create an OID. That is not the case here. They are used only as a simple integer.

In CBOR this is encoded as a major type 0 integer in CBOR and is typically 3 bytes. It is encoded as a number in JSON.

```
oemid-pen = int
```

```
oemid-ieee = bstr .size 3
```

```
oemid-random = bstr .size 16
```

```
$$claims-set-claims //= (  
    oemid-label =>  
        oemid-random / oemid-ieee / oemid-pen  
)
```

3.7. Hardware Version Claims (hardware-version-claims)

The hardware version can be claimed at three different levels, the chip, the circuit board and the final device assembly. An EAT can include any combination these claims.

The hardware version is a simple 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 [CoSWID].

The hardware version can also be given by a 13-digit [EAN-13]. A new CoSWID version scheme is registered with IANA by this document in Section 9.3.3. An EAN-13 is also known as an International Article Number or most commonly as a bar code.


```
$$claims-set-claims //= (
    chip-version-label => hw-version-type
)

$$claims-set-claims //= (
    board-version-label => hw-version-type
)

$$claims-set-claims //= (
    device-version-label => hw-version-type
)

hw-version-type = [
    version:  tstr,
    scheme:   $version-scheme
]
```

3.8. Software Name Claim

This is a simple free-form text claim for the name of the software. A CoSWID manifest or other type of manifest can be used instead if this is too simple.

```
$$claims-set-claims //= ( sw-name-label => tstr )
```

3.9. Software Version Claim

This makes use of the CoSWID version scheme data type to give a simple version for the software. A full CoSWID manifest or other type of manifest can be instead if this is too simple.

```
$$claims-set-claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version:  tstr,
    scheme:   $version-scheme / As defined by CoSWID /
]
```

3.10. The Security Level Claim (security-level)

This claim characterizes the device/entity ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is by defining four security levels as described below.

These claims describe security environment and countermeasures available on the end-entity/client device where the attestation key resides and the claims originate.

- 1 - Unrestricted: There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise, the EAT provides no meaningful security assurances.
- 2 - Restricted: Entities at this level are not general-purpose operating environments that host features such as app download systems, web browsers and complex productivity applications. It is akin to the secure-restricted level (see below) without the security orientation. Examples include a Wi-Fi subsystem, an IoT camera, or sensor device. Often these can be considered more secure than unrestricted just because they are much simpler and a smaller attack surface, but this won't always be the case. Some unrestricted devices may be implemented in a way that provides poor protection of signing keys.
- 3 - Secure-Restricted: Entities at this level must meet the criteria defined in section 4 of FIDO Allowed Restricted Operating Environments [FIDO.AROE]. Examples include TEE's and schemes using virtualization-based security. Like the FIDO security goal, security at this level is aimed at defending well against large-scale network/remote attacks against the device.
- 4 - Hardware: Entities at this level must include substantial defense against physical or electrical attacks against the device itself. It is assumed any potential attacker has captured the device and can disassemble it. Examples include TPMs and Secure Elements.

The entity should claim the highest security level it achieves and no higher. This set is not extensible so as to provide a common interoperable description of security level to the Relying Party. 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 security and its own proprietary claim as a refined indication.

This claim is not intended as a replacement for a proper end-device security certification scheme such as those based on FIPS 140 [FIPS-140] or those based on Common Criteria [Common.Criteria]. The claim made here is solely a self-claim made by the Attester.

```
$$claims-set-claims //= (
  security-level-label =>
    security-level-cbor-type /
    security-level-json-type
)

security-level-cbor-type = &(
  unrestricted: 1,
  restricted: 2,
  secure-restricted: 3,
  hardware: 4
)

security-level-json-type =
  "unrestricted" /
  "restricted" /
  "secure-restricted" /
  "hardware"
```

3.11. Secure Boot Claim (secure-boot)

The value of true indicates secure boot is enabled. Secure boot is considered enabled when base software, the firmware and operating system, are under control of the entity manufacturer identified in the OEMID claim described in Section 3.6. This may be because the software is in ROM or because it is cryptographically authenticated or some combination of the two or other.

```
$$claims-set-claims //= (secure-boot-label => bool)
```

3.12. Debug Status Claim (debug-status)

This applies to system-wide or submodule-wide debug facilities of the target device / submodule like JTAG and diagnostic hardware built into chips. It applies to any software debug facilities related to root, operating system or privileged software that allow 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-device public keys.

As with all claims, the absence of the debug level claim means it is not reported. A conservative interpretation might assume the Not Disabled state. It could however be that it is reported in a proprietary claim.

This claim is not extensible so as to provide a common interoperable description of debug status to the Relying Party. 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 Verifier or Relying Party receiving an EAT cannot assume that debug is turned off in a submodule because there is a claim indicating it is turned off in a superior module.

An individual target device / submodule 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 Relying Party can know the debug status of the submodules since there is no inheritance.

3.12.1. Enabled

If any debug facility, even manufacturer hardware diagnostics, is currently enabled, then this level must be indicated.

3.12.2. Disabled

This level indicates all debug facilities are currently disabled. It may be possible to enable them in the future, and it may also be possible that they were enabled in the past after the target device/sub-system booted/started, but they are currently disabled.

3.12.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the target device/sub-system booted/started.

3.12.4. Disabled Permanently

This level indicates all non-manufacturer facilities are permanently disabled such that no end user or developer cannot 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.

3.12.5. Disabled Fully and Permanently

This level indicates that all debug capabilities for the target device/sub-module are permanently disabled.

```
$$claims-set-claims // = (
    debug-status-label =>
        debug-status-cbor-type / debug-status-json-type
)

debug-status-cbor-type = &(
    enabled: 0,
    disabled: 1,
    disabled-since-boot: 2,
    disabled-permanently: 3,
    disabled-fully-and-permanently: 4
)

debug-status-json-type =
    "enabled" /
    "disabled" /
    "disabled-since-boot" /
    "disabled-permanently" /
    "disabled-fully-and-permanently"
```

3.13. Including Keys

An EAT may include a cryptographic key such as a public key. The signing of the EAT binds the key to all the other claims in the token.

The purpose for inclusion of the key may vary by use case. For example, the key may be included as part of an IoT device onboarding protocol. When the FIDO protocol includes a public key in its attestation message, the key represents the binding of a user, device

and Relying Party. This document describes how claims containing keys should be defined for the various use cases. It does not define specific claims for specific use cases.

Keys in CBOR format tokens SHOULD be the COSE_Key format [RFC8152] and keys in JSON format tokens SHOULD be the JSON Web Key format [RFC7517]. These two formats support many common key types. Their use avoids the need to decode other serialization formats. These two formats can be extended to support further key types through their IANA registries.

The general confirmation claim format [RFC8747], [RFC7800] may also be used. It provides key encryption. It also allows for inclusion by reference through a key ID. The confirmation claim format may be employed in the definition of some new claim for a particular use case.

When the actual confirmation claim is included in an EAT, this document associates no use case semantics other than proof of possession. Different EAT use cases may choose to associate further semantics. The key in the confirmation claim MUST be protected the same as the key used to sign the EAT. That is, the same, equivalent or better hardware defenses, access controls, key generation and such must be used.

3.14. The Location Claim (location)

The location claim gives the location of the device entity from which the attestation originates. It is derived from the W3C Geolocation API [W3C.GeoLoc]. The latitude, longitude, altitude and accuracy must conform to [WGS84]. The altitude is in meters above the [WGS84] ellipsoid. The two accuracy values are positive numbers in meters. The heading is in degrees relative to true north. If the device is stationary, the heading is NaN (floating-point not-a-number). The speed is the horizontal component of the device velocity in meters per second.

When encoding floating-point numbers half-precision should not be used. It usually does not provide enough precision for a geographic location. It is not a requirement that the receiver of an EAT implement half-precision, so the receiver may not be able to decode the location.

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 GPS 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.

The age data item can be used when the entity doesn't know what time it is either because it doesn't have a clock or it isn't set. The entity must still have a "ticker" that can measure a time interval. The age is the interval between acquisition of the location data and token creation.

See location-related privacy considerations in Section 10.2 below.

```
$$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 = 1 / "latitude"  
longitude = 2 / "longitude"  
altitude = 3 / "altitude"  
accuracy = 4 / "accuracy"  
altitude-accuracy = 5 / "altitude-accuracy"  
heading = 6 / "heading"  
speed = 7 / "speed"  
timestamp = 8 / "timestamp"  
age = 9 / "age"
```

3.15. The Uptime Claim (uptime)

The "uptime" claim contains a value that represents the number of seconds that have elapsed since the entity or submod was last booted.

```
$$claims-set-claims // = (uptime-label => uint)
```

3.16. The Boot Seed Claim (boot-seed)

The Boot Seed claim is a random value created at system boot time that will allow differentiation of reports from different boot sessions. This value is usually public and not protected. It is not the same as a seed for a random number generator which must be kept secret.

`$$claims-set-claims // = (boot-seed-label => bytes)`

3.17. The Intended Use Claim (intended-use)

EAT's may be used in the context of several different applications. The intended-use 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 uses 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 intended-use 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 device security state of the entity prior to provisioning.
- 4 - Certificate Issuance (Certificate Signing Request) Certifying authorities (CA's) may require attestations prior to the issuance of certificates related to keypairs hosted at the entity. 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-cbor-type / intended-use-json-type
)

intended-use-cbor-type = &(
    generic: 1,
    registration: 2,
    provisioning: 3,
    csr: 4,
    pop: 5
)

intended-use-json-type =
    "generic" /
    "registration" /
    "provisioning" /
    "csr" /
    "pop"

```

3.18. The Profile Claim (profile)

See Section 7 for the detailed description of a profile.

A profile is identified by either a URL or an 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 profile claim is to uniquely identify the profile even if it is a private profile.

The OID is encoded in CBOR according to [CBOR.OID] and the URI according to [RFC8949]. Both are unwrapped and thus not tags. The OID is always absolute and never relative. If the claims CBOR type is a text string it is a URI and if a byte string it is an OID.

Note that this named "eat_profile" for JWT and is distinct from the already registered "profile" claim in the JWT claims registry.

```

$$claims-set-claims //= (profile-label => ~uri / ~oid)

```

```

oid = #6.4000(bstr) ; TODO: Replace with CDDL from OID RFC

```

3.19. The DLOA (Digital Letter or Approval) Claim (dloas)

A DLOA (Digital Letter of Approval) [DLOA] is an XML document that describes a certification that a device or 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. When this claim is issued by a Verifier, it MUST be because the entity, device or submodule has received the certification in the DLOA.

This claim can contain more than one DLOA. If multiple DLOAs are present, it MUST be because the entity, device or submodule received all of the certifications.

DLOA XML documents are always fetched from a registrar that stores them. This claim contains several data items used to construct a URL for fetching the DLOA from the particular registrar.

The first data item is a URI for the registrar. The second data item is a platform label to indicate the particular platform that was certified. For platform certifications only these two are needed.

A DLOA may equally apply to an application. In that case it has the URI for the registrar, a platform label and additionally an application label.

The method of combining the registrar URI, platform label and possibly application label is specified in [DLOA].

```
$$claims-set-claims // = (
    dloas-label => [ + dloa-type ]
)
```

```
dloa-type = [
    dloa_registrar: ~uri
    dloa_platform_label: text
    ? dloa_application_label: text
]
```

3.20. The Software Manifests Claim (manifests)

This claim contains descriptions of software that is present on the device. These manifests are installed on the device when the software is installed or are created as part of the installation process. Installation is anything that adds software to the device, possibly factory installation, the user installing elective applications and so on. The defining characteristic is that they are created by the software manufacturer. The purpose of these claims in an EAT is to relay them without modification to the Verifier and/or the Relying Party.

In some cases these will be signed by the software manufacturer independent of any signing for the purpose of EAT attestation. Manifest claims should include the manufacturer's signature (which will be signed over by the attestation signature). In other cases the attestation signature will be the only one.

This claim allows multiple formats for the manifest. For example the manifest may be a CBOR-format CoSWID, an XML-format SWID or other. Identification of the type of manifest is always by a CBOR tag. In many cases, for examples CoSWID, a tag will already be registered with IANA. If not, a tag MUST be registered. It can be in the first-come-first-served space which has minimal requirements for registration.

The claim is an array of one or more manifests. To facilitate hand off of the manifest to a decoding library, each manifest is contained in a byte string. This occurs for CBOR-format manifests as well as non-CBOR format manifests.

If a particular manifest type uses CBOR encoding, then the item in the array for it MUST be a byte string that contains a CBOR tag. The EAT decoder must decode the byte string and then the CBOR within it to find the tag number to identify the type of manifest. The contents of the byte string is then handed to the particular manifest processor for that type of manifest. CoSWID and SUIF manifest are examples of this.

If a particular manifest type does not use CBOR encoding, then the item in the array for it must be a CBOR tag that contains a byte string. The EAT decoder uses the tag to identify the processor for that type of manifest. The contents of the tag, the byte string, are handed to the manifest processor. Note that a byte string is used to contain the manifest whether it is a text based format or not. An example of this is an XML format ISO/IEC 19770 SWID.

It is not possible to describe the above requirements in CDDL so the type for an individual manifest is any in the CDDL below. The above text sets the encoding requirement.

This claim allows for multiple manifests in one token since multiple software packages are likely to be present. The multiple manifests may be of multiple formats. In some cases EAT submodules may be used instead of the array structure in this claim for multiple manifests.

When the [CoSWID] format is used, it MUST be a payload CoSWID, not an evidence CoSWID.

```

$$claims-set-claims /= (
    manifests-label => manifests-type
)

manifests-type = [+ $$manifest-formats]

; Must be a CoSWID payload type
; TODO: signed CoSWIDs
coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>

$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx

; TODO: make this work too
; $$manifest-formats /= bytes .cbor SUIT_Envelope_Tagged

```

3.21. The Software Evidence Claim (swevidence)

This claim contains descriptions, lists, evidence or measurements of the software that exists on the device. The defining characteristic of this claim is that its contents are created by processes on the device that inventory, measure or otherwise characterize the software on the device. The contents of this claim do not originate from the software manufacturer.

In most cases the contents of this claim are signed as part of attestation signing, but independent signing in addition to the attestation signing is not ruled out when a particular evidence format supports it.

This claim uses the same mechanism for identification of the type of the swevidence as is used for the type of the manifest in the manifests claim. It also uses the same byte string based mechanism for containing the claim and easing the hand off to a processing library. See the discussion above in the manifests claim.

When the [CoSWID] format is used, it MUST be evidence CoSWIDs, not payload CoSWIDS.

```
$$claims-set-claims //= (  
    swevidence-label => swevidence-type  
)  
  
swevidence-type = [+ $$swevidence-formats]  
  
; Must be a CoSWID evidence type that is a CBOR tag  
; TODO: fix the CDDL so a signed CoSWID is allowed too  
coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>  
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag
```

3.22. The SW Measurement Results Claim (swresults)

This claim reports the outcome of the comparison of a measurement on some software to the expected Reference Values. It may report a successful comparison, failed comparison or other.

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 swevidence claim to Reference Values.

This claim can also be generated on the device if the device has the ability for one subsystem to measure 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 attestation target or submodule, multiple results can be reported. For example, it may be desirable to report the results for the kernel and each individual application separately.

For each software objective, the following can be reported.

3.22.1. Scheme

This is the free-form text name of the verification system or scheme that performed the verification. There is no official registry of schemes or systems. It may be the name of a commercial product or such.

3.22.2. Objective

This roughly characterizes the coverage of the software measurement software. This corresponds to the attestation target or the submodule. If all of the indicated target is not covered, the measurement must indicate partial.

- 1 - all Indicates all the software has been verified, for example, all the software in the attestation target or the submodule
- 2 - firmware Indicates all of and only the firmware
- 3 - kernel Refers to all of the most-privileged software, for example the Linux kernel
- 4 - privileged Refers to all of the software used by the root, system or administrative account
- 5 - system-libs Refers to all of the system libraries that are broadly shared and used by applications and such
- 6 - partial Some other partial set of the software

3.22.3. Results

This describes the result of the measurement and also the comparison to Reference Values.

- 1 - verification-not-run Indicates no attempt was made to run the verification
- 2 - verification-indeterminate The verification was attempted, but it did not produce a result; perhaps it ran out of memory, the battery died or such
- 3 - verification-failed The verification ran to completion, the comparison was completed and did not compare correctly to the Reference Values
- 4 - fully-verified The verification ran to completion and all measurements compared correctly to Reference Values
- 5 - partially-verified The verification ran to completion and some, but not all measurements compared correctly to Reference Values

3.22.4. Objective Name

This is a free-form text string that describes the objective. For example, "Linux kernel" or "Facebook App"

```
$$claims-set-claims // = (swresults-label => [ + swresult-type ])  
  
verification-result-cbor-type = &(  
    verification-not-run: 1,  
    verification-indeterminate: 2,  
    verification-failed: 3,  
    fully-verified: 4,  
    partially-verified: 5,  
)  
  
verification-result-json-type =  
    "verification-not-run" /  
    "verification-indeterminate" /  
    "verification-failed" /  
    "fully-verified" /  
    "partially-verified"  
  
verification-objective-cbor-type = &(  
    all: 1,  
    firmware: 2,  
    kernel: 3,  
    privileged: 4,  
    system-libs: 5,  
    partial: 6,  
)  
  
verification-objective-json-type =  
    "all" /  
    "firmware" /  
    "kernel" /  
    "privileged" /  
    "system-libs" /  
    "partial"  
  
swresult-type = [  
    verification-system: tstr,  
    objective: verification-objective-cbor-type /  
        verification-objective-json-type,  
    result: verification-result-cbor-type /  
        verification-result-json-type,  
    ? objective-name: tstr  
]
```


3.23. Submodules (submods)

Some devices are complex, having many subsystems. A mobile phone is a good example. It may have several connectivity subsystems for communications (e.g., Wi-Fi and cellular). It may have subsystems for low-power audio and video playback. It may have one or more security-oriented subsystems like a TEE or a Secure Element.

The claims for a subsystem can be grouped together in a submodule or submod.

The submods are in a single map/object, one entry per submodule. There is only one submods map/object in a token. It is identified by its specific label. It is a peer to other claims, but it is not called a claim because it is a container for a claims set rather than an individual claim. This submods part of a token allows what might be called recursion. It allows claims sets inside of claims sets inside of claims sets...

3.23.1. Submodule Types

The following sections define the three major types of submodules:

- o A submodule Claims-Set
- o A nested token, which can be any valid EAT token, CBOR or JSON
- o The digest of a detached Claims-Set

These are distinguished primarily by their data type which may be a map/object, string or array.

3.23.1.1. Submodule Claims-Set

This is simply a subordinate Claims-Set containing claims about the submodule.

The submodule claims-set is produced by the same Attester as the surrounding token. It is secured using the same mechanism as the enclosing token (e.g., it is signed by the same attestation key). It roughly corresponds to an Attester Target Environment as described in the RATS architecture.

It may contain claims that are the same as its surrounding token or superior submodules. For example, the top-level of the token may have a UEID, a submod may have a different UEID and a further subordinate submodule may also have a UEID.

The encoding of a submodule Claims-Set is always the same as the encoding as the token it is part of.

This data type for this type of submodule is a map/object as that is the type of a Claims-Set.

3.23.1.2. Nested Token

This type of submodule is a fully formed complete token. It is typically produced by a separate Attester. It is typically used by a Composite Device as described in RATS Architecture [RATS.Architecture]

In being a submodule of the surrounding token, it is cryptographically bound 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 does NOT need to use the same encoding as the enclosing token. This is to allow Composite Devices to be built without regards to the encoding supported by their Attesters.

Thus a CBOR-encoded token like a CWT or UCCS can have a JWT as a nested token submodule and a JSON-encoded token can have a CWT or UCCS as a nested token submodule.

The data type for this type of submodule is either a text or byte string.

Mechanisms are defined for identifying the encoding and type of the nested token. These mechanisms are different for CBOR and JSON encoding. The type of a CBOR-encoded nested token is identified using the CBOR tagging mechanism and thus is in common with identification used when any CBOR-encoded token is part of a CBOR-based protocol. A new simple type mechanism is defined for indication of the type of a JSON-encoded token since there is no JSON equivalent of tagging.

3.23.1.2.1. Surrounding EAT is CBOR-Encoded

If the submodule is a byte string, then the nested token is CBOR-encoded. The byte string always wraps a token that is a tag. The tag identifies whether the nested token is a CWT, a UCCS or a CBOR-encoded DEB.

If the submodule is a text string, then the nested token is JSON-encoded. The text string contains JSON. That JSON is the exactly

the JSON described in the next section with one exception. The token can't be CBOR-encoded.

```
; This specifies how one fully-formed token is nested inside a
; CBOR-format token. The fully-formed nested token is any valid
; token, CBOR or JSON (JWT, CWT, UCCS, DEB...) The mechanism for
; identifying the type of the nested token is specific to the format
; of the surrounding token, CBOR in this case.
;
; A primary reason this is encoding-specific is that JSON does not
; have an equivalent to CBOR tags.
;
; If the data type here is text, then the nested token is JSON
; format, one of a JWT, UJCS or JSON-encoded DEB. The means for
; distinguishing which is in the definition of JSON-encoded
; Nested-Token. If the data type is bstr, then the nested token
; is CBOR format. It is byte-string wrapped and identified by a
; CBOR tag.
```

```
Nested-Token =
    tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)
    bstr .cbor Tagged-CBOR-Token
```

3.23.1.2.2. Surrounding EAT is JSON-Encoded

A nested token in a JSON-encoded token is an array of two items. The first is a string that indicates the type of the second item as follows:

"JWT" A JWT formatted according to [RFC7519]

"CBOR" Some base64url-encoded CBOR that is a tag that is either a CWT, UCCS or CBOR-encoded DEB

"UJCS" A UJCS-Message. (A UJCS-Message is identical to a JSON-encoded Claims-Set)

"DEB" A JSON-encoded Detached EAT Bundle.

```
; This describes a nested token that occurs inside a JSON-encoded
; token. It uses an array that is made up of a type indicator and the
; actual token. This is a substitute for the CBOR tag mechanism that
; JSON does not have.
```

```
Nested-Token = [
  type : "JWT" / "CBOR" / "UJCS" / "DEB",
  nested-token : JWT-Message /
                  B64URL-Tagged-CBOR-Token /
                  DEB-JSON-Message /
                  UJCS-Message
]
```

```
; This text is a Tagged-CBOR-Token (see cbor-token.cddl) that is
; base64url encoded. For example, it is a CWT that is a COSE_Sign1
; that is a CBOR tag that has been base64url encoded.
```

```
B64URL-Tagged-CBOR-Token = tstr .regexp "[A-Za-z0-9_=-]+"
```

3.23.1.3. Detached Submodule Digest

This is type of submodule equivalent to a Claims-Set submodule, except the Claims-Set is conveyed separately outside of the token.

This type of submodule consists of a digest made using a cryptographic hash of a Claims-Set. The Claims-Set is not included in the token. It is conveyed to the Verifier outside of the token. The submodule containing the digest is called a detached digest. The separately conveyed Claims-Set is called a detached claims set.

The input to the digest is exactly the byte-string wrapped encoded form of the Claims-Set for the submodule. That Claims-Set can include other submodules including nested tokens and detached digests.

The primary use for this is to facilitate the implementation of a small and secure attester, perhaps purely in hardware. This small, secure attester implements COSE signing and only a few claims, perhaps just UEID and hardware identification. It has inputs for digests of submodules, perhaps 32-byte hardware registers. Software running on the device constructs larger claim sets, perhaps very large, encodes them and digests them. The digests are written into the small secure attesters registers. The EAT produced by the small secure attester only contains the UEID, hardware identification and digests and is thus simple enough to be implemented in hardware. Probably, every data item in it is of fixed length.

The integrity protection for the larger Claims Sets will not be as secure as those originating in hardware block, but the key material and hardware-based claims will be. It is possible for the hardware to enforce hardware access control (memory protection) on the digest registers so that some of the larger claims can be more secure. For example, one register may be writable only by the TEE, so the detached claims from the TEE will have TEE-level security.

The data type for this type of submodule is an array. It contains two data items, an algorithm identifier and a byte string containing the digest.

A DEB, described in Section 5, may be used to convey detached claims sets and the token with their detached digests. EAT, however, doesn't require use of a DEB. Any other protocols may be used to convey detached claims sets and the token with their detached digests. Note that since detached Claims-Sets are usually signed, protocols conveying them must make sure they are not modified in transit.

3.23.2. No Inheritance

The subordinate modules do not inherit anything from the containing token. The subordinate modules must explicitly include all of their claims. This is the case even for claims like the nonce.

This rule is in place for simplicity. It avoids complex inheritance rules that might vary from one type of claim to another.

3.23.3. Security Levels

The security level of the non-token subordinate modules should always be less than or equal to that of the containing modules in the case of non-token submodules. It makes no sense for a module of lesser security to be signing claims of a module of higher security. An example of this is a TEE signing claims made by the non-TEE parts (e.g. the high-level OS) of the device.

The opposite may be true for the nested tokens. They usually have their own more secure key material. An example of this is an embedded secure element.

3.23.4. Submodule Names

The label or name for each submodule in the submods map is a text string naming the submodule. No submodules may have the same name.

3.23.5. CDDL for submods

```

; This is the part of a token that contains all the submodules.  It
; is a peer with the claims in the token, but not a claim, only a
; map/object to hold all the submodules.

$$claims-set-claims // = (submods-label => { + text => Submodule })

; A submodule can be:
; - A simple Claims-Set (encoded in the same format as the token)
; - A digest of a detached Claims-Set (encoded in the same format as
;   the token)
; - A nested token which may be either CBOR or JSON format. Further,
;   the mechanism for identifying and containing the nested token
;   depends on the format of the surrounding token, particularly
;   because JSON doesn't have any equivalent of a CBOR tag so a
;   JSON-specific mechanism is invented. Also, there is the issue
;   that binary data must be B64 encoded when carried in
;   JSON. Nested-Token is defined in the format specific CDDL, not
;   here.

; Note that a nested token can either be a signed token like a CWT
; or JWT, an unsigned token like a UCCS or UJCS, or a DEB (detached
; EAT bundle). The specific encoding of these is format-specific
; so it doesn't appear here.

Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest

; This is for both JSON and CBOR.  JSON uses text label for
; algorithm from JOSE registry. CBOR uses integer label for
; algorithm from COSE registry. In JSON the digest is base64
; encoded.

Detached-Submodule-Digest = [
    algorithm : int / text,
    digest : bstr
]
```

4. Unprotected JWT Claims-Sets

This is simply the JSON equivalent of an Unprotected CWT Claims-Set [UCCS.Draft].

It has no protection of its own so protections must be provided by the protocol carrying it. These are extensively discussed in [UCCS.Draft]. All the security discussion and security considerations in [UCCS.Draft] apply to UJCS.

(Note: The EAT author is open to this definition being moved into the UCCS draft, perhaps along with the related CDDL. It is place here for now so that the current UCCS draft plus this document are complete. UJCS is needed for the same use cases that a UCCS is needed. Further, JSON will commonly be used to convey Attestation Results since JSON is common for server to server communications. Server to server communications will often have established security (e.g., TLS) therefore the signing and encryption from JWS and JWE are unnecessary and burdensome).

5. Detached EAT Bundles

A detached EAT bundle is a structure to convey a fully-formed and signed token plus detached claims set that relate to that token. It is a top-level EAT message like a CWT, JWT, UCCS and UJCS. It can be used any place that CWT, JWT, UCCS or UJCS messages are used. It may also be sent as a submodule.

A DEB has two main parts.

The first part is a full top-level token. This top-level token must have at least one submodule that is a detached digest. This top-level token may be either CBOR or JSON-encoded. It may be a CWT, JWT, UCCS or UJCS, but not a DEB. The same mechanism for distinguishing the type for nested token submodules is used here.

The second part is a map/object containing the detached Claims-Sets corresponding to the detached digests in the full token. When the DEB is CBOR-encoded, each Claims-Set is wrapped in a byte string. When the DEB is JSON-encoded, each Claims-Set is base64url encoded. All the detached Claims-Sets MUST be encoded in the same format as the DEB. No mixing of encoding formats is allowed for the Claims-Sets in a DEB.

For CBOR-encoded DEBs, tag TBD602 can be used to identify it. The normal 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.

; Top-level definition of a DEB for CBOR and JSON

```
Detached-EAT-Bundle = [  
  main-token : Nested-Token,  
  detached-claims-sets: {  
    + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set  
  }  
]
```

; text content is a base64url encoded JSON-format Claims-Set

```
json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"
```

```
cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

6. 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 [RATS.Architecture] 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 Reference Values to which EAT claims are compared as part of the verification process. 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.

6.1. 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.

6.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC8152] 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 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.

6.1.2. 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.

6.1.3. 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,...).

6.1.4. 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 device 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.

6.2. Other Considerations

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].

7. Profiles

This EAT specification does not guarantee that implementations of it will interoperate. The variability in this specification is necessary to accommodate the widely varying use cases. An EAT profile narrows the specification for a specific use case. An ideal EAT profile will guarantee interoperability.

The profile can be named in the token using the profile claim described in Section 3.18.

A profile can apply to Attestation Evidence or to Attestation Results or both.

7.1. 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.

In some cases CDDL may be created that replaces CDDL in this or other document to express some profile requirements. For example, to require the altitude data item in the location claim, CDDL can be written that replicates the location claim with the altitude no longer optional.

7.2. List of Profile Issues

The following is a list of EAT, CWT, UCCS, JWS, UJCS, COSE, JOSE and CBOR options that a profile should address.

7.2.1. Use of JSON, CBOR or both

The profile should indicate whether the token format should be CBOR, JSON, both or even some other encoding. If some other encoding, a specification for how the CDDL described here is serialized in that encoding is necessary.

This should be addressed for the top-level token and for any nested tokens. For example, a profile might require all nested tokens to be of the same encoding of the top level token.

7.2.2. CBOR Map and Array Encoding

The profile should indicate whether definite-length arrays/maps, indefinite-length arrays/maps or both are allowed. A good default is to allow only definite-length arrays/maps.

An alternate is to allow both definite and indefinite-length arrays/maps. The decoder should accept either. Encoders that need to fit on very small hardware or be actually implement in hardware can use indefinite-length encoding.

This applies to individual EAT claims, CWT and COSE parts of the implementation.

7.2.3. CBOR String Encoding

The profile should indicate whether definite-length strings, indefinite-length strings or both are allowed. A good default is to allow only definite-length strings. As with map and array encoding, allowing indefinite-length strings can be beneficial for some smaller implementations.

7.2.4. CBOR Preferred Serialization

The profile should indicate whether encoders must use preferred serialization. The profile should indicate whether decoders must accept non-preferred serialization.

7.2.5. COSE/JOSE Protection

COSE and JOSE have several options for signed, MACed and encrypted messages. EAT/CWT has the option to have no protection using UCCS and JOSE has a 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, CWT, UCCS and UJCS are allowed by EAT.

The profile should list the protections that must be supported by all decoders implementing the profile. The encoders then must implement a subset of what is listed for the decoders, perhaps only one.

Implementations may choose to sign or MAC before encryption so that the implementation layer doing the signing or MACing can be the smallest. It is often easier to make smaller implementations more secure, perhaps even implementing in solely in hardware. The key material for a signature or MAC is a private key, while for encryption it is likely to be a public key. The key for encryption requires less protection.

7.2.6. COSE/JOSE Algorithms

The profile document should list the COSE algorithms that a Verifier must implement. The Attester will select one of them. Since there is no negotiation, the Verifier should implement all algorithms listed in the profile. If detached submodules are used, the COSE algorithms allowed for their digests should also be in the profile.

7.2.7. DEB Support

A Detatched EAT Bundle Section 5 is a special case message that will not often be used. A profile may prohibit its use.

7.2.8. Verification Key Identification

Section 6 describes a number of methods for identifying a verification key. The profile document should specify one of these or one that is not described. The ones described in this document are only roughly described. The profile document should go into the full detail.

7.2.9. 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, where as key identification is not.

7.2.10. Freshness

Just about every use case will require some means of knowing the EAT is recent enough and not a replay of an old token. The profile should describe how freshness is achieved. The section on Freshness in [RATS.Architecture] describes some of the possible solutions to achieve this.

7.2.11. Required Claims

The profile can list claims whose absence results in Verification failure.

7.2.12. Prohibited Claims

The profile can list claims whose presence results in Verification failure.

7.2.13. Additional Claims

The profile may describe entirely new claims. These claims can be required or optional.

7.2.14. Refined Claim Definition

The profile may lock down optional aspects of individual claims. For example, it may require altitude in the location claim, or it may require that HW Versions always be described using EAN-13.

7.2.15. CBOR Tags

The profile should specify whether the token should be a CWT Tag or not. Similarly, the profile should specify whether the token should be a UCCS 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.

7.2.16. Manifests and Software Evidence Claims

The profile should specify which formats are allowed for the manifests and software evidence claims. The profile may also go on to say which parts and options of these formats are used, allowed and prohibited.

8. 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 format.

8.1. 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 as well as UCCS. 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, JWT and UCCS. 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, JWT or UCCS in CDDL without it. The CDDL definition of Claims-Set here is applicable to EAT, CWT, JWT and UCCS.

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 encoded in either CBOR or JSON, never a mixture.

CDDL for the seven claims defined by [RFC8392] and [RFC7519] is included here.

8.2. Encoding Data Types

This makes use of the types defined in [RFC8610] Appendix D, Standard Prelude.

8.2.1. Common Data Types

time-int is identical to the epoch-based time, but disallows floating-point representation.

Unless explicitly indicated, URIs are not the URI tag defined in [RFC8949]. They are just text strings that contain a URI.

```
string-or-uri = tstr
```

```
time-int = #6.1(int)
```

8.2.2. JSON Interoperability

JSON should be encoded per [RFC8610] Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

- o bstr - must be base64url encoded
- o time - must be encoded as NumericDate as described section 2 of [RFC7519].
- o string-or-uri - must be encoded as StringOrURI as described section 2 of [RFC7519].
- o uri - must be a URI [RFC3986].
- o oid - encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1").

8.2.3. Labels

Map labels, including Claims-Keys and Claim-Names, and enumerated-type values are always integers when encoding in CBOR and strings when encoding in JSON. There is an exception to this for naming submodules and detached claims sets in a DEB. These are strings in CBOR.

The CDDL in most cases gives both the integer label and the string label as it is not convenient to have conditional CDDL for such.

8.3. CBOR Interoperability

CBOR allows data items to be serialized in more than one form. If the sender uses a form that the receiver can't decode, there will not be interoperability.

This specification gives no blanket requirements to narrow CBOR serialization for all uses of EAT. This allows individual uses to tailor serialization to the environment. It also may result in EAT implementations that don't interoperate.

One way to guarantee interoperability is to clearly specify CBOR serialization in a profile document. See Section 7 for a list of serialization issues that should be addressed.

EAT will be commonly used where the device generating the attestation is constrained and the receiver/Verifier of the attestation is a capacious server. Following is a set of serialization requirements that work well for that use case and are guaranteed to interoperate. Use of this serialization is recommended where possible, but not required. An EAT profile may just reference the following section rather than spell out serialization details.

8.3.1. EAT Constrained Device Serialization

- o Preferred serialization described in section 4.1 of [RFC8949] is not required. The EAT decoder must accept all forms of number serialization. The EAT encoder may use any form it wishes.
- o The EAT decoder must accept indefinite length arrays and maps as described in section 3.2.2 of [RFC8949]. The EAT encoder may use indefinite length arrays and maps if it wishes.
- o The EAT decoder must accept indefinite length strings as described in section 3.2.3 of [RFC8949]. The EAT encoder may use indefinite length strings if it wishes.
- o Sorting of maps by key is not required. The EAT decoder must not rely on sorting.
- o Deterministic encoding described in Section 4.2 of [RFC8949] is not required.
- o Basic validity described in section 5.3.1 of [RFC8949] must be followed. The EAT encoder must not send duplicate map keys/labels or invalid UTF-8 strings.

8.4. Collected Common CDDL

```

; This is the fundamental definition of a Claims-Set for both CBOR
; and JSON. It is a set of label-value pairs each of which is a
; claim.
;
; In CBOR the labels can be integers or strings with a strong
; preference for integers. For JSON, the labels are always strings.
;
; The values can be anything, with some consideration for types that
; can work in both CBOR and JSON.

```

```

Claims-Set = {
    * $$claims-set-claims,
    * Claim-Label .feature "extended-label" => any
}

```

```

Claim-Label = int / text
string-or-uri = tstr

```

```

time-int = #6.1(int)
; This is CDDL for the 7 individual claims that are defined in CWT
; and JWT. This CDDL works for either CBOR format CWT or JSON format
; JWT The integer format CWT Claim Keys (the labels) are defined in
; cwt-labels.cddl. The string format JWT Claim Names (the labels)
; are defined in jwt-labels.cddl.

```

```

; $$claims-set-claims is defined in claims-set.cddl

```

```

$$claims-set-claims //= (iss-label => text)
$$claims-set-claims //= (sub-label => text)
$$claims-set-claims //= (aud-label => text)
$$claims-set-claims //= (exp-label => ~time)
$$claims-set-claims //= (nbf-label => ~time)
$$claims-set-claims //= (iat-label => ~time)

```

```

; TODO: how does the bstr get handled in JSON validation with the
; cddl tool? TODO: should this be a text for JSON?
; $$claims-set-claims //= (cti-label : bytes)
$$claims-set-claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

```

```

nonce-type = bstr .size (8..64)

```

```

$$claims-set-claims //= (ueid-label => ueid-type)

```

```

ueid-type = bstr .size (7..33)
$$claims-set-claims //= (sueids-label => sueids-type)

```

```
sueids-type = {
    + tstr => ueid-type
}

oemid-pen = int

oemid-ieee = bstr .size 3

oemid-random = bstr .size 16

$$claims-set-claims //= (
    oemid-label =>
        oemid-random / oemid-ieee / oemid-pen
)

$$claims-set-claims //= (
    chip-version-label => hw-version-type
)

$$claims-set-claims //= (
    board-version-label => hw-version-type
)

$$claims-set-claims //= (
    device-version-label => hw-version-type
)

hw-version-type = [
    version:  tstr,
    scheme:   $version-scheme
]

$$claims-set-claims //= ( sw-name-label => tstr )

$$claims-set-claims //= (
    security-level-label =>
        security-level-cbor-type /
        security-level-json-type
)

security-level-cbor-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)

security-level-json-type =
    "unrestricted" /
```

```
    "restricted" /
    "secure-restricted" /
    "hardware"
  $$claims-set-claims // = (secure-boot-label => bool)
  $$claims-set-claims // = (
    debug-status-label =>
      debug-status-cbor-type / debug-status-json-type
  )

debug-status-cbor-type = &(
  enabled: 0,
  disabled: 1,
  disabled-since-boot: 2,
  disabled-permanently: 3,
  disabled-fully-and-permanently: 4
)

debug-status-json-type =
  "enabled" /
  "disabled" /
  "disabled-since-boot" /
  "disabled-permanently" /
  "disabled-fully-and-permanently"
  $$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 = 1 / "latitude"
longitude = 2 / "longitude"
altitude = 3 / "altitude"
accuracy = 4 / "accuracy"
altitude-accuracy = 5 / "altitude-accuracy"
heading = 6 / "heading"
speed = 7 / "speed"
timestamp = 8 / "timestamp"
age = 9 / "age"

  $$claims-set-claims // = (uptime-label => uint)
```

```
$$claims-set-claims // = (boot-seed-label => bytes)
$$claims-set-claims // = (
    intended-use-label =>
        intended-use-cbor-type / intended-use-json-type
)

intended-use-cbor-type = &(
    generic: 1,
    registration: 2,
    provisioning: 3,
    csr: 4,
    pop: 5
)

intended-use-json-type =
    "generic" /
    "registration" /
    "provisioning" /
    "csr" /
    "pop"

$$claims-set-claims // = (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: ~uri
    dloa_platform_label: text
    ? dloa_application_label: text
]

$$claims-set-claims // = (profile-label => ~uri / ~oid)

oid = #6.4000(bstr) ; TODO: Replace with CDDL from OID RFC

$$claims-set-claims // = (
    manifests-label => manifests-type
)

manifests-type = [+ $$manifest-formats]

; Must be a CoSWID payload type
; TODO: signed CoSWIDs
coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>

$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx
```

```
; TODO: make this work too
; $$manifest-formats /= bytes .cbor SUIT_Envelope_Tagged

$$claims-set-claims // = (
    swevidence-label => swevidence-type
)

swevidence-type = [+ $$swevidence-formats]

; Must be a CoSWID evidence type that is a CBOR tag
; TODO: fix the CDDL so a signed CoSWID is allowed too
coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag

$$claims-set-claims // = (swresults-label => [ + swresult-type ])

verification-result-cbor-type = &(
    verification-not-run: 1,
    verification-indeterminate: 2,
    verification-failed: 3,
    fully-verified: 4,
    partially-verified: 5,
)

verification-result-json-type =
    "verification-not-run" /
    "verification-indeterminate" /
    "verification-failed" /
    "fully-verified" /
    "partially-verified"

verification-objective-cbor-type = &(
    all: 1,
    firmware: 2,
    kernel: 3,
    privileged: 4,
    system-libs: 5,
    partial: 6,
)

verification-objective-json-type =
    "all" /
    "firmware" /
    "kernel" /
    "privileged" /
    "system-libs" /
```

"partial"

```
swresult-type = [
  verification-system: tstr,
  objective: verification-objective-cbor-type /
    verification-objective-json-type,
  result: verification-result-cbor-type /
    verification-result-json-type,
  ? objective-name: tstr
]
; This is the part of a token that contains all the submodules. It
; is a peer with the claims in the token, but not a claim, only a
; map/object to hold all the submodules.

$$claims-set-claims // = (submods-label => { + text => Submodule })

; A submodule can be:
; - A simple Claims-Set (encoded in the same format as the token)
; - A digest of a detached Claims-Set (encoded in the same format as
;   the token)
; - A nested token which may be either CBOR or JSON format. Further,
;   the mechanism for identifying and containing the nested token
;   depends on the format of the surrounding token, particularly
;   because JSON doesn't have any equivalent of a CBOR tag so a
;   JSON-specific mechanism is invented. Also, there is the issue
;   that binary data must be B64 encoded when carried in
;   JSON. Nested-Token is defined in the format specific CDDL, not
;   here.

; Note that a nested token can either be a signed token like a CWT
; or JWT, an unsigned token like a UCCS or UJCS, or a DEB (detached
; EAT bundle). The specific encoding of these is format-specific
; so it doesn't appear here.

Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest

; This is for both JSON and CBOR. JSON uses text label for
; algorithm from JOSE registry. CBOR uses integer label for
; algorithm from COSE registry. In JSON the digest is base64
; encoded.

Detached-Submodule-Digest = [
  algorithm : int / text,
  digest : bstr
]
```

; Top-level definition of a DEB for CBOR and JSON

```
Detached-EAT-Bundle = [  
  main-token : Nested-Token,  
  detached-claims-sets: {  
    + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set  
  }  
]
```

; text content is a base64url encoded JSON-format Claims-Set

json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"

cbor-wrapped-claims-set = bstr .cbor Claims-Set

8.5. Collected CDDL for CBOR

; The top-level definition of a CBOR-encoded token.

CBOR-Token = Tagged-CBOR-Token / Untagged-CBOR-Token

; All forms of a CBOR-encoded token that are a CBOR tag.

```
Tagged-CBOR-Token = CWT-Tagged-Message  
Tagged-CBOR-Token /= UCCS-Tagged-Message  
Tagged-CBOR-Token /= DEB-Tagged-Message
```

; All forms of a CBOR-encoded token that are not a CBOR tag.

```
Untagged-CBOR-Token = CWT-Untagged-Message  
Untagged-CBOR-Token /= UCCS-Untagged-Message  
Untagged-CBOR-Token /= DEB-Untagged-Message
```

; The payload of the COSE message is always a Claims-Set

```
CWT-Tagged-Message = COSE_Tagged_Message  
CWT-Untagged-Message = COSE_Untagged_Message
```

UCCS-Message = UCCS-Tagged-Message / UCCS-Untagged-Message

UCCS-Tagged-Message = #6.601(UCCS-Untagged-Message)

UCCS-Untagged-Message = Claims-Set

DEB-Tagged-Message = #6.602(DEB-Untagged-Message)

DEB-Untagged-Message = Detached-EAT-Bundle

```
; This specifies how one fully-formed token is nested inside a
; CBOR-format token. The fully-formed nested token is any valid
; token, CBOR or JSON (JWT, CWT, UCCS, DEB...) The mechanism for
; identifying the type of the nested token is specific to the format
; of the surrounding token, CBOR in this case.
```

```
;
; A primary reason this is encoding-specific is that JSON does not
; have an equivalent to CBOR tags.
```

```
;
; If the data type here is text, then the nested token is JSON
; format, one of a JWT, UJCS or JSON-encoded DEB. The means for
; distinguishing which is in the definition of JSON-encoded
; Nested-Token. If the data type is bstr, then the nested token
; is CBOR format. It is byte-string wrapped and identified by a
;CBOR tag.
```

```
Nested-Token =
    tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)
    bstr .cbor Tagged-CBOR-Token
```

```
; This is the CDDL definition of the labels for a CBOR format web
; token, a CWT. The CDDL for the claims is in web-token-claims.cddl
```

```
iss-label = 1
sub-label = 2
aud-label = 3
exp-label = 4
nbf-label = 5
iat-label = 6
cti-label = 7; The following Claim Keys (labels) are pre-assigned by IANA.
; They are for CBOR-based tokens (CWT and UCCS).
; They are not expected to change in the final publication as an RFC.
```

```
nonce-label = 10
ueid-label = 11
oemid-label = 13
security-level-label = 14
```



```
secure-boot-label = 15
debug-status-label = 16
location-label = 17
profile-label = 18
submods-label = 20
```

```
; These are not yet assigned in any way and may change.
; These are intentionally above 24 so as to not use up
; single-byte labels.
```

```
sueids-label = <TBD25>
chip-version-label = <TBD26>
board-version-label = <TBD27>
device-version-label = <TBD28>
sw-name-label = <TBD29>
sw-version-label = <TBD30>
uptime-label = <TBD31>
boot-seed-label = <TBD32>
intended-use-label = <TBD33>
dloas-label = <TBD34>
manifests-label = <TBD35>
swevidence-label = <TBD36>
swresults-label = <TBD37>
```

8.6. Collected CDDL for JSON

```
; 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_=-]+
```

```
; This defines the JSON equivalent of a UCCS message, a token with
; no integrity or authenticity protection.
```

```
UJCS-Message = Claims-Set
```

```
; This describes a nested token that occurs inside a JSON-encoded
; token. It uses an array that is made up of a type indicator and the
; actual token. This is a substitute for the CBOR tag mechanism that
; JSON does not have.
```

```
Nested-Token = [
  type : "JWT" / "CBOR" / "UJCS" / "DEB",
  nested-token : JWT-Message /
```

```

        B64URL-Tagged-CBOR-Token /
        DEB-JSON-Message /
        UJCS-Message
    ]

```

; This text is a Tagged-CBOR-Token (see cbor-token.cddl) that is
; base64url encoded. For example, it is a CWT that is a COSE_Sign1
; that is a CBOR tag that has been base64url encoded.

B64URL-Tagged-CBOR-Token = tstr .regex "[A-Za-z0-9_=-]+"
; This is the CDDL definition of the labels for a JSON format web
; token, a JWT. The CDDL for the claims is in web-token-claims.cddl

```

iss-label = "iss"
sub-label = "sub"
aud-label = "aud"
exp-label = "exp"
nbf-label = "nbf"
iat-label = "iat"
cti-label = "cti"; The following are claim names for JSON encoded tokens.

```

```

ueid-label /= "ueid"
sueids-label /= "sueids"
nonce-label /= "nonce"
oemid-label /= "oemid"
security-level-label /= "secllevel"
secure-boot-label /= "secboot"
debug-status-label /= "dbgstat"
location-label /= "location"
uptime-label /= "uptime"
profile-label /= "eat-profile"
intended-use-label /= "intuse"
boot-seed-label /= "bootseed"
submods-label /= "submods"
timestamp /= "timestamp"
manifests-label /= "manifests"
swevidence-label /= "swevidence"
dloas-label /= "dloas"
swresults-label /= "swresults"
sw-name-label /= "swname"
sw-version-label /= "swversion"

```

```

latitude /= "lat"
longitude /= "long"
altitude /= "alt"
accuracy /= "accry"
altitude-accuracy /= "alt-accry"

```

```
heading /= "heading"  
speed /= "speed"
```

9. IANA Considerations

9.1. 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.

9.2. Claim Characteristics

The following is design guidance for creating new EAT claims, particularly those to be registered with IANA.

Much of this guidance is generic and could also be considered when designing new CWT or JWT claims.

9.2.1. 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.

9.2.2. 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 SW 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.

9.2.3. 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.

9.2.4. 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 format 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.

9.2.5. Proprietary Claims

EAT allows the definition and use of proprietary claims.

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.

9.3. 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 (except for the nonce claim which is already registered for JWT, but not registered for CWT).

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.

9.3.1. Claims for Early Assignment

RFC Editor: in the final publication this section should be combined with the following section as it will no longer be necessary to distinguish claims with early assignment. Also, the following paragraph should be removed.

The claims in this section have been (requested for / given) early assignment according to [RFC7120]. They have been assigned values and registered before final publication of this document. While their semantics is not expected to change in final publication, it is possible that they will. The JWT Claim Names and CWT Claim Keys are not expected to change.

- o Claim Name: Nonce
- o Claim Description: Nonce
- o JWT Claim Name: "nonce" (already registered for JWT)
- o Claim Key: 10
- o Claim Value Type(s): byte string
- o Change Controller: IESG

- o Specification Document(s): [OpenIDConnectCore], *this document*
- o Claim Name: UEID
- o Claim Description: The Universal Entity ID
- o JWT Claim Name: "ueid"
- o CWT Claim Key: 11
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: OEMID
- o Claim Description: IEEE-based OEM ID
- o JWT Claim Name: "oemid"
- o Claim Key: 13
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Security Level
- o Claim Description: Characterization of the security of an Attester or submodule
- o JWT Claim Name: "secllevel"
- o Claim Key: 14
- o Claim Value Type(s): integer
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Secure Boot
- o Claim Description: Indicate whether the boot was secure

- o JWT Claim Name: "secboot"
- o Claim Key: 15
- o Claim Value Type(s): Boolean
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Debug Status
- o Claim Description: Indicate status of debug facilities
- o JWT Claim Name: "dbgstat"
- o Claim Key: 16
- o Claim Value Type(s): integer
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Location
- o Claim Description: The geographic location
- o JWT Claim Name: "location"
- o Claim Key: 17
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Profile
- o Claim Description: Indicates the EAT profile followed
- o JWT Claim Name: "eat_profile"
- o Claim Key: 18
- o Claim Value Type(s): map

- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Submodules Section
- o Claim Description: The section containing submodules (not actually a claim)
- o JWT Claim Name: "submods"
- o Claim Key: 20
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*

9.3.2. To be Assigned Claims

TODO: add the rest of the claims in here

9.3.3. Version Schemes Registered by this Document

IANA is requested to register a new value in the "Software Tag Version Scheme Values" established by [CoSWID].

The new value is a version scheme a 13-digit European Article Number [EAN-13]. An EAN-13 is also known as an International Article Number or most commonly as a bar code. This version scheme is the ASCII text representation of EAN-13 digits, the same ones often printed with a bar code. This version scheme must comply with the EAN allocation and assignment rules. For example, this requires the manufacturer to obtain a manufacture code from GS1.

Index	Version Scheme Name	Specification
5	ean-13	This document

9.3.4. 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

9.3.5. Tag for Detached EAT Bundle

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

Tag	Data Items	Semantics
TBD602	array	Detached EAT Bundle Section 5

10. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity and therefore, implementations should consider providing privacy-preserving options dependent on the intended usage of the EAT. Examples would include suppression of location claims for EAT's provided to unauthenticated consumers.

10.1. UEID and SUEID Privacy Considerations

A UEID is usually not privacy-preserving. Any set of Relying Parties that receives tokens that happen to be from a single device will be able to know the tokens are all from the same device and be able to track the device. Thus, in many usage situations UEID violates governmental privacy regulation. 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:

- o The device obtains explicit permission from the user of the device 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.

- o The UEID/SUEID is used only in a particular context or particular use case. It is used only by one Relying Party.
- o The device 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 device, then the device generates a UEID just for that Relying Party by hashing a proofed Relying Party ID with the main device UEID/SUEID.

Note that some of these privacy preservation strategies result in multiple UEIDs and SUEIDs per device. Each UEID/SUEID is used in a different context, use case or system on the device. However, from the view of the Relying Party, there is just one UEID and it is still globally universal across manufacturers.

10.2. 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 device generating the attestation. For example, many mobile phones prompt the user for permission when before sending location data.

11. 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. In addition, implementors should consider the following.

11.1. 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.

11.1.1. 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.

11.2. Transport Security

As stated in Section 8 of [RFC8392], "The security of the CWT relies upon on the protections offered by COSE". Similar considerations apply to EAT when sent as a CWT. However, EAT introduces the concept of a nonce to protect against replay. Since an EAT may be created by an entity that may not support the same type of transport security as the consumer of the EAT, intermediaries may be required to bridge communications between the entity and consumer. As a result, it is RECOMMENDED that both the consumer create a nonce, and the entity leverage the nonce along with COSE mechanisms for encryption and/or signing to create the EAT.

Similar considerations apply to the use of EAT as a JWT. Although the security of a JWT leverages the JSON Web Encryption (JWE) and JSON Web Signature (JWS) specifications, it is still recommended to make use of the EAT nonce.

11.3. 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 should leverage a secure protocol (e.g. one that uses transport-layer security, i.e. TLS),

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 transport 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.

12. References

12.1. Normative References

[CBOR.OID]

Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", draft-ietf-cbor-tags-oid-08 (work in progress), May 2021.

[CoSWID]

Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-19 (work in progress), October 2021.

[DLOA]

"Digital Letter of Approval", November 2015, <https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf>.

[EAN-13]

GS1, "International Article Number - EAN/UPC barcodes", 2019, <<https://www.gs1.org/standards/barcodes/ean-upc>>.

[FIDO.AROE]

The FIDO Alliance, "FIDO Authenticator Allowed Restricted Operating Environments List", November 2020, <<https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-allowed-restricted-operating-environments-list-v1.2-fd-20201102.html>>.

[IANA.cbor-tags]

"IANA CBOR Tags Registry", n.d., <<https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>>.

[IANA.CWT.Claims]

IANA, "CBOR Web Token (CWT) Claims", <[http://www.iana.org/assignments/cwt](https://www.iana.org/assignments/cwt)>.

- [IANA.JWT.Claims]
IANA, "JSON Web Token (JWT) Claims",
<<https://www.iana.org/assignments/jwt>>.
- [OpenIDConnectCore]
Sakimura, N., Bradley, J., Jones, M., Medeiros, B. D., and
C. Mortimore, "OpenID Connect Core 1.0 incorporating
errata set 1", November 2014,
<https://openid.net/specs/openid-connect-core-1_0.html>.
- [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>>.
- [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>>.
- [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>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517,
DOI 10.17487/RFC7517, May 2015,
<<https://www.rfc-editor.org/info/rfc7517>>.
- [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>>.
- [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-
Possession Key Semantics for JSON Web Tokens (JWTs)",
RFC 7800, DOI 10.17487/RFC7800, April 2016,
<<https://www.rfc-editor.org/info/rfc7800>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for
Writing an IANA Considerations Section in RFCs", BCP 26,
RFC 8126, DOI 10.17487/RFC8126, June 2017,
<<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)",
RFC 8152, DOI 10.17487/RFC8152, July 2017,
<<https://www.rfc-editor.org/info/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/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/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>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [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>>.
- [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>>.
- [UCCS.Draft]
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", draft-ietf-rats-uccs-01 (work in progress), July 2021.
- [WGS84] National Geospatial-Intelligence Agency (NGA), "WORLD GEODETIC SYSTEM 1984, NGA.STND.0036_1.0.0_WGS84", July 2014, <<https://earth-info.nga.mil/php/download.php?file=coord-wgs84>>.

12.2. Informative References

- [BirthdayAttack]
"Birthday attack", <https://en.wikipedia.org/wiki/Birthday_attack>.

- [CBOR.Cert.Draft]
Mattsson, J. P., Selander, G., Raza, S., Hoeglund, J., and
M. Furuheid, "CBOR Encoded X.509 Certificates (C509
Certificates)", draft-ietf-cose-chor-encoded-cert-02 (work
in progress), July 2021.
- [Common.Criteria]
"Common Criteria for Information Technology Security
Evaluation", April 2017,
<<https://www.commoncriteriaportal.org/cc/>>.
- [COSE.X509.Draft]
Schaad, J., "CBOR Object Signing and Encryption (COSE):
Header parameters for carrying and referencing X.509
certificates", draft-ietf-cose-x509-08 (work in progress),
December 2020.
- [ECMAScript]
"Ecma International, "ECMAScript Language Specification,
5.1 Edition", ECMA Standard 262", June 2011,
<<http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf>>.
- [FIPS-140]
National Institute of Standards, "Security Requirements for
Cryptographic Modules", May 2001,
<<https://csrc.nist.gov/publications/detail/fips/140/2/final>>.
- [IEEE.802-2001]
"IEEE Standard For Local And Metropolitan Area Networks
Overview And Architecture", 2007,
<<https://webstore.ansi.org/standards/ieee/ieee8022001r2007>>.
- [IEEE.802.1AR]
"IEEE Standard, "IEEE 802.1AR Secure Device Identifier",
December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.
- [IEEE.RA] "IEEE Registration Authority",
<<https://standards.ieee.org/products-services/regauth/index.html>>.

[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>>.

[RATS.Architecture]

Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-12 (work in progress), April 2021.

[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>>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <<https://www.rfc-editor.org/info/rfc7120>>.

[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>>.

[W3C.GeoLoc]

Worldwide Web Consortium, "Geolocation API Specification 2nd Edition", January 2018, <https://www.w3.org/TR/geolocation-API/#coordinates_interface>.

Appendix A. Examples

These examples are either UCCS, shown as CBOR diagnostic, or UJCS messages. Full CWT and JWT examples with signing and encryption are not given.

All UCCS examples can be the payload of a CWT. To do so, they must be converted from the UCCS message to a Claims-Set, which is achieved by "removing" the tag.

UJCS messages can be directly used as the payload of a JWT.

WARNING: These examples use tag and label numbers not yet assigned by IANA.

A.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 a UCCS EAT that describes a simple TEE. /

```
601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 14: 3, / secure-restricted /
  / secure-boot /    15: true,
  / debug-status /   16: 2, / disabled-since-boot /
  / manifests /      35: [
                        / This is byte-string wrapped /
                        / payload CoSWID. It gives the TEE /
                        / software name, the version and /
                        / the name of the file it is in. /
                        h' da53574944a60064336132340c01016b
                        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.2. 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). /

```

```

601({
  / nonce /                10: h'948f8860d13a463e',
  / UEID /                 11: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID /                13: 64242, / Private Enterprise Number /
  / security-level /       14: 4, / hardware level security /
  / secure-boot /          15: true,
  / debug-status /         16: 3, / disabled-permanently /
  / chip-version /         26: [ "3.1", 1 ] / Type is multipartnumeric /
})

```

A.3. Detached EAT Bundle

In this DEB 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 DEB itself can be assembled by untrusted SW.

/ This is a detached EAT bundle (DEB) tag. /

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. /

/ /
 / This token here is in UCCS format (unsigned). In a more /
 / realistic example, it would be a signed CWT. /

h'd90259a80a48948f8860d13a463e0b500198f50a4ff6c058
 61c8860d13a638ea0d19faf20e040ff51003181a8263332e
 310114a163544545822f5820e5cf95fd24fab71446742dd5
 8d43dae178e55fe2b94291a9291082ffc2635a0b',

{
 / A CBOR-encoded byte-string wrapped EAT claims-set. It /
 / contains claims suitable for a TEE /
 "TEE" : h'a50a48948f8860d13a463e0e030ff51002182381
 585dda53574944a60064336132340c01016b4163
 6d6520544545204f530d65332e312e340282a218
 1f6b41636d6520544545204f53182101a2181f6b
 41636d6520544545204f5318210206a111a11818
 6e61636d655f7465655f332e657865'

}

)

```

/ 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 /

601({
  / nonce / 10: h'948f8860d13a463e',
  / UEID / 11: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID / 13: 64242, / Private Enterprise Number /
  / security-level / 14: 4, / hardware level security /
  / secure-boot / 15: true,
  / debug-status / 16: 3, / disabled-permanently /
  / chip-version / 26: [ "3.1", 1 ], / multipartnumeric /
  / submods/ 20: {
    "TEE": [ / detached digest submod /
      -16, / SHA-256 /
      h'e5cf95fd24fab7144674
      2dd58d43dae178e55fe2
      b94291a9291082ffc2635
      a0b'
    ]
  }
})

```

A.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, for example a /
/ 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. /
/ /
/ This is similar to Android Key Attestation. /

```

```
601({
```

```

/ nonce /          10: h'948f8860d13a463e',
/ security-level / 14: 3, / secure-restricted /
/ debug-status /   16: 2, / disabled-since-boot /
/ secure-boot /     15: true,
/ manifests /       35: [
                        h'da53574944a600683762623334383766
                        0c000169436172626f6e6974650d6331
                        2e320e0102a2181f75496e6475737472
                        69616c204175746f6d6174696f6e1821
                        02'
                        / Above is an encoded CoSWID      /
                        / with the following data          /
                        /   SW Name: "Carbonite"           /
                        /   SW Vers: "1.2"                 /
                        /   SW Creator:                     /
                        /     "Industrial Automation"       /
                        ],
/ expiration /      4: 1634324274, / 2021-10-15T18:57:54Z /
/ creation time /   6: 1634317080, / 2021-10-15T16:58:00Z /
                        -80000 : "fingerprint",
                        -80001 : { / The key -- A COSE_Key /
/ kty /             1: 2, / EC2, elliptic 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 /         20 : {
                        "HLOS" : { / submod for high-level OS /
/ nonce /           10: h'948f8860d13a463e',
/ security-level / 14: 1, / unrestricted /
/ secure-boot /     15: true,
/ manifests /       35: [
                        h'da53574944a600687337
                        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.5. SW Measurements of an IoT Device

This is a simple token that might be for an IoT device. It includes CoSWID format measurements of the SW. The CoSWID is in byte-string wrapped in the token and also shown in diagnostic form.

```

/ This EAT UCCS 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. /

```

```

601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 14: 3, / secure-restricted /
  / secure-boot /    15: true,
  / debug-status /   16: 2, / disabled-since-boot /
  / OEMID /          13: h'8945ad', / IEEE CID based /
  / UEID /           11: h'0198f50a4ff6c05861c8860d13a638ea',
  / sumods /         20: {
    "OS" : {
      / security-level / 14: 2, / restricted /
      / secure-boot /    15: true,
      / debug-status /   16: 2, / disabled-since-boot /
      / swevidence /     36: [
        / This is a byte-string wrapped /
        / evidence CoSWID. It has /
        / hashes of the main files of /
        / the IoT OS. /
        h'da53574944a600663463613234350c
        17016d41636d6520522d496f542d4f
        530d65332e312e3402a2181f724163
        6d6520426173652041747465737465
        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 fictitious 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.6. Attestation Results in JSON format

This is a UJCS format token 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 security level directly through to the Relying Party. The Verifier also knows the Reference Values for the measured SW components and is able to check them. It informs the Relying Party that they were correct in the swresults claim. "Trustus Verifications" is the name of the services that verifies the SW component measurements.

This UJCS is identical to JSON-encoded Claims-Set that could be a JWT payload.

```
{  
  "nonce" : "lI+IYNE6Rj4=",  
  "secllevel" : "secure-restricted",  
  "secboot" : true,  
  "dbgstat" : "disabled-since-boot",  
  "OEMID" : "iUWt",  
  "UEID" : "AZj1Ck/2wFhhyIYNE6Y4",  
  "submods" : {  
    "secllevel" : "restricted",  
    "secboot" : true,  
    "dbgstat" : "disabled-since-boot",  
    "swname" : "Acme R-IoT-OS",  
    "sw-version" : [  
      "3.1.4"  
    ],  
    "swresults" : [  
      [  
        "Trustus Verifications",  
        "all",  
        "fully-verified"  
      ]  
    ]  
  }  
}
```

Appendix B. UEID Design Rationale

B.1. Collision Probability

This calculation is to determine the probability of a collision of 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})

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 SW 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

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

Clearly, 128 bits is enough for the near future thus the requirement that 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 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.

UUIDs seem to have been designed for scenarios where the implementor does not have full control over the environment and uniqueness has to be constructed from identifiers at hand. UEID takes the view that

hardware, software and/or manufacturing process directly implement UEID in a simple and direct way. It takes the view that cryptographic quality random number generators are readily available as they are implemented in commonly used CPU hardware.

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 are used with an open set of network protocols for authentication and such. In these protocols the DevID secret is used to sign a nonce or similar to proof the association of the DevID certificates with the device.

By contrast, EAT defines network protocol for proving trustworthiness to a Relying Party, the very thing that is not defined in [IEEE.802.1AR]. Nor does not 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 defines a network protocol and [IEEE.802.1AR] doesn't. Vice versa, EAT doesn't define a device implementation and DevID does.

Hence, EAT can be the network protocol 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. Changes from Previous Drafts

The following is a list of known changes from the previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences.

D.1. From draft-rats-eat-01

- o Added UEID design rationale appendix

D.2. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but no new claims have been added.

- o Separate information and data model using CDDL.
- o Say an EAT is a CWT or JWT
- o Use a map to structure the boot_state and location claims

D.3. From draft-ietf-rats-eat-01

- o Clarifications and corrections for OEMID claim
- o Minor spelling and other fixes

- o Add the nonce claim, clarify jti claim
- D.4. From draft-ietf-rats-eat-02
- o Roll all EUIs back into one UEID type
 - o UEIDs can be one of three lengths, 128, 192 and 256.
 - o Added appendix justifying UEID design and size.
 - o Submods part now includes nested eat tokens so they can be named and there can be more than one of them
 - o Lots of fixes to the CDDL
 - o Added security considerations
- D.5. From draft-ietf-rats-eat-03
- o Split boot_state into secure-boot and debug-disable claims
 - o Debug disable is an enumerated type rather than Booleans
- D.6. From draft-ietf-rats-eat-04
- o Change IMEI-based UEIDs to be encoded as a 14-byte string
 - o CDDL cleaned up some more
 - o CDDL allows for JWTs and UCCSs
 - o CWT format submodules are byte string wrapped
 - o Allows for JWT nested in CWT and vice versa
 - o Allows UCCS (unsigned CWTs) and JWT unsecured tokens
 - o Clarify tag usage when nesting tokens
 - o Add section on key inclusion
 - o Add hardware version claims
 - o Collected CDDL is now filled in. Other CDDL corrections.
 - o Rename debug-disable to debug-status; clarify that it is not extensible

- o Security level claim is not extensible
 - o Improve specification of location claim and added a location privacy section
 - o Add intended use claim
- D.7. From draft-ietf-rats-eat-05
- o CDDL format issues resolved
 - o Corrected reference to Location Privacy section
- D.8. From draft-ietf-rats-eat-06
- o Added boot-seed claim
 - o Rework CBOR interoperability section
 - o Added profiles claim and section
- D.9. From draft-ietf-rats-eat-07
- o Filled in IANA and other sections for possible preassignment of Claim Keys for well understood claims
- D.10. From draft-ietf-rats-eat-08
- o Change profile claim to be either a URL or an OID rather than a test string
- D.11. From draft-ietf-rats-eat-09
- o Add SUEIDs
 - o Add appendix comparing IDevID to EAT
 - o Added section on use for Evidence and Attestation Results
 - o Fill in the key ID and endorsements identificaiton section
 - o Remove origination claim as it is replaced by key IDs and endorsements
 - o Added manifests and software evidence claims
 - o Add string labels non-claim labels for use with JSON (e.g. labels for members of location claim)

- o EAN-13 HW versions are no longer a separate claim. Now they are folded in as a CoSWID version scheme.

D.12. From draft-ietf-rats-eat-10

- o Hardware version is made into an array of two rather than two claims
- o Corrections and wording improvements for security levels claim
- o Add swresults claim
- o Add dloas claim - Digital Letter of Approvals, a list of certifications
- o CDDL for each claim no longer in a separate sub section
- o Consistent use of terminology from RATS architecture document
- o Consistent use of terminology from CWT and JWT documents
- o Remove operating model and procedures; refer to CWT, JWT and RATS architecture instead
- o Some reorganization of Section 1
- o Moved a few references, including RATS Architecture, to informative.
- o Add detached submodule digests and detached eat bundles (DEBs)
- o New simpler and more universal scheme for identifying the encoding of a nested token
- o Made clear that CBOR and JSON are only mixed when nesting a token in another token
- o Clearly separate CDDL for JSON and CBOR-specific data items
- o Define UJCS (unsigned JWTs)
- o Add CDDL for a general Claims-Set used by UCCS, UJCS, CWT, JWT and EAT
- o Top level CDDL for CWT correctly refers to COSE
- o OEM ID is specifically for HW, not for SW

- o HW OEM ID can now be a PEN
- o HW OEM ID can now be a 128-bit random number
- o Expand the examples section
- o Add software and version claims as easy / JSON alternative to CoSWID

Authors' Addresses

Laurence Lundblade
Security Theory LLC

EMail: lg1@securitytheory.com

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough GU14 7LS
United Kingdom

Phone: +44 1252 363189
EMail: jodonogh@qti.qualcomm.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2022

L. Lundblade
Security Theory LLC
G. Mandyam
J. O'Donoghue
Qualcomm Technologies Inc.
February 23, 2022

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-12

Abstract

An Entity Attestation Token (EAT) provides an attested claims set that describes state and characteristics of an entity, a device like a phone, IoT device, network equipment or such. This claims set is used by a relying party, server or service to determine how much it wishes to trust the entity.

An EAT is either a CBOR Web Token (CWT) or JSON Web Token (JWT) with attestation-oriented claims. To a large degree, all this document does is extend CWT and JWT.

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 August 27, 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	5
1.1. Entity Overview	6
1.2. CWT, JWT, UCCS, UJCS and DEB	7
1.3. CDDL, CBOR and JSON	8
1.4. Operating Model and RATS Architecture	8
1.4.1. Relationship between Attestation Evidence and Attestation Results	9
2. Terminology	10
3. The Claims	11
3.1. Token ID Claim (cti and jti)	11
3.2. Timestamp claim (iat)	11
3.3. Nonce Claim (nonce)	12
3.4. Universal Entity ID Claim (ueid)	12
3.5. Semi-permanent UEIDs (SUEIDs)	15
3.6. Hardware OEM Identification (oemid)	16
3.6.1. Random Number Based OEMID	16
3.6.2. IEEE Based OEMID	16
3.6.3. IANA Private Enterprise Number Based OEMID	17
3.7. Hardware Model Claim (hardware-model)	17
3.8. Hardware Version Claims (hardware-version-claims)	18
3.9. Software Name Claim	19
3.10. Software Version Claim	19
3.11. The Security Level Claim (security-level)	19
3.12. Secure Boot Claim (secure-boot)	21
3.13. Debug Status Claim (debug-status)	21
3.13.1. Enabled	22
3.13.2. Disabled	22
3.13.3. Disabled Since Boot	22
3.13.4. Disabled Permanently	22
3.13.5. Disabled Fully and Permanently	22
3.14. Including Keys	23
3.15. The Location Claim (location)	24
3.16. The Uptime Claim (uptime)	25
3.17. The Boot Odometer Claim (odometer)	25
3.18. The Boot Seed Claim (boot-seed)	25
3.19. The Intended Use Claim (intended-use)	26
3.20. The Profile Claim (profile)	27
3.21. The DLOA (Digital Letter or Approval) Claim (dloas)	27
3.22. The Software Manifests Claim (manifests)	28

3.23. The Software Evidence Claim (swevidence)	30
3.24. The SW Measurement Results Claim (swresults)	30
3.24.1. Scheme	31
3.24.2. Objective	31
3.24.3. Results	31
3.24.4. Objective Name	32
3.25. Submodules (submods)	34
3.25.1. Submodule Types	34
3.25.1.1. Submodule Claims-Set	34
3.25.1.2. Nested Token	35
3.25.1.3. Detached Submodule Digest	37
3.25.2. No Inheritance	38
3.25.3. Security Levels	38
3.25.4. Submodule Names	39
3.25.5. CDDL for submods	39
4. Unprotected JWT Claims-Sets	39
5. Detached EAT Bundles	39
6. Endorsements and Verification Keys	40
6.1. Identification Methods	41
6.1.1. COSE/JWS Key ID	41
6.1.2. JWS and COSE X.509 Header Parameters	42
6.1.3. CBOR Certificate COSE Header Parameters	42
6.1.4. Claim-Based Key Identification	42
6.2. Other Considerations	42
7. Profiles	43
7.1. Format of a Profile Document	43
7.2. List of Profile Issues	43
7.2.1. Use of JSON, CBOR or both	43
7.2.2. CBOR Map and Array Encoding	44
7.2.3. CBOR String Encoding	44
7.2.4. CBOR Preferred Serialization	44
7.2.5. COSE/JOSE Protection	44
7.2.6. COSE/JOSE Algorithms	45
7.2.7. DEB Support	45
7.2.8. Verification Key Identification	45
7.2.9. Endorsement Identification	45
7.2.10. Freshness	45
7.2.11. Required Claims	45
7.2.12. Prohibited Claims	45
7.2.13. Additional Claims	46
7.2.14. Refined Claim Definition	46
7.2.15. CBOR Tags	46
7.2.16. Manifests and Software Evidence Claims	46
8. Encoding and Collected CDDL	46
8.1. Claims-Set and CDDL for CWT and JWT	46
8.2. Encoding Data Types	47
8.2.1. Common Data Types	47
8.2.2. JSON Interoperability	47

8.2.3. Labels	48
8.3. CBOR Interoperability	48
8.3.1. EAT Constrained Device Serialization	48
8.4. Collected Common CDDL	49
8.5. Collected CDDL for CBOR	54
8.6. Collected CDDL for JSON	55
9. IANA Considerations	56
9.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries	56
9.2. Claim Characteristics	57
9.2.1. Interoperability and Relying Party Orientation . . .	57
9.2.2. Operating System and Technology Neutral	57
9.2.3. Security Level Neutral	58
9.2.4. Reuse of Extant Data Formats	58
9.2.5. Proprietary Claims	58
9.3. Claims Registered by This Document	58
9.3.1. Claims for Early Assignment	59
9.3.2. To be Assigned Claims	62
9.3.3. Version Schemes Registered by this Document	65
9.3.4. UEID URN Registered by this Document	66
9.3.5. Tag for Detached EAT Bundle	66
10. Privacy Considerations	66
10.1. UEID and SUEID Privacy Considerations	67
10.2. Location Privacy Considerations	67
10.3. Replay Protection and Privacy	68
11. Security Considerations	68
11.1. Key Provisioning	68
11.1.1. Transmission of Key Material	69
11.2. Transport Security	69
11.3. Multiple EAT Consumers	69
12. References	70
12.1. Normative References	70
12.2. Informative References	73
Appendix A. Examples	76
A.1. Simple TEE Attestation	76
A.2. Submodules for Board and Device	77
A.3. EAT Produced by Attestation Hardware Block	79
A.4. Detached EAT Bundle	79
A.5. Key / Key Store Attestation	81
A.6. SW Measurements of an IoT Device	83
A.7. Attestation Results in JSON format	86
Appendix B. UEID Design Rationale	87
B.1. Collision Probability	87
B.2. No Use of UUID	89
Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)	90
C.1. DevID Used With EAT	90
C.2. How EAT Provides an Equivalent Secure Device Identity . .	91

C.3. An X.509 Format EAT	91
C.4. Device Identifier Permanence	92
Appendix D. Changes from Previous Drafts	92
D.1. From draft-rats-eat-01	92
D.2. From draft-mandyam-rats-eat-00	92
D.3. From draft-ietf-rats-eat-01	92
D.4. From draft-ietf-rats-eat-02	93
D.5. From draft-ietf-rats-eat-03	93
D.6. From draft-ietf-rats-eat-04	93
D.7. From draft-ietf-rats-eat-05	94
D.8. From draft-ietf-rats-eat-06	94
D.9. From draft-ietf-rats-eat-07	94
D.10. From draft-ietf-rats-eat-08	94
D.11. From draft-ietf-rats-eat-09	94
D.12. From draft-ietf-rats-eat-10	95
D.13. From draft-ietf-rats-eat-11	96
Authors' Addresses	96

1. Introduction

EAT provides the definition of a base set of claims that can be made about an entity, a device, some software and/or some hardware. This claims set is received by a relying party who uses it to decide if and how it will interact with the remote entity. It may choose to not trust the entity and not interact with it. It may choose to trust it. It may partially trust it, for example allowing monetary transactions only up to a limit.

EAT defines the encoding of the claims set in CBOR [RFC8949] and JSON [RFC7159]. EAT is an extension to CBOR Web Token (CWT) [RFC8392] and JSON Web Token (JWT) [RFC7519].

The claims set is secured in transit with the same mechanisms used by CWT and JWT, in particular CBOR Object Signing and Encryption (COSE) [RFC8152] and JSON Object Signing and Encryption (JOSE) [RFC7515] [RFC7516]. Authenticity and integrity protection must always be provided. Privacy (encryption) may additionally be provided. The key material used to sign and encrypt is specifically created and provisioned for the purpose of attestation. It is the use of this key material that make the claims set "attested" rather than just some parameters sent to the relying party by the device.

EAT is focused on authenticating, identifying and characterizing implementations where implementations are devices, chips, hardware, software and such. This is distinct from protocols like TLS [RFC8446] that authenticate and identify servers and services. It is equally distinct from protocols like SASL [RFC4422] that authenticate and identify persons.

The notion of attestation is large, ranging over a broad variety of use cases and security levels. Here are a few examples of claims:

- o Make and model of manufactured consumer device
- o Make and model of a chip or processor, particularly for a security-oriented chip
- o Identification and measurement of the software running on a device
- o Configuration and state of a device
- o Environmental characteristics of a device like its GPS location
- o Formal certifications received

EAT also supports nesting of sets of claims and EAT tokens for use with complex composite devices.

This document uses the terminology and main operational model defined in [RATS.Architecture]. In particular, it can be used for RATS Attestation Evidence and Attestation Results.

1.1. Entity Overview

The document uses the term "entity" to refer to the target of the attestation token. The claims defined in this document are claims about an entity.

An entity is an implementation in hardware, software or both.

An entity is the same as the Attester Target Environment defined in RATS Architecture.

An entity also corresponds to a "system component" as defined in the Internet Security Glossary [RFC4949]. That glossary also defines "entity" and "system entity" as something that may be a person or organization as well as a system component. Here "entity" never refers to a person or organization.

An entity is never a server or a service.

An entity may be the whole device or it may be a subsystem, a subsystem of a subsystem and so on. EAT allows claims to be organized into submodules, nested EATs and so on. See Section 3.25. The entity to which a claim applies is the submodule in which it appears, or to the top-level entity if it doesn't appear in a submodule.

Some examples of entities:

- o A Secure Element
- o A TEE
- o A card in a network router
- o A network router, perhaps with each card in the router a submodule
- o An IoT device
- o An individual process
- o An app on a smartphone
- o A smartphone with many submodules for its many subsystems
- o A subsystem in a smartphone like the modem or the camera

An entity may have strong security like 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. CWT, JWT, UCCS, UJCS and DEB

An EAT is a claims set about an entity based on one of the following:

- o CBOR Web Token (CWT) [RFC8392]
- o Unprotected CWT Claims Sets (UCCS) [UCCS.Draft]
- o JSON Web Token (JWT) [RFC7519]

All definitions, requirements, creation and validation procedures, security considerations, IANA registrations and so on from these carry over to EAT.

This specification extends those specifications by defining additional claims for attestation. This specification also describes the notion of a "profile" that can narrow the definition of an EAT, ensure interoperability and fill in details for specific usage scenarios. This specification also adds some considerations for registration of future EAT-related claims.

The identification of a protocol element as an EAT, whether CBOR or JSON encoded, follows the general conventions used by CWT, JWT and

UCCS. Largely this depends on the protocol carrying the EAT. In some cases it may be by content type (e.g., MIME type). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This specification adds two more top-level messages:

- o Unprotected JWT Claims Set (UJCS) Section 4
- o Detached EAT Bundle (DEB), Section 5

A DEB is structure to hold a collection of detached claims sets and the EAT that separately provides integrity and authenticity protection for them. It can be either CBOR or JSON encoded.

1.3. CDDL, CBOR and JSON

This document defines Concise Binary Object Representation (CBOR) [RFC8949] and Javascript Object Notation (JSON) [RFC7159] encoding for an EAT. All claims in an EAT MUST use the same encoding except where explicitly allowed. It is explicitly allowed for a nested token to be of a different encoding. Some claims explicitly contain objects and messages that may use a different encoding than the enclosing EAT.

This specification uses Concise Data Definition Language (CDDL) [RFC8610] for all definitions. The implementor interprets the CDDL to come to either the CBOR or JSON encoding. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 8.2.2 where Appendix E is insufficient.

The CWT and JWT specifications were authored before CDDL was available and did not use CDDL. 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].

The UCCS specification does not include CDDL. This specification provides CDDL for it.

1.4. Operating Model and RATS Architecture

While it is not required that EAT be used with the RATS operational model described in Figure 1 in [RATS.Architecture], or even that it be used for attestation, this document is oriented around that model.

To summarize, an Attester generates Attestation Evidence. Attestation Evidence is a claims set describing various characteristics of an entity. Attestation Evidence also is usually

signed by a key that proves the entity and the evidence it produces are authentic. The claims set includes a nonce or some other means to provide freshness. EAT is designed to carry Attestation Evidence. The Attestation Evidence goes to a Verifier where the signature is verified. Some of the claims may also be checked against Reference Values. The Verifier then produces Attestation Results which is also usually a claims set. EAT is also designed to carry Attestation Results. The Attestation Results go to the Relying Party which is the ultimate consumer of the Remote Attestation Procedure. The Relying Party uses the Attestation Results as needed for the use case, perhaps allowing an entity on the network, allowing a financial transaction or such.

Note that sometimes the Verifier and Relying Party are not separate and thus there is no need for a protocol to carry Attestation Results.

1.4.1. Relationship between Attestation Evidence and Attestation Results

Any claim defined in this document or in the IANA CWT or JWT registry may be used in Attestation Evidence or Attestation Results.

Many claims in Attestation Evidence simply will pass through the Verifier to the Relying Party without modification. They will be verified as authentic from the entity by the Verifier just through normal verification of the Attester's signature. The UEID, Section 3.4, and Location, Section 3.15, are examples of claims that may be passed through.

Some claims in Attestation Evidence will be verified by the Verifier by comparison to Reference Values. These claims will not likely be conveyed to the Relying Party. Instead, some claim indicating they were checked may be added to the Attestation Results or it may be tacitly known that the Verifier always does this check. For example, the Verifier receives the Software Evidence claim, Section 3.23, compares it to Reference Values and conveys the results to the Relying Party in a Software Measurement Results Claim, Section 3.24.

In some cases the Verifier may provide privacy-preserving functionality by stripping or modifying claims that do not possess sufficient privacy-preserving characteristics. For example, the data in the Location claim, Section 3.15, may be modified to have a precision of a few kilometers rather than a few meters.

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.

This document reuses terminology from JWT [RFC7519] and CWT [RFC8392].

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.

Claim Value: The value portion of the claim. A claim value can be any CBOR data item or JSON value.

CWT/JWT Claims Set: The CBOR map or JSON object that contains the claims conveyed by the CWT or JWT.

This document reuses terminology from RATS Architecture [RATS.Architecture]

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 Attestation 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].

Attestation Evidence: A Claims Set generated by an Attester to be appraised by a Verifier. Attestation 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 Attestation 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.

3. 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.

This section also describes how several extant CWT and JWT claims apply in EAT.

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 8 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

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.

3.1. Token ID Claim (cti and jti)

CWT defines the "cti" claim. JWT defines the "jti" claim. These are equivalent to each other in EAT and carry a unique token identifier as they do in JWT and CWT. They may be used to defend against re use of the token but are distinct from the nonce that is used by the Relying Party to guarantee freshness and defend against replay.

3.2. Timestamp claim (iat)

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 floating-point for this claim. EAT disallows the use of floating-point. An EAT token MUST NOT contain an iat claim in float-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 epoch time 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.

3.3. Nonce Claim (nonce)

All EATs should have a nonce to prevent replay attacks. The nonce is generated by the Relying Party, the end consumer of the token. It is conveyed to the entity over whatever transport is in use before the token is generated and then included in the token as the nonce claim.

This documents the nonce claim for registration in the IANA CWT claims registry. This is equivalent to the JWT nonce claim that is already registered.

The nonce must be at least 8 bytes (64 bits) long as fewer bytes are unlikely to be secure. A maximum of 64 bytes is set to limit the memory a constrained implementation uses. This size range is not set for the already-registered JWT nonce, but it should follow this size recommendation when used in an EAT.

Multiple nonces are allowed to accommodate multistage verification and consumption.

```
$$claims-set-claims //=  
    (nonce-label => nonce-type / [ 2* nonce-type ])  
  
nonce-type = bstr .size (8..64)
```

3.4. Universal Entity ID Claim (ueid)

A UEID 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. 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).

There are privacy considerations for UEIDs. See Section 10.1.

The UEID is permanent. It MUST never change for a given entity.

A UEID is constructed of a single type byte followed by the bytes that are the identifier. Several types are allowed to accommodate different industries, different manufacturing processes and to have an alternative that doesn't require paying a registration fee.

Creation of new types requires a Standards Action [RFC8126].

UEIDs are variable length. All implementations MUST be able to receive UEIDs that are 33 bytes long (1 type byte and 256 bits). No UEID longer than 33 bytes SHOULD be sent.

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 uses the IEEE company identification registry. 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) [IEEE.802-2001], [OUI.Guide].
0x03	IMEI	This is a 14-digit identifier consisting of an 8-digit Type Allocation Code 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

UEIDs are not designed for direct use by humans (e.g., printing on the case of a device), so no textual representation is defined.

The consumer (the Relying Party) of a UEID MUST treat a UEID as a completely opaque string of bytes and not make any use of its internal structure. For example, they should not use the OUI part of a type 0x02 UEID to identify the manufacturer of the entity. Instead, they should use the OEMID claim. See Section 3.6. The reasons for this are:

- o UEIDs types may vary freely from one manufacturer to the next.

- o New types of UEIDs may be created. For example, a type 0x07 UEID may be created based on some other manufacturer registration scheme.
- o Entity manufacturers are allowed to change from one type of UEID to another anytime they want. For example, they may find they can optimize their manufacturing by switching from type 0x01 to type 0x02 or vice versa. The essential requirement on the manufacturer is that UEIDs be universally unique.

A Device Identifier URN is registered for UEIDs. See Section 9.3.4.

```
$$claims-set-claims // = (ueid-label => ueid-type)
```

```
ueid-type = bstr .size (7..33)
```

3.5. Semi-permanent UEIDs (SUEIDs)

An SEUID is of the same format as a UEID, but it MAY change to a different value on device life-cycle events. Examples of these events are change of ownership, factory reset and on-boarding into an IoT device management system. An entity MAY have both a UEID and SUEIDs, neither, one or the other.

There MAY be multiple SUEIDs. Each one has a text string label the purpose of which is to distinguish it from others in the token. The label MAY name the purpose, application or type of the SUEID. Typically, there will be few SUEIDs so there is no need for a formal labeling mechanism like a registry. The EAT profile MAY describe how SUEIDs should be labeled. If there is only one SUEID, the claim remains a map and there still must be a label. For example, the label for the SUEID used by FIDO Onboarding Protocol could simply be "FIDO".

There are privacy considerations for SUEIDs. See Section 10.1.

A Device Identifier URN is registered for SUEIDs. See Section 9.3.4.

```
$$claims-set-claims // = (sueids-label => sueids-type)
```

```
sueids-type = {
  + tstr => ueid-type
}
```

3.6. Hardware OEM Identification (oemid)

This 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.

3.6.1. Random Number Based OEMID

The random number based OEMID MUST always 16 bytes (128 bits).

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.

The OEM MAY also use a hash function like SHA-256 and truncate the output to 128 bits. The input to the hash should be somethings that have at least 96 bits of entropy, but preferably 128 bits of entropy. The input to the hash MAY be something whose uniqueness is managed by a central registry like a domain name.

In JSON format tokens this MUST be base64url encoded.

3.6.2. IEEE Based OEMID

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 format tokens, this MUST be base64url encoded and always 4 bytes.

3.6.3. IANA Private Enterprise Number Based OEMID

IANA maintains a integer-based company registry called the Private Enterprise Number (PEN) [PEN].

PENs are often used to create an OID. That is not the case here. They are used only as an integer.

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.

```
oemid-pen = int
```

```
oemid-ieee = bstr .size 3
```

```
oemid-random = bstr .size 16
```

```
$$claims-set-claims //= (  
    oemid-label =>  
        oemid-random / oemid-ieee / oemid-pen  
)
```

3.7. Hardware Model Claim (hardware-model)

This claim differentiates hardware models, products and variants manufactured by a particular OEM, the one identified by OEM ID in Section 3.6.

This claim must be unique so as to differentiate the models and products for the OEM ID. This claim does not have to be globally unique, but it can be. A receiver of this claim MUST not assume it is globally unique. To globally identify a particular product, the receiver should concatenate the OEM ID and this claim.

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 purpose of this claim is to identify models within protocols, not for human-readable descriptions. 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.

```
hardware-model-type = bytes .size (1..32)
```

```
$$claims-set-claims // = (  
    hardware-model-label => hardware-model-type  
)
```

3.8. Hardware Version Claims (hardware-version-claims)

The hardware version 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 [CoSWID]. It is useful to know how to sort versions so the newer can be distinguished from the older.

The hardware version can also be given by a 13-digit [EAN-13]. A new CoSWID version scheme is registered with IANA by this document in Section 9.3.3. An EAN-13 is also known as an International Article Number or most commonly as a bar code.

```
$$claims-set-claims // = (  
    hardware-version-label => hardware-version-type  
)
```

```
hardware-version-type = [  
    version:  tstr,  
    scheme:  $version-scheme  
]
```

3.9. Software Name Claim

This is a free-form text claim for the name of the software for the entity or submodule. A CoSWID manifest or other type of manifest can be used instead if this claim is too limited to correctly characterize the SW for the entity or submodule.

```
$$claims-set-claims //= ( sw-name-label => tstr )
```

3.10. Software Version Claim

This makes use of the CoSWID version scheme data type to give a simple version for the software. A full CoSWID manifest or other type of manifest can be used instead if this is too simple.

```
$$claims-set-claims //= (sw-version-label => sw-version-type)
```

```
sw-version-type = [  
    version:  tstr,  
    scheme:   $version-scheme ; As defined by CoSWID  
]
```

3.11. The Security Level Claim (security-level)

This claim characterizes the entity's ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is by defining four security levels.

This claim describes the security environment and countermeasures available on the entity where the attestation key resides and the claims originate.

- 1 - Unrestricted: There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise, the EAT provides no meaningful security assurances.
- 2 - Restricted: Entities at this level are not general-purpose operating environments that host features, such as app download systems, web browsers and complex applications. It is akin to the secure-restricted level (see below) without the security orientation. Examples include a Wi-Fi subsystem, an IoT camera, or sensor device. Often these can be considered more secure than unrestricted just because they are much simpler and a smaller attack surface, but this won't always be the case. Some unrestricted devices may be implemented in a way that provides poor protection of signing keys.

- 3 - Secure-Restricted: Entities at this level must meet the criteria defined in Section 4 of FIDO Allowed Restricted Operating Environments [FIDO.AROE]. Examples include TEE's and schemes using virtualization-based security. Security at this level is aimed at defending against large-scale network/remote attacks against the entity.
- 4 - Hardware: Entities at this level must include substantial defense against physical or electrical attacks against the entity itself. It is assumed the potential attacker has captured the entity and can disassemble it. Examples include TPMs and Secure Elements.

The entity should claim the highest security level it achieves and no higher. This set is not extensible so as to provide a common interoperable description of security level to the Relying Party. If a particular use case 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 security and its own proprietary claim as a refined indication.

This claim is not intended as a replacement for a formal security certification scheme, such as those based on FIPS 140 [FIPS-140] or those based on Common Criteria [Common.Criteria]. See Section 3.21.

```
$$claims-set-claims // = (
    security-level-label =>
        security-level-chor-type /
        security-level-json-type
)
```

```
security-level-chor-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)
```

```
security-level-json-type =
    "unrestricted" /
    "restricted" /
    "secure-restricted" /
    "hardware"
```

3.12. Secure Boot Claim (secure-boot)

The value of true indicates secure boot is enabled. Secure boot is considered enabled when the firmware and operating system, are under control of the manufacturer of the entity identified in the OEMID claim described in Section 3.6. Control by the manufacturer of the firmware and the operating system may be by it being in ROM, being cryptographically authenticated, a combination of the two or similar.

```
$$claims-set-claims // = (secure-boot-label => bool)
```

3.13. Debug Status Claim (debug-status)

This 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 root, operating system or privileged software that allow 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 debug level claim means it is not reported. A conservative interpretation might assume the enabled state.

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.

3.13.1. Enabled

If any debug facility, even manufacturer hardware diagnostics, is currently enabled, then this level must be indicated.

3.13.2. 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.

3.13.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the entity booted/started.

3.13.4. 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.

3.13.5. 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-cbor-type / debug-status-json-type
)

debug-status-cbor-type = &(
    enabled: 0,
    disabled: 1,
    disabled-since-boot: 2,
    disabled-permanently: 3,
    disabled-fully-and-permanently: 4
)

debug-status-json-type =
    "enabled" /
    "disabled" /
    "disabled-since-boot" /
    "disabled-permanently" /
    "disabled-fully-and-permanently"
```

3.14. Including Keys

An EAT may include a cryptographic key such as a public key. The signing of the EAT binds the key to all the other claims in the token.

The purpose for inclusion of the key may vary by use case. For example, the key may be included as part of an IoT device onboarding protocol. When the FIDO protocol includes a public key in its attestation message, the key represents the binding of a user, device and Relying Party. This document describes how claims containing keys should be defined for the various use cases. It does not define specific claims for specific use cases.

Keys in CBOR format tokens SHOULD be the COSE_Key format [RFC8152] and keys in JSON format tokens SHOULD be the JSON Web Key format [RFC7517]. These two formats support many common key types. Their use avoids the need to decode other serialization formats. These two formats can be extended to support further key types through their IANA registries.

The general confirmation claim format [RFC8747], [RFC7800] may also be used. It provides key encryption. It also allows for inclusion by reference through a key ID. The confirmation claim format may be employed in the definition of some new claim for a particular use case.

When the actual confirmation claim is included in an EAT, this document associates no use case semantics other than proof of possession. Different EAT use cases may choose to associate further semantics. The key in the confirmation claim **MUST** be protected in the same way as the key used to sign the EAT. That is, the same, equivalent or better hardware defenses, access controls, key generation and such must be used.

3.15. The Location Claim (location)

The location claim gives the location of the entity from which the attestation originates. It is derived from the W3C Geolocation API [W3C.GeoLoc]. The latitude, longitude, altitude and accuracy must conform to [WGS84]. The altitude is in meters above the [WGS84] ellipsoid. The two accuracy values are positive numbers in meters. The heading is in degrees relative to true north. If the entity is stationary, the heading is NaN (floating-point not-a-number). The speed is the horizontal component of the entity velocity in meters per second.

When encoding floating-point numbers half-precision **SHOULD NOT** be used. They usually do not provide enough precision for a geographic location.

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 GPS 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.

The age data item can be used when the entity doesn't know what time it is either because it doesn't have a clock or it isn't set. The entity **MUST** still have a "ticker" that can measure a time interval. The age is the interval between acquisition of the location data and token creation.

See location-related privacy considerations in Section 10.2.

```
$$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 = 1 / "latitude"  
longitude = 2 / "longitude"  
altitude = 3 / "altitude"  
accuracy = 4 / "accuracy"  
altitude-accuracy = 5 / "altitude-accuracy"  
heading = 6 / "heading"  
speed = 7 / "speed"  
timestamp = 8 / "timestamp"  
age = 9 / "age"
```

3.16. The Uptime Claim (uptime)

The "uptime" claim MUST contain a value that represents the number of seconds that have elapsed since the entity or submod was last booted.

```
$$claims-set-claims // = (uptime-label => uint)
```

3.17. The Boot Odometer Claim (odometer)

The "odometer" claim contains a value that represents the number of times the entity or submod has been booted. Support for this claim requires a persistent storage on the device.

```
$$claims-set-claims // = (odometer-label => uint)
```

3.18. The Boot Seed Claim (boot-seed)

The Boot Seed claim MUST contain a random value created at system boot time that will allow differentiation of reports from different boot sessions.

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.

`$$claims-set-claims // = (boot-seed-label => bytes)`

3.19. The Intended Use Claim (intended-use)

EAT's may be used in the context of several different applications. The intended-use 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 uses 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 intended-use 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 (CA's) may require attestations prior to the issuance of certificates related to keypairs hosted at the entity. 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-cbor-type / intended-use-json-type
)

intended-use-cbor-type = &(
    generic: 1,
    registration: 2,
    provisioning: 3,
    csr: 4,
    pop: 5
)

intended-use-json-type =
    "generic" /
    "registration" /
    "provisioning" /
    "csr" /
    "pop"
```

3.20. The Profile Claim (profile)

See Section 7 for the detailed description of a profile.

A profile is identified by either a URL or an 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 profile claim is to uniquely identify the profile even if it is a private profile.

The OID is always absolute and never relative. In CBOR tokens, the OID MUST be encoded according to [RFC9090] and the URI according to [RFC8949]. Both are unwrapped and thus not CBOR tags. In JSON tokens, the OID is a string of the form "X.X.X", and a URI is a normal URI string.

Note that this is named "eat_profile" for JWT and is distinct from the already registered "profile" claim in the JWT claims registry.

```
$$claims-set-claims //= (profile-label => ~uri / ~oid)
```

3.21. The DLOA (Digital Letter or Approval) Claim (dloas)

A DLOA (Digital Letter of Approval) [DLOA] is an XML 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. When this claim is issued by a Verifier, it MUST be because the entity has received the certification in the DLOA.

This claim MAY contain more than one DLOA. If multiple DLOAs are present, it MUST be because the entity received all of the certifications.

DLOA XML documents are always fetched from a registrar that stores them. This claim contains several data items used to construct a 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 URI 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 URI, platform label and possibly application label is specified in [DLOA].

```
$$claims-set-claims // = (  
    dloas-label => [ + dloa-type ]  
)
```

```
dloa-type = [  
    dloa_registrar: ~uri  
    dloa_platform_label: text  
    ? dloa_application_label: text  
]
```

3.22. The Software Manifests Claim (manifests)

This 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 is they are created by the software manufacturer. The purpose of these claims in an EAT is to relay them without modification to the Verifier and possibly to the Relying Party.

Some manifests may be signed by their software manufacturer before they are put into this EAT claim. When such manifests are put into this claim, the manufacturer's signature SHOULD be included. For

example, the manifest might be a CoSWID signed by the software manufacturer, in which case the full signed CoSWID should be put in this claim.

This claim allows multiple formats for the manifest. For example, the manifest may be a CBOR-format CoSWID, an XML-format SWID or other. Identification of the type of manifest is always by a CBOR tag. In many cases, for examples CoSWID, a tag will already be registered with IANA. If not, a tag MUST be registered. It can be in the first-come-first-served space which has minimal requirements for registration.

The claim is an array of one or more manifests. To facilitate hand off of the manifest to a decoding library, each manifest is contained in a byte string. This occurs for CBOR-format manifests as well as non-CBOR format manifests.

If a particular manifest type uses CBOR encoding, then the item in the array for it MUST be a byte string that contains a CBOR tag. The EAT decoder must decode the byte string and then the CBOR within it to find the tag number to identify the type of manifest. The contents of the byte string is then handed to the particular manifest processor for that type of manifest. CoSWID and SUIT manifest are examples of this.

If a particular manifest type does not use CBOR encoding, then the item in the array for it MUST be a CBOR tag that contains a byte string. The EAT decoder uses the tag to identify the processor for that type of manifest. The contents of the tag, the byte string, are handed to the manifest processor. Note that a byte string is used to contain the manifest whether it is a text based format or not. An example of this is an XML format ISO/IEC 19770 SWID.

It is not possible to describe the above requirements in CDDL, so the type for an individual manifest is any in the CDDL below. The above text sets the encoding requirement.

This claim allows for multiple manifests in one token since multiple software packages are likely to be present. The multiple manifests MAY be of multiple formats. In some cases EAT submodules may be used instead of the array structure in this claim for multiple manifests.

When the [CoSWID] format is used, it MUST be a payload CoSWID, not an evidence CoSWID.


```
$$claims-set-claims //= (  
    manifests-label => manifests-type  
)  
  
manifests-type = [+ $$manifest-formats]  
  
coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>  
  
$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx
```

3.23. The Software Evidence Claim (swevidence)

This claim contains descriptions, lists, evidence or measurements of the software that exists on the entity. 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 software manufacturer.

This claim uses the same mechanism for identification of the type of the swevidence as is used for the type of the manifest in the manifests claim. It also uses the same byte string based mechanism for containing the claim and easing the hand off to a processing library. See the discussion above in the manifests claim.

When the [CoSWID] format is used, it MUST be evidence CoSWIDs, not payload CoSWIDS.

```
$$claims-set-claims //= (  
    swevidence-label => swevidence-type  
)  
  
swevidence-type = [+ $$swevidence-formats]  
  
coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>  
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag
```

3.24. The SW Measurement Results Claim (swresults)

This claims reports the outcome of the comparison of a measurement on some software to the expected Reference Values. It may report a successful comparison, failed comparison or other.

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 swevidence claim to Reference Values.

This claim MAY also be generated on the entity if the entity has the ability for one subsystem to measure 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 attestation target or submodule, multiple results can be reported. For example, it may be desirable to report the results for the kernel and each individual application separately.

For each software objective, the following can be reported. TODO: defined objective

3.24.1. Scheme

This is the free-form text name of the verification system or scheme that performed the verification. There is no official registry of schemes or systems. It may be the name of a commercial product or such.

3.24.2. Objective

This roughly characterizes the coverage of the software measurement software. This corresponds to the attestation target or the submodule. If all of the indicated target is not covered, the measurement must indicate partial.

- 1 - all: Indicates all the software has been verified, for example, all the software in the attestation target or the submodule
- 2 - firmware: Indicates all of and only the firmware
- 3 - kernel: Refers to all of the most-privileged software, for example the Linux kernel
- 4 - privileged: Refers to all of the software used by the root, system or administrative account
- 5 - system-libs: Refers to all of the system libraries that are broadly shared and used by applications and such
- 6 - partial: Some other partial set of the software

3.24.3. Results

This describes the result of the measurement and also the comparison to Reference Values.

- 1 - verification-not-run: Indicates that no attempt was made to run the verification
- 2 - verification-indeterminate: The verification was attempted, but it did not produce a result; perhaps it ran out of memory, the battery died or such
- 3 - verification-failed: The verification ran to completion, the comparison was completed and did not compare correctly to the Reference Values
- 4 - fully-verified: The verification ran to completion and all measurements compared correctly to Reference Values
- 5 - partially-verified: The verification ran to completion and some, but not all, measurements compared correctly to Reference Values

3.24.4. Objective Name

This is a free-form text string that describes the objective. For example, "Linux kernel" or "Facebook App"

```
$$claims-set-claims // = (swresults-label => [ + swresult-type ])  
  
verification-result-cbor-type = &(  
    verification-not-run: 1,  
    verification-indeterminate: 2,  
    verification-failed: 3,  
    fully-verified: 4,  
    partially-verified: 5,  
)  
  
verification-result-json-type =  
    "verification-not-run" /  
    "verification-indeterminate" /  
    "verification-failed" /  
    "fully-verified" /  
    "partially-verified"  
  
verification-objective-cbor-type = &(  
    all: 1,  
    firmware: 2,  
    kernel: 3,  
    privileged: 4,  
    system-libs: 5,  
    partial: 6,  
)  
  
verification-objective-json-type =  
    "all" /  
    "firmware" /  
    "kernel" /  
    "privileged" /  
    "system-libs" /  
    "partial"  
  
swresult-type = [  
    verification-system: tstr,  
    objective: verification-objective-cbor-type /  
        verification-objective-json-type,  
    result: verification-result-cbor-type /  
        verification-result-json-type,  
    ? objective-name: tstr  
]
```

3.25. Submodules (submods)

Some devices are complex, having many subsystems. A mobile phone is a good example. It may have several connectivity subsystems for communications (e.g., Wi-Fi and cellular). It may have subsystems for low-power audio and video playback. It may have multiple security-oriented subsystems like a TEE and a Secure Element.

The claims for a subsystem can be grouped together in a submodule or submod.

The submods are in a single map/object, one entry per submodule. There is only one submods map/object in a token. It is identified by its specific label. It is a peer to other claims, but it is not called a claim because it is a container for a claims set rather than an individual claim. This submods part of a token allows what might be called recursion. It allows claims sets inside of claims sets inside of claims sets...

3.25.1. Submodule Types

The following sections define the three types of submodules:

- o A submodule Claims-Set
- o A nested token, which can be any valid EAT token, CBOR or JSON
- o The digest of a detached Claims-Set

3.25.1.1. Submodule Claims-Set

This is a subordinate Claims-Set containing claims about the submodule.

The submodule claims-set is produced by the same Attester as the surrounding token. It is secured using the same mechanism as the enclosing token (e.g., it is signed by the same attestation key). It roughly corresponds to an Attester Target Environment, as described in the RATS architecture.

It may contain claims that are the same as its surrounding token or superior submodules. For example, the top-level of the token may have a UEID, a submod may have a different UEID and a further subordinate submodule may also have a UEID.

The encoding of a submodule Claims-Set MUST be the same as the encoding as the token it is part of.

This data type for this type of submodule is a map/object. It is identified when decoding by it's type being a map/object.

3.25.1.2. Nested Token

This type of submodule is a fully formed complete token. It is typically produced by a separate Attester. It is typically used by a Composite Device as described in RATS Architecture [RATS.Architecture] In being a submodule of the surrounding token, it is cryptographically bound 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 does not need to use the same encoding as the enclosing token. This is to allow Composite Devices to be built without regards to the encoding supported by their Attesters. Thus a CBOR-encoded token like a CWT or UCCS can have a JWT as a nested token submodule and a JSON-encoded token can have a CWT or UCCS as a nested token submodule.

The following two sections describe how to encode and decode a nested token.

3.25.1.2.1. Surrounding EAT is CBOR-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a CBOR-encoded token.

If the nested token is CBOR-encoded, then it MUST be a CBOR tag and MUST be wrapped in a byte string. The tag identifies whether the nested token is a CWT, a UCCS, a CBOR-encoded DEB, or some other CBOR-format token defined in the future. A nested CBOR-encoded token that is not a CBOR tag is NOT allowed.

If the nested token is JSON-encoded, then the data item MUST be a text string. The text string MUST contain a JSON-encoded array of two items. The first item is a string identifying the type of the token. The second item is the JSON-encoded token.

The string identifying the JSON-encoded token MUST be one of the following:

"JWT": The second item MUST be a JWT formatted according to [RFC7519]

"UJCS": The second item MUST be a UJCS-Message as defined in this document.

"DEB": The second item MUST be a JSON-encoded Detached EAT Bundle as defined in this document.

The definition of additional types requires a standards action.

When decoding, if a byte string is encountered, it is known to be a nested CBOR-encoded token. The byte string wrapping is removed. The type of the token is determined by the CBOR tag.

When decoding, if a text string is encountered, it is known to be a JSON-encoded token. The two-item array is decoded and tells the type of the JSON-encoded token.

```
Nested-Token =  
    tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)  
    bstr .cbor Tagged-CBOR-Token
```

3.25.1.2.2. Surrounding EAT is JSON-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a JSON-encoded token.

The nested token MUST be an array of two in the same format as described in the section above.

A CBOR-encoded token nested inside a JSON-encoded MUST use the same array of two, but with the type as follows:

"CBOR": Some base64url-encoded CBOR that is a tag, typically a CWT, UCCS or CBOR-encoded DEB

When decoding, the array of two is decoded. The first item indicates the type and encoding of the nested token. If the type string is not "CBOR", then the token is JSON-encoded and of the type indicated by the string.

If the type string is "CBOR", then the token is CBOR-encoded. The base64url encoding is removed. The CBOR-encoded data is then decoded. The type of nested token is determined by the CBOR-tag. It is an error if the CBOR is not a tag.

```
Nested-Token = [  
  type : "JWT" / "CBOR" / "UJCS" / "DEB",  
  nested-token : JWT-Message /  
                  B64URL-Tagged-CBOR-Token /  
                  DEB-JSON-Message /  
                  UJCS-Message  
]
```

```
B64URL-Tagged-CBOR-Token = tstr .regexp "[A-Za-z0-9_=-]+"
```

3.25.1.3. Detached Submodule Digest

This is type of submodule equivalent to a Claims-Set submodule, except the Claims-Set is conveyed separately outside of the token.

This type of submodule consists of a digest made using a cryptographic hash of a Claims-Set. The Claims-Set is not included in the token. It is conveyed to the Verifier outside of the token. The submodule containing the digest is called a detached digest. The separately conveyed Claims-Set is called a detached claims set.

The input to the digest is exactly the byte-string wrapped encoded form of the Claims-Set for the submodule. That Claims-Set can include other submodules including nested tokens and detached digests.

The primary use for this is to facilitate the implementation of a small and secure attester, perhaps purely in hardware. This small, secure attester implements COSE signing and only a few claims, perhaps just UEID and hardware identification. It has inputs for digests of submodules, perhaps 32-byte hardware registers. Software running on the device constructs larger claim sets, perhaps very large, encodes them and digests them. The digests are written into the small secure attesters registers. The EAT produced by the small secure attester only contains the UEID, hardware identification and digests and is thus simple enough to be implemented in hardware. Probably, every data item in it is of fixed length.

The integrity protection for the larger Claims Sets will not be as secure as those originating in hardware block, but the key material and hardware-based claims will be. It is possible for the hardware to enforce hardware access control (memory protection) on the digest registers so that some of the larger claims can be more secure. For example, one register may be writable only by the TEE, so the detached claims from the TEE will have TEE-level security.

The data type for this type of submodule MUST be an array. It contains two data items, an algorithm identifier and a byte string containing the digest.

When decoding a CBOR format token the detached digest type is distinguished from the other types by it being an array. In CBOR the none of other submodule types are arrays.

When decoding a JSON format token, a little more work is required because both the nested token and detached digest types are an array. To distinguish the nested token from the detached digest, the first element in the array is examined. If it is "JWT", "UJCS" or "DEB", the the submodule is a nested token. Otherwise it will contain an algorithm identifier and is a detached digest.

A DEB, described in Section 5, may be used to convey detached claims sets and the token with their detached digests. EAT, however, doesn't require use of a DEB. Any other protocols may be used to convey detached claims sets and the token with their detached digests. Note that since detached Claims-Sets are usually signed, protocols conveying them must make sure they are not modified in transit.

3.25.2. No Inheritance

The subordinate modules do not inherit anything from the containing token. The subordinate modules must explicitly include all of their claims. This is the case even for claims like the nonce.

This rule is in place for simplicity. It avoids complex inheritance rules that might vary from one type of claim to another.

3.25.3. Security Levels

The security level of the non-token subordinate modules should always be less than or equal to that of the containing modules in the case of non-token submodules. It makes no sense for a module of lesser security to be signing claims of a module of higher security. An example of this is a TEE signing claims made by the non-TEE parts (e.g. the high-level OS) of the device.

The opposite may be true for the nested tokens. They usually have their own more secure key material. An example of this is an embedded secure element.

3.25.4. Submodule Names

The label or name for each submodule in the submods map is a text string naming the submodule. No submodules may have the same name.

3.25.5. CDDL for submods

The submodule type is distinguished in the encoded bytes by its data type, map/object for a Claims-Set, string for nested token and array for a detached submodule. Nested tokens are byte-string wrapped when encoded in CBOR and base64 encoded for JSON.

```
$$claims-set-claims // = (submods-label => { + text => Submodule })
```

```
Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest
```

```
Detached-Submodule-Digest = [  
    algorithm : int / text,  
    digest : bstr  
]
```

4. Unprotected JWT Claims-Sets

This is simply the JSON equivalent of an Unprotected CWT Claims-Set [UCCS.Draft].

It has no protection of its own so protections must be provided by the protocol carrying it. These are extensively discussed in [UCCS.Draft]. All the security discussion and security considerations in [UCCS.Draft] apply to UJCS.

(Note: The EAT author is open to this definition being moved into the UCCS draft, perhaps along with the related CDDL. It is place here for now so that the current UCCS draft plus this document are complete. UJCS is needed for the same use cases that a UCCS is needed. Further, JSON will commonly be used to convey Attestation Results since JSON is common for server to server communications. Server to server communications will often have established security (e.g., TLS) therefore the signing and encryption from JWS and JWE are unnecessary and burdensome).

5. Detached EAT Bundles

A detached EAT bundle is a structure to convey a fully-formed and signed token plus detached claims set that relate to that token. It is a top-level EAT message like a CWT, JWT, UCCS and UJCS. It can be used any place that CWT, JWT, UCCS or UJCS messages are used. It may also be sent as a submodule.

A DEB has two main parts.

The first part is a full top-level token. This top-level token must have at least one submodule that is a detached digest. This top-level token may be either CBOR or JSON-encoded. It may be a CWT, JWT, UCCS or UJCS, but not a DEB. The same mechanism for distinguishing the type for nested token submodules is used here.

The second part is a map/object containing the detached Claims-Sets corresponding to the detached digests in the full token. When the DEB is CBOR-encoded, each Claims-Set is wrapped in a byte string. When the DEB is JSON-encoded, each Claims-Set is base64url encoded. All the detached Claims-Sets MUST be encoded in the same format as the DEB. No mixing of encoding formats is allowed for the Claims-Sets in a DEB.

For CBOR-encoded DEBs, tag TBD602 can be used to identify it. The normal 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.

```
Detached-EAT-Bundle = [  
  main-token : Nested-Token,  
  detached-claims-sets: {  
    + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set  
  }  
]
```

```
json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"
```

```
cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

6. 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 [RATS.Architecture] 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 Reference Values to which EAT claims are compared as part of the verification process. 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.

6.1. 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.

6.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC8152] 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

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.

6.1.2. 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.

6.1.3. 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,...).

6.1.4. 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.

6.2. Other Considerations

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].

7. Profiles

This EAT specification does not guarantee that implementations of it will interoperate. The variability in this specification is necessary to accommodate the widely varying use cases. An EAT profile narrows the specification for a specific use case. An ideal EAT profile will guarantee interoperability.

The profile can be named in the token using the profile claim described in Section 3.20.

A profile can apply to Attestation Evidence or to Attestation Results or both.

7.1. 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.

In some cases CDDL may be created that replaces CDDL in this or other document to express some profile requirements. For example, to require the altitude data item in the location claim, CDDL can be written that replicates the location claim with the altitude no longer optional.

7.2. List of Profile Issues

The following is a list of EAT, CWT, UCCS, JWS, UJCS, COSE, JOSE and CBOR options that a profile should address.

7.2.1. Use of JSON, CBOR or both

The profile should indicate whether the token format should be CBOR, JSON, both or even some other encoding. If some other encoding, a specification for how the CDDL described here is serialized in that encoding is necessary.

This should be addressed for the top-level token and for any nested tokens. For example, a profile might require all nested tokens to be of the same encoding of the top level token.

7.2.2. CBOR Map and Array Encoding

The profile should indicate whether definite-length arrays/maps, indefinite-length arrays/maps or both are allowed. A good default is to allow only definite-length arrays/maps.

An alternate is to allow both definite and indefinite-length arrays/maps. The decoder should accept either. Encoders that need to fit on very small hardware or be actually implement in hardware can use indefinite-length encoding.

This applies to individual EAT claims, CWT and COSE parts of the implementation.

7.2.3. CBOR String Encoding

The profile should indicate whether definite-length strings, indefinite-length strings or both are allowed. A good default is to allow only definite-length strings. As with map and array encoding, allowing indefinite-length strings can be beneficial for some smaller implementations.

7.2.4. CBOR Preferred Serialization

The profile should indicate whether encoders must use preferred serialization. The profile should indicate whether decoders must accept non-preferred serialization.

7.2.5. COSE/JOSE Protection

COSE and JOSE have several options for signed, MACed and encrypted messages. EAT/CWT has the option to have no protection using UCCS and JOSE has a 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, CWT, UCCS and UJCS are allowed by EAT.

The profile should list the protections that must be supported by all decoders implementing the profile. The encoders then must implement a subset of what is listed for the decoders, perhaps only one.

Implementations may choose to sign or MAC before encryption so that the implementation layer doing the signing or MACing can be the smallest. It is often easier to make smaller implementations more secure, perhaps even implementing in solely in hardware. The key material for a signature or MAC is a private key, while for encryption it is likely to be a public key. The key for encryption requires less protection.

7.2.6. COSE/JOSE Algorithms

The profile document should list the COSE algorithms that a Verifier must implement. The Attester will select one of them. Since there is no negotiation, the Verifier should implement all algorithms listed in the profile. If detached submodules are used, the COSE algorithms allowed for their digests should also be in the profile.

7.2.7. DEB Support

A Detached EAT Bundle Section 5 is a special case message that will not often be used. A profile may prohibit its use.

7.2.8. Verification Key Identification

Section 6 describes a number of methods for identifying a verification key. The profile document should specify one of these or one that is not described. The ones described in this document are only roughly described. The profile document should go into the full detail.

7.2.9. 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, where as key identification is not.

7.2.10. Freshness

Just about every use case will require some means of knowing the EAT is recent enough and not a replay of an old token. The profile should describe how freshness is achieved. The section on Freshness in [RATS.Architecture] describes some of the possible solutions to achieve this.

7.2.11. Required Claims

The profile can list claims whose absence results in Verification failure.

7.2.12. Prohibited Claims

The profile can list claims whose presence results in Verification failure.

7.2.13. Additional Claims

The profile may describe entirely new claims. These claims can be required or optional.

7.2.14. Refined Claim Definition

The profile may lock down optional aspects of individual claims. For example, it may require altitude in the location claim, or it may require that HW Versions always be described using EAN-13.

7.2.15. CBOR Tags

The profile should specify whether the token should be a CWT Tag or not. Similarly, the profile should specify whether the token should be a UCCS 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.

7.2.16. Manifests and Software Evidence Claims

The profile should specify which formats are allowed for the manifests and software evidence claims. The profile may also go on to say which parts and options of these formats are used, allowed and prohibited.

8. 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 format.

8.1. 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 as well as UCCS. 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, JWT and UCCS. 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, JWT or UCCS in CDDL without it. The CDDL definition of Claims-Set here is applicable to EAT, CWT, JWT and UCCS.

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 encoded in either CBOR or JSON, never a mixture.

CDDL for the seven claims defined by [RFC8392] and [RFC7519] is included here.

8.2. Encoding Data Types

This makes use of the types defined in [RFC8610] Appendix D, Standard Prelude.

8.2.1. Common Data Types

time-int is identical to the epoch-based time, but disallows floating-point representation.

Unless explicitly indicated, URIs are not the URI tag defined in [RFC8949]. They are just text strings that contain a URI.

string-or-uri = tstr

time-int = #6.1(int)

8.2.2. JSON Interoperability

JSON should be encoded per [RFC8610] Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

- o bstr - must be base64url encoded
- o time - must be encoded as NumericDate as described section 2 of [RFC7519].
- o string-or-uri - must be encoded as StringOrURI as described section 2 of [RFC7519].
- o uri - must be a URI [RFC3986].

- o oid - encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1").

8.2.3. Labels

Map labels, including Claims-Keys and Claim-Names, and enumerated-type values are always integers when encoding in CBOR and strings when encoding in JSON. There is an exception to this for naming submodules and detached claims sets in a DEB. These are strings in CBOR.

The CDDL in most cases gives both the integer label and the string label as it is not convenient to have conditional CDDL for such.

8.3. CBOR Interoperability

CBOR allows data items to be serialized in more than one form. If the sender uses a form that the receiver can't decode, there will not be interoperability.

This specification gives no blanket requirements to narrow CBOR serialization for all uses of EAT. This allows individual uses to tailor serialization to the environment. It also may result in EAT implementations that don't interoperate.

One way to guarantee interoperability is to clearly specify CBOR serialization in a profile document. See Section 7 for a list of serialization issues that should be addressed.

EAT will be commonly used where the entity generating the attestation is constrained and the receiver/Verifier of the attestation is a capacious server. Following is a set of serialization requirements that work well for that use case and are guaranteed to interoperate. Use of this serialization is recommended where possible, but not required. An EAT profile may just reference the following section rather than spell out serialization details.

8.3.1. EAT Constrained Device Serialization

- o Preferred serialization described in section 4.1 of [RFC8949] is not required. The EAT decoder must accept all forms of number serialization. The EAT encoder may use any form it wishes.
- o The EAT decoder must accept indefinite length arrays and maps as described in section 3.2.2 of [RFC8949]. The EAT encoder may use indefinite length arrays and maps if it wishes.

- o The EAT decoder must accept indefinite length strings as described in section 3.2.3 of [RFC8949]. The EAT encoder may use indefinite length strings if it wishes.
- o Sorting of maps by key is not required. The EAT decoder must not rely on sorting.
- o Deterministic encoding described in Section 4.2 of [RFC8949] is not required.
- o Basic validity described in section 5.3.1 of [RFC8949] must be followed. The EAT encoder must not send duplicate map keys/labels or invalid UTF-8 strings.

8.4. Collected Common CDDL

```
Claims-Set = {  
    * $$claims-set-claims,  
    * Claim-Label .feature "extended-label" => any  
}  
  
Claim-Label = int / text  
string-or-uri = tstr  
  
time-int = #6.1(int)  
$$claims-set-claims /= (iss-label => text)  
$$claims-set-claims /= (sub-label => text)  
$$claims-set-claims /= (aud-label => text)  
$$claims-set-claims /= (exp-label => ~time)  
$$claims-set-claims /= (nbf-label => ~time)  
$$claims-set-claims /= (iat-label => ~time)  
  
$$claims-set-claims /=  
    (nonce-label => nonce-type / [ 2* nonce-type ])  
  
nonce-type = bstr .size (8..64)  
$$claims-set-claims /= (ueid-label => ueid-type)  
  
ueid-type = bstr .size (7..33)  
$$claims-set-claims /= (sueids-label => sueids-type)  
  
sueids-type = {  
    + tstr => ueid-type  
}  
oemid-pen = int  
  
oemid-ieee = bstr .size 3
```

```
oemid-random = bstr .size 16

$$claims-set-claims //= (
    oemid-label =>
        oemid-random / oemid-ieee / oemid-pen
)
$$claims-set-claims //= (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version: tstr,
    scheme: $version-scheme
]
hardware-model-type = bytes .size (1..32)

$$claims-set-claims //= (
    hardware-model-label => hardware-model-type
)
$$claims-set-claims //= ( sw-name-label => tstr )
$$claims-set-claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version: tstr,
    scheme: $version-scheme ; As defined by CoSWID
]
$$claims-set-claims //= (
    security-level-label =>
        security-level-cbor-type /
        security-level-json-type
)

security-level-cbor-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)

security-level-json-type =
    "unrestricted" /
    "restricted" /
    "secure-restricted" /
    "hardware"
$$claims-set-claims //= (secure-boot-label => bool)
$$claims-set-claims //= (
    debug-status-label =>
```

```
        debug-status-cbor-type / debug-status-json-type
    )

    debug-status-cbor-type = &(
        enabled: 0,
        disabled: 1,
        disabled-since-boot: 2,
        disabled-permanently: 3,
        disabled-fully-and-permanently: 4
    )

    debug-status-json-type =
        "enabled" /
        "disabled" /
        "disabled-since-boot" /
        "disabled-permanently" /
        "disabled-fully-and-permanently"
    $$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 = 1 / "latitude"
    longitude = 2 / "longitude"
    altitude = 3 / "altitude"
    accuracy = 4 / "accuracy"
    altitude-accuracy = 5 / "altitude-accuracy"
    heading = 6 / "heading"
    speed = 7 / "speed"
    timestamp = 8 / "timestamp"
    age = 9 / "age"
    $$claims-set-claims // = (uptime-label => uint)
    $$claims-set-claims // = (boot-seed-label => bytes)
    $$claims-set-claims // = (odometer-label => uint)
    $$claims-set-claims // = (
        intended-use-label =>
            intended-use-cbor-type / intended-use-json-type
    )
```

```
intended-use-cbor-type = &(
    generic: 1,
    registration: 2,
    provisioning: 3,
    csr: 4,
    pop: 5
)

intended-use-json-type =
    "generic" /
    "registration" /
    "provisioning" /
    "csr" /
    "pop"
$$claims-set-claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: ~uri
    dloa_platform_label: text
    ? dloa_application_label: text
]
$$claims-set-claims //= (profile-label => ~uri / ~oid)
$$claims-set-claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ $$manifest-formats]

coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>

$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx$$claims-set-claims
// = (
    swevidence-label => swevidence-type
)

swevidence-type = [+ $$swevidence-formats]

coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag
$$claims-set-claims //= (swresults-label => [ + swresult-type ])

verification-result-cbor-type = &(
    verification-not-run: 1,
    verification-indeterminate: 2,
    verification-failed: 3,
    fully-verified: 4,
    partially-verified: 5,
```

```
)

verification-result-json-type =
    "verification-not-run" /
    "verification-indeterminate" /
    "verification-failed" /
    "fully-verified" /
    "partially-verified"

verification-objective-cbor-type = &(
    all: 1,
    firmware: 2,
    kernel: 3,
    privileged: 4,
    system-libs: 5,
    partial: 6,
)

verification-objective-json-type =
    "all" /
    "firmware" /
    "kernel" /
    "privileged" /
    "system-libs" /
    "partial"

swresult-type = [
    verification-system: tstr,
    objective: verification-objective-cbor-type /
        verification-objective-json-type,
    result: verification-result-cbor-type /
        verification-result-json-type,
    ? objective-name: tstr
]
$$claims-set-claims // = (submods-label => { + text => Submodule })

Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest

Detached-Submodule-Digest = [
    algorithm : int / text,
    digest : bstr
]
Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set
    }
]
```



```
    }  
  ]
```

```
json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"
```

```
cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

8.5. Collected CDDL for CBOR

```
CBOR-Token = Tagged-CBOR-Token / Untagged-CBOR-Token
```

```
Tagged-CBOR-Token  = CWT-Tagged-Message  
Tagged-CBOR-Token /= UCCS-Tagged-Message  
Tagged-CBOR-Token /= DEB-Tagged-Message
```

```
Untagged-CBOR-Token  = CWT-Untagged-Message  
Untagged-CBOR-Token /= UCCS-Untagged-Message  
Untagged-CBOR-Token /= DEB-Untagged-Message
```

```
CWT-Tagged-Message = COSE_Tagged_Message  
CWT-Untagged-Message = COSE_Untagged_Message
```

```
UCCS-Message = UCCS-Tagged-Message / UCCS-Untagged-Message
```

```
UCCS-Tagged-Message = #6.601(UCCS-Untagged-Message)
```

```
UCCS-Untagged-Message = Claims-Set
```

```
DEB-Tagged-Message = #6.602(DEB-Untagged-Message)
```

```
DEB-Untagged-Message = Detached-EAT-Bundle
```

```
Nested-Token =  
  tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)  
  bstr .cbor Tagged-CBOR-Token
```

```
iss-label = 1  
sub-label = 2  
aud-label = 3
```

```
exp-label = 4
nbf-label = 5
iat-label = 6
cti-label = 7nonce-label = 10
ueid-label = 256
sueids-label = 257
oemid-label = 258
hardware-model-label = 259
hardware-version-label = 260
secure-boot-label = 262
debug-status-label = 263
location-label = 264
profile-label = 265
submods-label = 266
security-level-label = <TBD>
uptime-label = <TBD>
boot-seed-label = <TB>
odometer-label = <TBD>
intended-use-label = <TBD>
dloas-label = <TBD>
sw-name-label = <TBD>
sw-version-label = <TBD>
manifests-label = <TBD>
swevidence-label = <TBD>
swresults-label = <TBD>
```

8.6. Collected CDDL for JSON

```
JWT-Message = text .regexp [A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]+
```

```
UJCS-Message = Claims-Set
```

```
Nested-Token = [
  type : "JWT" / "CBOR" / "UJCS" / "DEB",
  nested-token : JWT-Message /
                  B64URL-Tagged-CBOR-Token /
                  DEB-JSON-Message /
                  UJCS-Message
]
```

```
B64URL-Tagged-CBOR-Token = tstr .regexp "[A-Za-z0-9_=-]+"
iss-label = "iss"
sub-label = "sub"
aud-label = "aud"
```

```
exp-label = "exp"
nbf-label = "nbf"
iat-label = "iat"
cti-label = "cti"nonce-label /= "nonce"
```

```
ueid-label /= "ueid"
sueids-label /= "sueids"
oemid-label /= "oemid"
hardware-model-label /= "hwmodel"
hardware-version-label /= "hwversion"
security-level-label /= "seclevel"
secure-boot-label /= "secboot"
debug-status-label /= "dbgstat"
location-label /= "location"
profile-label /= "eat-profile"
uptime-label /= "uptime"
boot-seed-label /= "bootseed"
odometer-label /= "odometer"
intended-use-label /= "intuse"
dloas-label /= "dloas"
sw-name-label /= "swname"
sw-version-label /= "swversion"
manifests-label /= "manifests"
swevidence-label /= "swevidence"
swresults-label /= "swresults"
submods-label /= "submods"
```

```
latitude /= "lat"
longitude /= "long"
altitude /= "alt"
accuracy /= "accry"
altitude-accuracy /= "alt-accry"
heading /= "heading"
speed /= "speed"
```

9. IANA Considerations

9.1. 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.

9.2. Claim Characteristics

The following is design guidance for creating new EAT claims, particularly those to be registered with IANA.

Much of this guidance is generic and could also be considered when designing new CWT or JWT claims.

9.2.1. 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.

9.2.2. 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 SW 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.

9.2.3. 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.

9.2.4. 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 format 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.

9.2.5. Proprietary Claims

EAT allows the definition and use of proprietary claims.

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.

9.3. 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 (except for the nonce claim which is already registered for JWT, but not registered for CWT).

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.

9.3.1. Claims for Early Assignment

RFC Editor: in the final publication this section should be combined with the following section as it will no longer be necessary to distinguish claims with early assignment. Also, the following paragraph should be removed.

The claims in this section have been (requested for / given) early assignment according to [RFC7120]. They have been assigned values and registered before final publication of this document. While their semantics is not expected to change in final publication, it is possible that they will. The JWT Claim Names and CWT Claim Keys are not expected to change.

In draft -06 an early allocation was described. The processing of that early allocation was never correctly completed. This early allocation assigns different numbers for the CBOR claim labels. This early allocation will presumably complete correctly

- o Claim Name: Nonce
- o Claim Description: Nonce
- o JWT Claim Name: "nonce" (already registered for JWT)
- o Claim Key: TBD (requested value 10)
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): [OpenIDConnectCore], *this document*
- o Claim Name: UEID
- o Claim Description: The Universal Entity ID
- o JWT Claim Name: "ueid"

- o CWT Claim Key: TBD (requested value 256)
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SUEIDs
- o Claim Description: Semi-permanent UEIDs
- o JWT Claim Name: "sueids"
- o CWT Claim Key: TBD (requested value 257)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Hardware OEMID
- o Claim Description: Hardware OEM ID
- o JWT Claim Name: "oemid"
- o Claim Key: TBD (requested value 258)
- o Claim Value Type(s): byte string or integer
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Hardware Model
- o Claim Description: Model identifier for hardware
- o JWT Claim Name: "hwmodel"
- o Claim Key: TBD (requested value 259)
- o Claim Value Type(s): byte string
- o Change Controller: IESG

- o Specification Document(s): *this document*
- o Claim Name: Hardware Version
- o Claim Description: Hardware Version Identifier
- o JWT Claim Name: "hwversion"
- o Claim Key: TBD (requested value 260)
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Secure Boot
- o Claim Description: Indicate whether the boot was secure
- o JWT Claim Name: "secboot"
- o Claim Key: TBD (requested value 262)
- o Claim Value Type(s): Boolean
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Debug Status
- o Claim Description: Indicate status of debug facilities
- o JWT Claim Name: "dbgstat"
- o Claim Key: TBD (requested value 263)
- o Claim Value Type(s): integer or string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Location
- o Claim Description: The geographic location

- o JWT Claim Name: "location"
- o Claim Key: TBD (requested value 264)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Profile
- o Claim Description: Indicates the EAT profile followed
- o JWT Claim Name: "eat_profile"
- o Claim Key: TBD (requested value 265)
- o Claim Value Type(s): URI or OID
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Submodules Section
- o Claim Description: The section containing submodules
- o JWT Claim Name: "submods"
- o Claim Key: TBD (requested value 266)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*

9.3.2. To be Assigned Claims

(Early assignment is NOT requested for these claims. Implementers should be aware they may change)

- o Claim Name: Security Level
- o Claim Description: Characterization of the security of an Attester or submodule

- o JWT Claim Name: "secclevel"
- o Claim Key: TBD
- o Claim Value Type(s): integer or string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Uptime
- o Claim Description: Uptime
- o JWT Claim Name: "uptime"
- o Claim Key: TBD
- o Claim Value Type(s): unsigned integer
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Boot Seed
- o Claim Description: Identifies a boot cycle
- o JWT Claim Name: "bootseed"
- o Claim Key: TBD
- o Claim Value Type(s): bytes
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Intended Use
- o Claim Description: Indicates intended use of the EAT
- o JWT Claim Name: "intuse"
- o Claim Key: TBD
- o Claim Value Type(s): integer or string

- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: DLOAs
- o Claim Description: Certifications received as Digital Letters of Approval
- o JWT Claim Name: "dloas"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Name
- o Claim Description: The name of the SW running in the entity
- o JWT Claim Name: "swname"
- o Claim Key: TBD
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Version
- o Claim Description: The version of SW running in the entity
- o JWT Claim Name: "swversion"
- o Claim Key: TBD
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Manifests

- o Claim Description: Manifests describing the SW installed on the entity
- o JWT Claim Name: "manifests"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Evidence
- o Claim Description: Measurements of the SW, memory configuration and such on the entity
- o JWT Claim Name: "swevidence"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Measurment Results
- o Claim Description: The results of comparing SW measurements to reference values
- o JWT Claim Name: "swresults"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*

9.3.3. Version Schemes Registered by this Document

IANA is requested to register a new value in the "Software Tag Version Scheme Values" established by [CoSWID].

The new value is a version scheme a 13-digit European Article Number [EAN-13]. An EAN-13 is also known as an International Article Number or most commonly as a bar code. This version scheme is the ASCII text representation of EAN-13 digits, the same ones often printed with a bar code. This version scheme must comply with the EAN allocation and assignment rules. For example, this requires the manufacturer to obtain a manufacture code from GS1.

Index	Version Scheme Name	Specification
5	ean-13	This document

9.3.4. 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

9.3.5. Tag for Detached EAT Bundle

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

Tag	Data Items	Semantics
TBD602	array	Detached EAT Bundle Section 5

10. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity and therefore, implementations should consider providing privacy-preserving options dependent on the intended usage of the EAT. Examples would include suppression of location claims for EAT's provided to unauthenticated consumers.

10.1. UEID and SUEID Privacy Considerations

A UEID is usually not privacy-preserving. Any set of Relying Parties that receives tokens that happen to be from a particular entity will be able to know the tokens are all from the same entity and be able to track it.

Thus, in many usage situations UEID violates governmental privacy regulation. 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:

- o 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.
- o The UEID/SUEID is used only in a particular context or particular use case. It is used only by one Relying Party.
- o 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.

10.2. 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 when before sending location data.

10.3. Replay Protection and Privacy

EAT offers 2 primary mechanisms for token replay protection (also sometimes known as token "freshness"): the cti/jti claim and the nonce claim. The cti/jti claim in a CWT/JWT is a field that may be optionally included in the EAT and is in general derived on the same device in which the entity is instantiated. The nonce claim is based on a value that is usually derived remotely (outside of the entity). These claims can be used to extract and convey personally-identifying information either inadvertently or by intention. For instance, an implementor may choose a cti that is 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 (e.g., if the account name is used to derive the nonce). In order to avoid the conveyance of privacy-related information in either the cti/jti or nonce claims, these fields 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.

11. 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. In addition, implementors should consider the following.

11.1. 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.

11.1.1. 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.

11.2. Transport Security

As stated in Section 8 of [RFC8392], "The security of the CWT relies upon on the protections offered by COSE". Similar considerations apply to EAT when sent as a CWT. However, EAT introduces the concept of a nonce to protect against replay. Since an EAT may be created by an entity that may not support the same type of transport security as the consumer of the EAT, intermediaries may be required to bridge communications between the entity and consumer. As a result, it is RECOMMENDED that both the consumer create a nonce, and the entity leverage the nonce along with COSE mechanisms for encryption and/or signing to create the EAT.

Similar considerations apply to the use of EAT as a JWT. Although the security of a JWT leverages the JSON Web Encryption (JWE) and JSON Web Signature (JWS) specifications, it is still recommended to make use of the EAT nonce.

11.3. 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 should leverage a secure protocol (e.g. one that uses transport-layer security, i.e. TLS),

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 transport 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.

12. References

12.1. Normative References

- [CoSWID] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-20 (work in progress), January 2022.
- [DLOA] "Digital Letter of Approval", November 2015, <https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf>.
- [EAN-13] GS1, "International Article Number - EAN/UPC barcodes", 2019, <<https://www.gs1.org/standards/barcodes/ean-upc>>.
- [FIDO.AROE] The FIDO Alliance, "FIDO Authenticator Allowed Restricted Operating Environments List", November 2020, <<https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-allowed-restricted-operating-environments-list-v1.2-fd-20201102.html>>.
- [IANA.cbor-tags] "IANA CBOR Tags Registry", n.d., <<https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>>.
- [IANA.CWT.Claims] IANA, "CBOR Web Token (CWT) Claims", <<http://www.iana.org/assignments/cwt>>.
- [IANA.JWT.Claims] IANA, "JSON Web Token (JWT) Claims", <<https://www.iana.org/assignments/jwt>>.
- [OpenIDConnectCore] Sakimura, N., Bradley, J., Jones, M., Medeiros, B. D., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

- [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>>.
- [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>>.
- [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/info/rfc7159>>.
- [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>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.
- [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>>.
- [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/info/rfc7800>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/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/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/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>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [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>>.
- [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>>.
- [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>>.
- [UCCS.Draft]
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", draft-ietf-rats-uccs-02 (work in progress), January 2022.
- [WGS84] National Geospatial-Intelligence Agency (NGA), "WORLD GEODETIC SYSTEM 1984, NGA.STND.0036_1.0.0_WGS84", July 2014, <<https://earth-info.nga.mil/php/download.php?file=coord-wgs84>>.

12.2. Informative References

- [BirthdayAttack]
"Birthday attack",
<https://en.wikipedia.org/wiki/Birthday_attack>.
- [CBOR.Cert.Draft]
Mattsson, J. P., Selander, G., Raza, S., Hoeglund, J., and
M. Furuheid, "CBOR Encoded X.509 Certificates (C509
Certificates)", draft-ietf-cose-chor-encoded-cert-03 (work
in progress), January 2022.
- [Common.Criteria]
"Common Criteria for Information Technology Security
Evaluation", April 2017,
<<https://www.commoncriteriaportal.org/cc/>>.
- [COSE.X509.Draft]
Schaad, J., "CBOR Object Signing and Encryption (COSE):
Header parameters for carrying and referencing X.509
certificates", draft-ietf-cose-x509-08 (work in progress),
December 2020.
- [FIPS-140]
National Institute of Standards, "Security Requirements for
Cryptographic Modules", May 2001,
<<https://csrc.nist.gov/publications/detail/fips/140/2/final>>.
- [IEEE.802-2001]
"IEEE Standard For Local And Metropolitan Area Networks
Overview And Architecture", 2007,
<<https://webstore.ansi.org/standards/ieee/ieee8022001r2007>>.
- [IEEE.802.1AR]
"IEEE Standard, "IEEE 802.1AR Secure Device Identifier",
December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.
- [IEEE.RA]
"IEEE Registration Authority",
<<https://standards.ieee.org/products-services/regauth/index.html>>.

[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>>.

[RATS.Architecture]

Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-15 (work in progress), February 2022.

[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>>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006,
<<https://www.rfc-editor.org/info/rfc4422>>.

[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>>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <<https://www.rfc-editor.org/info/rfc7120>>.

[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/info/rfc8446>>.

[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>>.

[W3C.GeoLoc]

Worldwide Web Consortium, "Geolocation API Specification
2nd Edition", January 2018, <[https://www.w3.org/TR/
geolocation-API/#coordinates_interface](https://www.w3.org/TR/geolocation-API/#coordinates_interface)>.

Appendix A. Examples

These examples are either UCCS, shown as CBOR diagnostic, or UJCS messages. Full CWT and JWT examples with signing and encryption are not given.

All UCCS examples can be the payload of a CWT. To do so, they must be converted from the UCCS message to a Claims-Set, which is achieved by "removing" the tag.

UJCS messages can be directly used as the payload of a JWT.

WARNING: These examples use tag and label numbers not yet assigned by IANA.

A.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 a UCCS EAT that describes a simple TEE. /

```
601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 261: 3, / secure-restricted /
  / secure-boot /    262: true,
  / debug-status /   263: 2, / disabled-since-boot /
  / manifests /      273: [
                                / This is byte-string wrapped /
                                / payload CoSWID. It gives the TEE /
                                / software name, the version and /
                                / the name of the file it is in. /
                                h' da53574944a60064336132340c01016b
                                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.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. /

{
  / nonce /          10: h'948f8860d13a463e8e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / HW OEM ID /      258: h'894823', / IEEE OUI format OEM ID /
  / HW Model ID /    259: h'549dcecc8b987c737b44e40f7c635ce8'
                        / Hash of chip model name /,
  / HW Version /     260: ["1.3.4", 1], / Multipartnumeric version /
  / SW Name /        271: "Acme OS",
  / SW Version /     272: ["3.5.5", 1],
  / secure-boot /    262: true,
  / debug-status /   263: 3, / permanent-disable /
  / timestamp (iat) / 6: 1526542894,
  / security-level / 261: 3, / secure restricted OS /
  / submods / 266: {
    / A submodule to hold some claims about the circuit board /
    "board" : {
      / HW OEM ID /    258: h'9bef8787ebal3e2c8f6e7cb4blf4619a',
      / HW Model ID / 259: h'ee80f5a66c1fb9742999a8fdab930893'
                        / Hash of board module name /,
      / HW Version /   260: ["2.0a", 2] / multipartnumeric+suffix /
    },

    / A submodule to hold claims about the overall device /
    "device" : {
      / HW OEM ID /    258: 61234, / PEN Format OEM ID /
      / HW Version /   260: ["4012345123456", 5] / EAN-13 format (barcode) /
    }
  }
}

```

A.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). /

601({
  / nonce /          10: h'948f8860d13a463e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID /          258: 64242, / Private Enterprise Number /
  / security-level / 261: 4, / hardware level security /
  / secure-boot /    262: true,
  / debug-status /   263: 3, / disabled-permanently /
  / HW version /     260: [ "3.1", 1 ] / Type is multipartnumeric /
})

```

A.4. Detached EAT Bundle

In this DEB 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 DEB itself can be assembled by untrusted SW.

```

/ This is a detached EAT bundle (DEB) tag.  /

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.                                     /
  /                                                         /
  / This token here is in UCCS format (unsigned). In a more /
  / realistic example, it would be a signed CWT.           /
  h'd90259a80a48948f8860d13a463e190100500198
  f50a4ff6c05861c8860d13a638ea19010219faf2
  19010504190106f5190107031901048263332e31
  0119010aa163544545822f5820e5cf95fd24fab7
  1446742dd58d43dae178e55fe2b94291a9291082
  ffc2635a0b',
  {
    / A CBOR-encoded byte-string wrapped EAT claims-set. It /
    / contains claims suitable for a TEE                       /
    "TEE" : h'a50a48948f8860d13a463e19010503190106f519
           01070219011181585dda53574944a60064336132
           340c01016b41636d6520544545204f530d65332e
           312e340282a2181f6b41636d6520544545204f53
           182101a2181f6b41636d6520544545204f531821
           0206a111a118186e61636d655f7465655f332e65
           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 /

601({
  / nonce /          10: h'948f8860d13a463e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID /          258: 64242, / Private Enterprise Number /
  / security-level / 261: 4, / hardware level security /
  / secure-boot /    262: true,
  / debug-status /   263: 3, / disabled-permanently /
  / hw version /     260: [ "3.1", 1 ], / multipartnumeric /
  / submods/         266: {
                                "TEE": [ / detached digest submod /
                                          -16, / SHA-256 /
                                          h'e5cf95fd24fab7144674
                                          2dd58d43dae178e55fe2
                                          b94291a9291082ffc2635
                                          a0b'
                                ]
  }
})

```

A.5. 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, for example a /
/ 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. /
/ /
/ This is similar to Android Key Attestation. /

```

```
601({
```

```

/ nonce /          10: h'948f8860d13a463e',
/ security-level / 261: 3, / secure-restricted /
/ secure-boot /    262: true,
/ debug-status /   263: 2, / disabled-since-boot /
/ manifests /      273: [
                                h'da53574944a600683762623334383766
                                0c000169436172626f6e6974650d6331
                                2e320e0102a2181f75496e6475737472
                                69616c204175746f6d6174696f6e1821
                                02'
                                / Above is an encoded CoSWID      /
                                / with the following data          /
                                /   SW Name: "Carbonite"           /
                                /   SW Vers: "1.2"                 /
                                /   SW Creator:                     /
                                /     "Industrial Automation"      /
                                ],
/ expiration /      4: 1634324274, / 2021-10-15T18:57:54Z /
/ creation time /   6: 1634317080, / 2021-10-15T16:58:00Z /
-80000 : "fingerprint",
-80001 : { / The key -- A COSE_Key /
/ kty /          1: 2, / EC2, elliptic 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 /
/ nonce /          10: h'948f8860d13a463e',
/ security-level / 261: 1, / unrestricted /
/ secure-boot /    262: true,
/ manifests /      273: [
                                h'da53574944a600687337
                                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.6. SW Measurements of an IoT Device

This is a simple token that might be for an IoT device. It includes CoSWID format measurements of the SW. The CoSWID is in byte-string wrapped in the token and also shown in diagnostic form.

```

/ This EAT UCCS 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. /

```

```

601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 261: 3, / secure-restricted /
  / secure-boot /    262: true,
  / debug-status /   263: 2, / disabled-since-boot /
  / OEMID /          258: h'8945ad', / IEEE CID based /
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / sumods /         266: {
    "OS" : {
      / security-level / 261: 2, / restricted /
      / secure-boot /    262: true,
      / debug-status /   263: 2, / disabled-since-boot /
      / swevidence /     274: [
        / This is a byte-string wrapped /
        / evidence CoSWID. It has /
        / hashes of the main files of /
        / the IoT OS. /
        h'da53574944a600663463613234350c
        17016d41636d6520522d496f542d4f
        530d65332e312e3402a2181f724163
        6d6520426173652041747465737465
        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 fictitious 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.7. Attestation Results in JSON format

This is a UJCS format token 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 security level directly through to the Relying Party. The Verifier also knows the Reference Values for the measured SW components and is able to check them. It informs the Relying Party that they were correct in the swresults claim. "Trustus Verifications" is the name of the services that verifies the SW component measurements.

This UJCS is identical to JSON-encoded Claims-Set that could be a JWT payload.

```
{  
  "nonce" : "lI+IYNE6Rj4=",  
  "secllevel" : "secure-restricted",  
  "secboot" : true,  
  "dbgstat" : "disabled-since-boot",  
  "OEMID" : "iUWt",  
  "UEID" : "AZjlCk/2wFhhyIYNE6Y4",  
  "submods" : {  
    "secllevel" : "restricted",  
    "secboot" : true,  
    "dbgstat" : "disabled-since-boot",  
    "swname" : "Acme R-IoT-OS",  
    "sw-version" : [  
      "3.1.4"  
    ],  
    "swresults" : [  
      [  
        "Trustus Verifications",  
        "all",  
        "fully-verified"  
      ]  
    ]  
  }  
}
```

Appendix B. UEID Design Rationale

B.1. Collision Probability

This calculation is to determine the probability of a collision of 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})

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 SW 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

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

Clearly, 128 bits is enough for the near future thus the requirement that 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 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.

UUIDs seem to have been designed for scenarios where the implementor does not have full control over the environment and uniqueness has to be constructed from identifiers at hand. UEID takes the view that

hardware, software and/or manufacturing process directly implement UEID in a simple and direct way. It takes the view that cryptographic quality random number generators are readily available as they are implemented in commonly used CPU hardware.

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 are used with an open set of network protocols for authentication and such. In these protocols the DevID secret is used to sign a nonce or similar to proof the association of the DevID certificates with the device.

By contrast, EAT defines network protocol for proving trustworthiness to a Relying Party, the very thing that is not defined in [IEEE.802.1AR]. Nor does not 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 defines a network protocol and [IEEE.802.1AR] doesn't. Vice versa, EAT doesn't define a device implementation and DevID does.

Hence, EAT can be the network protocol 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. Changes from Previous Drafts

The following is a list of known changes from the previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences.

D.1. From draft-rats-eat-01

- o Added UEID design rationale appendix

D.2. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but no new claims have been added.

- o Separate information and data model using CDDL.
- o Say an EAT is a CWT or JWT
- o Use a map to structure the boot_state and location claims

D.3. From draft-ietf-rats-eat-01

- o Clarifications and corrections for OEMID claim
- o Minor spelling and other fixes

- o Add the nonce claim, clarify jti claim
- D.4. From draft-ietf-rats-eat-02
- o Roll all EUIs back into one UEID type
 - o UEIDs can be one of three lengths, 128, 192 and 256.
 - o Added appendix justifying UEID design and size.
 - o Submods part now includes nested eat tokens so they can be named and there can be more than one of them
 - o Lots of fixes to the CDDL
 - o Added security considerations
- D.5. From draft-ietf-rats-eat-03
- o Split boot_state into secure-boot and debug-disable claims
 - o Debug disable is an enumerated type rather than Booleans
- D.6. From draft-ietf-rats-eat-04
- o Change IMEI-based UEIDs to be encoded as a 14-byte string
 - o CDDL cleaned up some more
 - o CDDL allows for JWTs and UCCSs
 - o CWT format submodules are byte string wrapped
 - o Allows for JWT nested in CWT and vice versa
 - o Allows UCCS (unsigned CWTs) and JWT unsecured tokens
 - o Clarify tag usage when nesting tokens
 - o Add section on key inclusion
 - o Add hardware version claims
 - o Collected CDDL is now filled in. Other CDDL corrections.
 - o Rename debug-disable to debug-status; clarify that it is not extensible

- o Security level claim is not extensible
 - o Improve specification of location claim and added a location privacy section
 - o Add intended use claim
- D.7. From draft-ietf-rats-eat-05
- o CDDL format issues resolved
 - o Corrected reference to Location Privacy section
- D.8. From draft-ietf-rats-eat-06
- o Added boot-seed claim
 - o Rework CBOR interoperability section
 - o Added profiles claim and section
- D.9. From draft-ietf-rats-eat-07
- o Filled in IANA and other sections for possible preassignment of Claim Keys for well understood claims
- D.10. From draft-ietf-rats-eat-08
- o Change profile claim to be either a URL or an OID rather than a test string
- D.11. From draft-ietf-rats-eat-09
- o Add SUEIDs
 - o Add appendix comparing IDevID to EAT
 - o Added section on use for Evidence and Attestation Results
 - o Fill in the key ID and endorsements identificaiton section
 - o Remove origination claim as it is replaced by key IDs and endorsements
 - o Added manifests and software evidence claims
 - o Add string labels non-claim labels for use with JSON (e.g. labels for members of location claim)

- o EAN-13 HW versions are no longer a separate claim. Now they are folded in as a CoSWID version scheme.

D.12. From draft-ietf-rats-eat-10

- o Hardware version is made into an array of two rather than two claims
- o Corrections and wording improvements for security levels claim
- o Add swresults claim
- o Add dloas claim - Digital Letter of Approvals, a list of certifications
- o CDDL for each claim no longer in a separate sub section
- o Consistent use of terminology from RATS architecture document
- o Consistent use of terminology from CWT and JWT documents
- o Remove operating model and procedures; refer to CWT, JWT and RATS architecture instead
- o Some reorganization of Section 1
- o Moved a few references, including RATS Architecture, to informative.
- o Add detached submodule digests and detached eat bundles (DEBs)
- o New simpler and more universal scheme for identifying the encoding of a nested token
- o Made clear that CBOR and JSON are only mixed when nesting a token in another token
- o Clearly separate CDDL for JSON and CBOR-specific data items
- o Define UJCS (unsigned JWTs)
- o Add CDDL for a general Claims-Set used by UCCS, UJCS, CWT, JWT and EAT
- o Top level CDDL for CWT correctly refers to COSE
- o OEM ID is specifically for HW, not for SW

- o HW OEM ID can now be a PEN
- o HW OEM ID can now be a 128-bit random number
- o Expand the examples section
- o Add software and version claims as easy / JSON alternative to CoSWID

D.13. From draft-ietf-rats-eat-11

- o Add HW model claim
- o Change reference for CBOR OID draft to RFC 9090
- o Correct the iat claim in some examples
- o Make HW Version just one claim rather than 3 (device, board and chip)
- o Remove CDDL comments from CDDL blocks
- o More clearly define "entity" and use it more broadly, particularly instead of "device"
- o Re do early allocation of CBOR labels since last one didn't complete correctly
- o Lots of rewording and tightening up of section 1
- o Lots of wording improvements in section 3, particularly better use of normative language
- o Improve wording in submodules section, particularly how to distinguish types when decoding
- o Remove security-level from early allocation
- o Add boot odometer claim
- o Add privacy considerations for replay protection

Authors' Addresses

Laurence Lundblade
Security Theory LLC

EMail: lg1@securitytheory.com

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough GU14 7LS
United Kingdom

Phone: +44 1252 363189
EMail: jodonogh@qti.qualcomm.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 19 April 2022

H. Birkholz
Fraunhofer SIT
E. Voit
Cisco
W. Pan
Huawei
16 October 2021

Attestation Event Stream Subscription
draft-ietf-rats-network-device-subscription-00

Abstract

This memo defines how to subscribe to YANG Event Streams for Remote Attestation Procedures (RATS). In RATS, Conceptual 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 19 April 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified 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 Conceptual 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 Conceptual 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 Conceptual 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 Conceptual 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 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.

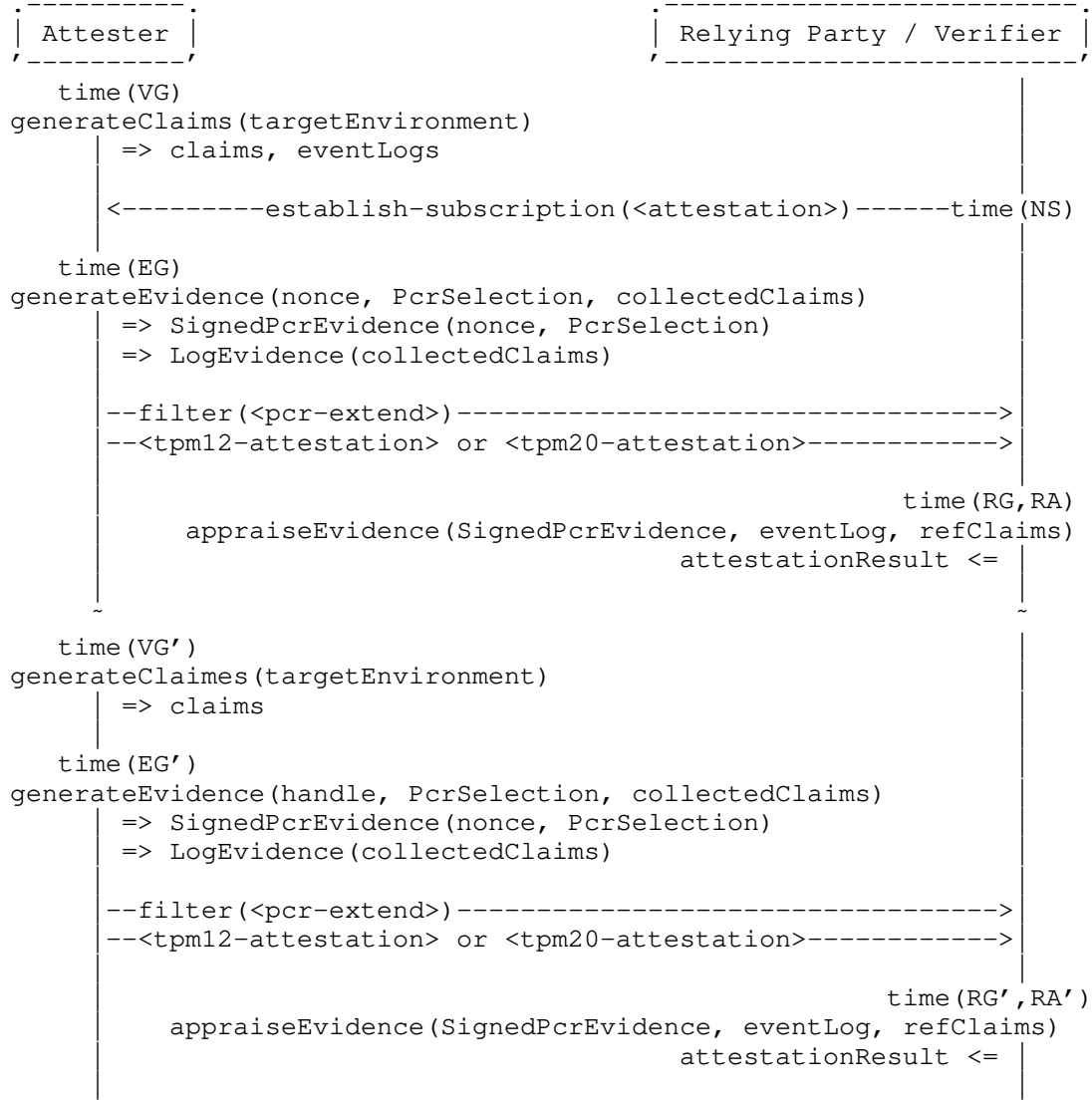


Figure 1: YANG Subscription Model for Remote Attestation

- * 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

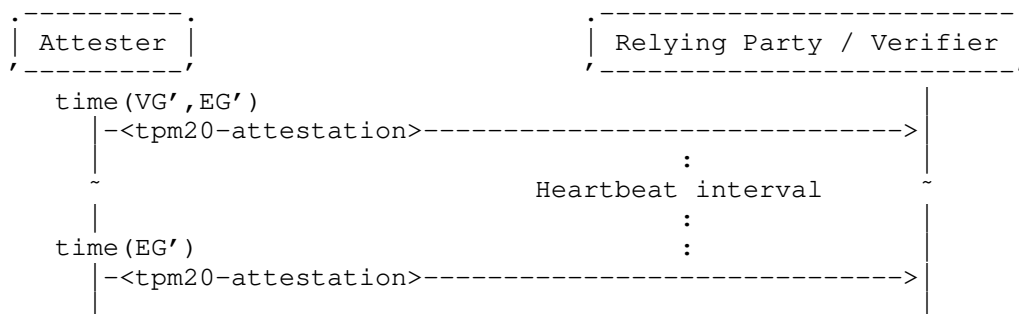
The [RFC8639] notification format includes the <eventTime> object. This can be used to determine the amount of time subsequent to the initial subscription each notification was sent. However this time is not part of the signed results which are returned from the Quote, and therefore is not trustworthy 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 counters.

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.



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 assist 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 emitted 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 emitted 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@2020-12-15.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) 2021 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 2021-05-11 {
    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 described 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 marshallng-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 Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-12, 23 April 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-12.txt>>.

[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-04, 26 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-reference-interaction-models-04.txt>>.

[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-08, 26 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-tpm-based-network-device-attest-08.txt>>.

[I-D.ietf-rats-yang-tpm-charra]
Birkholz, H., Eckel, M., Bhandari, S., Voit, E., Sulzen, B., (Frank), L. X., Laffey, T., and G. C. 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-11, 26 August 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-yang-tpm-charra-11.txt>>.

[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>>.

[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>>.

[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://www.rfc-editor.org/info/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. E., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-05, 12 July 2021, <<https://www.ietf.org/archive/id/draft-birkholz-rats-tuda-05.txt>>.

[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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

H. Birkholz
Fraunhofer SIT
E. Voit
Cisco
W. Pan
Huawei
7 March 2022

Attestation Event Stream Subscription
draft-ietf-rats-network-device-subscription-01

Abstract

This memo defines how to subscribe to YANG Event Streams for Remote Attestation Procedures (RATS). In RATS, Conceptual 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 8 September 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	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	15
5. YANG Module	15
6. Event Streams for Conceptual Messages	22
7. Security Considerations	22
8. IANA Considerations	22
9. References	23
9.1. Normative References	23
9.2. Informative References	24
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 Conceptual Messages from a TPM-based network device. However, there are limitations inherent in the challenge-response based conceptual interaction model (CHARRA [I-D.ietf-rats-reference-interaction-models]) upon which these documents are based. One of these limitation is that it is up to a Verifier to request signed Evidence as provided by [I-D.ietf-rats-yang-tpm-charra], from a separate Attester which contains a TPM. 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 Conceptual 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 Conceptual 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 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.

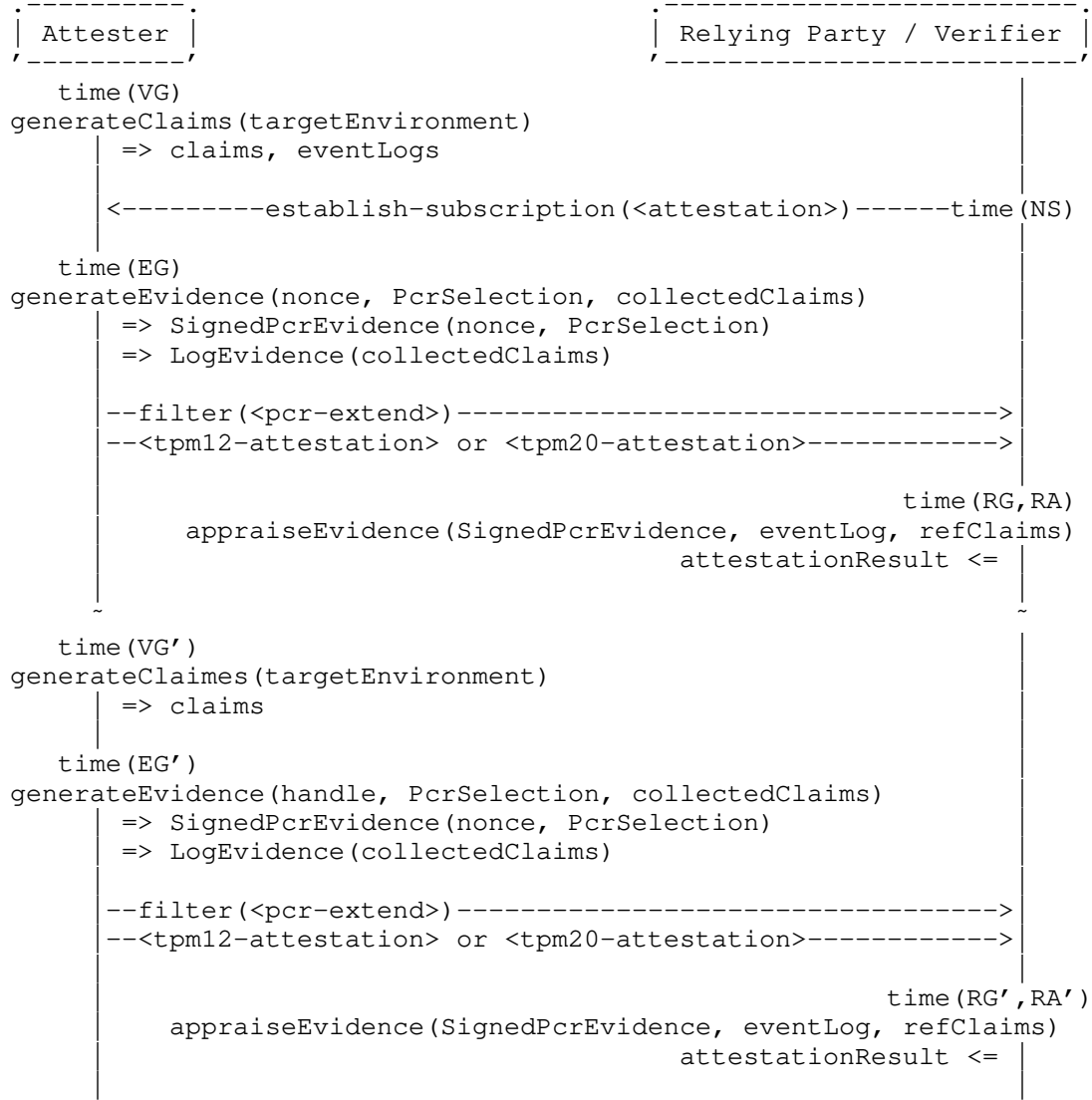


Figure 1: YANG Subscription Model for Remote Attestation

- * 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

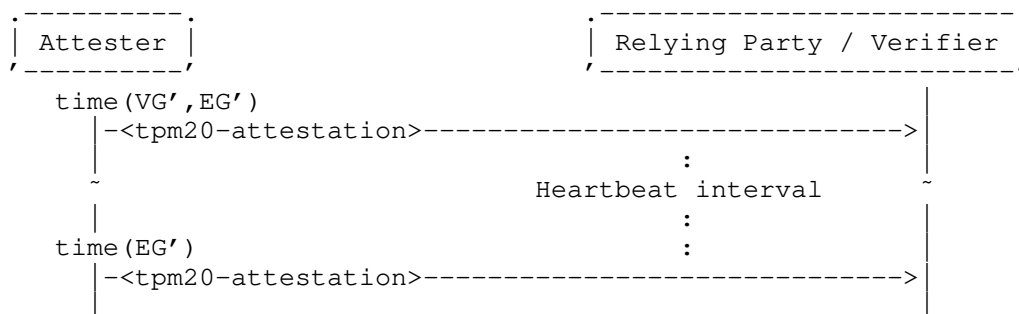
The [RFC8639] notification format includes the <eventTime> object. This can be used to determine the amount of time subsequent to the initial subscription each notification was sent. However this time is not part of the signed results which are returned from the Quote, and therefore is not trustworthy 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 counters.

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.



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 assist 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 emitted 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 emitted 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) {tpm:bios}?
+--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) {tpm:ima}?
+--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) {tpm:netequip_boot}?
+--ro boot-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

```

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 tras:marshalling-period?                uint8
  +--rw tras:tpm12-subscribed-signature-scheme?
  |   -> ../tpm:attester-supported-algos/tpm12-asymmetric-signing
  |       {taa:TPM12}?
  +--rw tras:tpm20-subscribed-signature-scheme?
  |   -> ../tpm:attester-supported-algos/tpm20-asymmetric-signing
  |       {taa:TPM20}?
  +--rw tras:tpm20-subscription-heartbeat?      uint16
  |       {taa:TPM20}?

augment /tpm:rats-support-structures/tpm:tpms:
  +--rw tras:subscription-aik?                  tpm:certificate-name-ref
  +--rw (tras:subscribable)?
  |   +--:(tras:tpm12-stream) {taa:tpm12}?
  |   |   +--rw tras:tpm12-hash-algo?          identityref
  |   |   +--rw tras:tpm12-pcr-index*          tpm:pcr
  |   +--:(tras:tpm20-stream) {taa:tpm20}?
  |   |   +--rw tras:tpm20-hash-algo?          identityref
  |   |   +--rw tras: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@2021-05-11.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) 2021 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 2021-05-11 {
    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 tpml2-attestation {
  if-feature "taa:tpml2";
  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:tpml2-attestation;
  uses tpm:tpml2-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 marshallng-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 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 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 Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-15, 8 February 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-15.txt>>.
- [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-05, 26 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-reference-interaction-models-05.txt>>.
- [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-13, 1 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-tpm-based-network-device-attest-13.txt>>.
- [I-D.ietf-rats-yang-tpm-charra]
Birkholz, H., Eckel, M., Bhandari, S., Voit, E., Sulzen, B., (Frank), L. X., Laffey, T., and G. C. 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-16, 2 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-yang-tpm-charra-16.txt>>.
- [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>>.
- [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://www.rfc-editor.org/info/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. E., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-06, 12 January 2022, <<https://www.ietf.org/archive/id/draft-birkholz-rats-tuda-06.txt>>.

[KGV]

TCG, "KGV", October 2003, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-NetEq-Attestation-Workflow-Outline_v1r9b_pubrev.pdf>.

Appendix A. Change Log

v00-v01

- * minor updates, partly based on the dependent Charra going through IESG.

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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: 14 January 2022

H. Birkholz
Fraunhofer SIT
J. O'Donoghue
Qualcomm Technologies Inc.
N. Cam-Winget
Cisco Systems
C. Bormann
Universität Bremen TZI
13 July 2021

A CBOR Tag for Unprotected CWT Claims Sets
draft-ietf-rats-uccs-01

Abstract

CBOR Web Token (CWT, RFC 8392) Claims Sets sometimes do not need the protection afforded by wrapping them into COSE, as is required for a true CWT. This specification defines a CBOR tag for such unprotected CWT Claims Sets (UCCS) and discusses conditions for its proper use.

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 14 January 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Example Use Cases	4
3. Characteristics of a Secure Channel	4
3.1. UCCS and Remote ATtestation procedureS (RATS)	5
3.2. Privacy Preserving Channels	6
4. IANA Considerations	6
5. Security Considerations	7
5.1. General Considerations	7
5.2. AES-CBC_MAC	8
5.3. AES-GCM	8
5.4. AES-CCM	8
5.5. ChaCha20 and Poly1305	8
6. References	9
6.1. Normative References	9
6.2. Informative References	9
Appendix A. Example	10
Authors' Addresses	11

1. Introduction

A CBOR Web Token (CWT) as specified by [RFC8392] is always wrapped in a CBOR Object Signing and Encryption (COSE, [RFC8152]) envelope. COSE provides -- amongst other things -- the end-to-end data origin authentication and integrity protection employed by RFC 8392 and optional encryption for CWTs. Under the right circumstances (Section 3), though, a signature providing proof for authenticity and integrity can be provided through the transfer protocol and thus omitted from the information in a CWT without compromising the intended goal of authenticity and integrity. In other words, if communicating parties have a pre-existing security association they can reuse it to provide authenticity and integrity for their messages, enabling the basic principle of using resources parsimoniously. Specifically, if a mutually Secured Channel is established between two remote peers, and if that Secure Channel provides the required properties (as discussed below), it is possible to omit the protection provided by COSE, creating a use case for unprotected CWT Claims Sets. Similarly, if there is one-way authentication, the party that did not authenticate may be in a position to send authentication information through this channel that allows the already authenticated party to authenticate the other party.

This specification allocates a CBOR tag to mark Unprotected CWT Claims Sets (UCCS) as such and discusses conditions for its proper use in the scope of Remote ATtestation procedureS (RATS) and the conveyance of Evidence from an Attester to a Verifier.

This specification does not change [RFC8392]: A true CWT does not make use of the tag allocated here; the UCCS tag is an alternative to using COSE protection and a CWT tag. Consequently, within the well-defined scope of a secured channel, it can be acceptable and economic to use the contents of a CWT without its COSE container and tag it with a UCCS CBOR tag for further processing within that scope -- or to use the contents of a UCCS CBOR tag for building a CWT to be signed by some entity that can vouch for those contents.

1.1. Terminology

The term Claim is used as in [RFC7519].

The terms Claim Key, Claim Value, and CWT Claims Set are used as in [RFC8392].

The terms Attester, Attesting Environment and Verifier are used as in [I-D.ietf-rats-architecture].

UCCS: Unprotected CWT Claims Set(s); CBOR map(s) of Claims as defined by the CWT Claims Registry that are composed of pairs of Claim Keys and Claim Values.

Secure Channel: A protected communication channel between two peers that can ensure the same qualities associated for UCCS conveyance as CWT conveyance without any additional protection.

All terms referenced or defined in this section are capitalized in the remainder of this document.

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. Example Use Cases

Use cases involving the conveyance of Claims, in particular, remote attestation procedures (RATS, see [I-D.ietf-rats-architecture]) require a standardized data definition and encoding format that can be transferred and transported using different communication channels. As these are Claims, [RFC8392] is a suitable format. However, the way these Claims are secured depends on the deployment, the security capabilities of the device, as well as their software stack. For example, a Claim may be securely stored and conveyed using a device's Trusted Execution Environment (TEE, see [I-D.ietf-tee-architecture]) or especially in some resource constrained environments, the same process that provides the secure communication transport is also the delegate to compose the Claim to be conveyed. Whether it is a transfer or transport, a Secure Channel is presumed to be used for conveying such UCCS. The following sections further describe the RATS usage scenario and corresponding requirements for UCCS deployment.

3. Characteristics of a Secure Channel

A Secure Channel for the conveyance of UCCS needs to provide the security properties that would otherwise be provided by COSE for a CWT. In this regard, UCCS is similar in security considerations to JWTs [RFC8725] using the algorithm "none". RFC 8725 states:

[...] if a JWT is cryptographically protected end-to-end by a transport layer, such as TLS using cryptographically current algorithms, there may be no need to apply another layer of cryptographic protections to the JWT. In such cases, the use of the "none" algorithm can be perfectly acceptable.

The security considerations discussed, e.g., in Sections 2.1, 3.1, and 3.2 of [RFC8725] apply in an analogous way to the use of UCCS as elaborated on in this document.

Secure Channels are often set up in a handshake protocol that mutually derives a session key, where the handshake protocol establishes the (identity and thus) authenticity of one or both ends of the communication. The session key can then be used to provide confidentiality and integrity of the transfer of information inside the Secure Channel. A well-known example of a such a Secure Channel setup protocol is the TLS [RFC8446] handshake; the TLS record protocol can then be used for secure conveyance.

As UCCS were initially created for use in Remote Attestation procedureS (RATS) Secure Channels, the following subsection provides a discussion of their use in these channels. Where other environments are intended to be used to convey UCCS, similar considerations need to be documented before UCCS can be used.

3.1. UCCS and Remote Attestation procedureS (RATS)

For the purposes of this section, the Verifier is the receiver of the UCCS and the Attester is the provider of the UCCS.

Secure Channels can be transient in nature. For the purposes of this specification, the mechanisms used to establish a Secure Channel are out of scope.

As a minimum requirement in the scope of RATS Claims, the Verifier MUST authenticate the Attester as part of the establishment of the Secure Channel. Furthermore, the channel MUST provide integrity of the communication from the Attester to the Verifier. If confidentiality is also required, the receiving side needs to be authenticated as well; this can be achieved if the Verifier and the Attester mutually authenticate when establishing the Secure Channel.

The extent to which a Secure Channel can provide assurances that UCCS originate from a trustworthy attesting environment depends on the characteristics of both the cryptographic mechanisms used to establish the channel and the characteristics of the attesting environment itself.

A Secure Channel established or maintained using weak cryptography may not provide the assurance required by a relying party of the authenticity and integrity of the UCCS.

Ultimately, it is up to the Verifier's policy to determine whether to accept a UCCS from the Attester and to the type of Secure Channel it must negotiate. While the security considerations of the cryptographic algorithms used are similar to COSE, the considerations of the secure channel should also adhere to the policy configured at each of the Attester and the Verifier. However, the policy controls and definitions are out of scope for this document.

Where the security assurance required of an attesting environment by a relying party requires it, the attesting environment may be implemented using techniques designed to provide enhanced protection from an attacker wishing to tamper with or forge UCCS. A possible approach might be to implement the attesting environment in a hardened environment such as a TEE [I-D.ietf-teep-architecture] or a TPM [TPM2].

When UCCS emerge from the Secure Channel and into the Verifier, the security properties of the Secure Channel no longer apply and UCCS have the same properties as any other unprotected data in the Verifier environment. If the Verifier subsequently forwards UCCS, they are treated as though they originated within the Verifier.

As with EATs nested in other EATs (Section 3.20.1.2 of [I-D.ietf-rats-eat]), the Secure Channel does not endorse fully formed CWTs transferred through it. Effectively, the COSE envelope of a CWT shields the CWT Claims Set from the endorsement of the Secure Channel. (Note that EAT might add a nested UCCS Claim, and this statement does not apply to UCCS nested into UCCS, only to fully formed CWTs)

3.2. Privacy Preserving Channels

A Secure Channel which preserves the privacy of the Attester may provide security properties equivalent to COSE, but only inside the life-span of the session established. In general, a Verifier cannot correlate UCCS received in different sessions from the same attesting environment based on the cryptographic mechanisms used when a privacy preserving Secure Channel is employed.

In the case of a Remote Attestation, the attester must consider whether any UCCS it returns over a privacy preserving Secure Channel compromises the privacy in unacceptable ways. As an example, the use of the EAT UEID [I-D.ietf-rats-eat] Claim in UCCS over a privacy preserving Secure Channel allows a verifier to correlate UCCS from a single attesting environment across many Secure Channel sessions. This may be acceptable in some use-cases (e.g. if the attesting environment is a physical sensor in a factory) and unacceptable in others (e.g. if the attesting environment is a device belonging to a child).

4. 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
TBD601	map	Unprotected CWT Claims Set [RFCthis]

Table 1: Values for Tags

5. Security Considerations

The security considerations of [RFC8949] apply. The security considerations of [RFC8392] need to be applied analogously, replacing the role of COSE with that of the Secured Channel.

Section 3 discusses security considerations for Secure Channels, in which UCCS might be used. This document provides the CBOR tag definition for UCCS and a discussion on security consideration for the use of UCCS in Remote ATtestation procedures (RATS). Uses of UCCS outside the scope of RATS are not covered by this document. The UCCS specification - and the use of the UCCS CBOR tag, correspondingly - is not intended for use in a scope where a scope-specific security consideration discussion has not been conducted, vetted and approved for that use.

5.1. General Considerations

Implementations of Secure Channels are often separate from the application logic that has security requirements on them. Similar security considerations to those described in [I-D.ietf-cose-rfc8152bis-struct] for obtaining the required levels of assurance include:

- * Implementations need to provide sufficient protection for private or secret key material used to establish or protect the Secure Channel.
- * Using a key for more than one algorithm can leak information about the key and is not recommended.
- * An algorithm used to establish or protect the Secure Channel may have limits on the number of times that a key can be used without leaking information about the key.

The Verifier needs to ensure that the management of key material used establish or protect the Secure Channel is acceptable. This may include factors such as:

- * Ensuring that any permissions associated with key ownership are respected in the establishment of the Secure Channel.
- * Cryptographic algorithms are used appropriately.
- * Key material is used in accordance with any usage restrictions such as freshness or algorithm restrictions.

- * Ensuring that appropriate protections are in place to address potential traffic analysis attacks.

5.2. AES-CBC_MAC

- * A given key should only be used for messages of fixed or known length.
- * Different keys should be used for authentication and encryption operations.
- * A mechanism to ensure that IV cannot be modified is required.

Section 3.2.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

5.3. AES-GCM

- * The key and nonce pair are unique for every encrypted message.
- * The maximum number of messages to be encrypted for a given key is not exceeded.

Section 4.1.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

5.4. AES-CCM

- * The key and nonce pair are unique for every encrypted message.
- * The maximum number of messages to be encrypted for a given block cipher is not exceeded.
- * The number of messages both successfully and unsuccessfully decrypted is used to determine when rekeying is required.

Section 4.2.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

5.5. ChaCha20 and Poly1305

- * The nonce is unique for every encrypted message.
- * The number of messages both successfully and unsuccessfully decrypted is used to determine when rekeying is required.

Section 4.3.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

6. References

6.1. Normative References

- [IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags",
<<http://www.iana.org/assignments/cbor-tags>>.
- [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>>.
- [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>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/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/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <<https://www.rfc-editor.org/info/rfc8725>>.
- [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>>.

6.2. Informative References

- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.
- [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-12, 23 April 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-12.txt>>.
- [I-D.ietf-rats-eat]
Mandym, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-10, 7 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-eat-10.txt>>.
- [I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-14, 22 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-teep-architecture-14.txt>>.
- [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/info/rfc8446>>.
- [TPM2] "Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59 ed., Trusted Computing Group", 2019.

Appendix A. Example

The example CWT Claims Set from Appendix A.1 of [RFC8392] can be turned into an UCCS by enclosing it with a tag number TBD601:

```
<TBD601>(  
  {  
    / iss / 1: "coap://as.example.com",  
    / sub / 2: "erikw",  
    / aud / 3: "coap://light.example.com",  
    / exp / 4: 1444064944,  
    / nbf / 5: 1443944944,  
    / iat / 6: 1443944944,  
    / cti / 7: h'0b71'  
  }  
)
```

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough
GU14 7LS
United Kingdom

Email: jodonogh@qti.qualcomm.com

Nancy Cam-Winget
Cisco Systems
3550 Cisco Way
San Jose, CA 95134
United States of America

Email: ncamwing@cisco.com

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

Phone: +49-421-218-63921

Email: cabo@tzi.org

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 16 July 2022

H. Birkholz
Fraunhofer SIT
J. O'Donoghue
Qualcomm Technologies Inc.
N. Cam-Winget
Cisco Systems
C. Bormann
Universität Bremen TZI
12 January 2022

A CBOR Tag for Unprotected CWT Claims Sets
draft-ietf-rats-uccs-02

Abstract

CBOR Web Token (CWT, RFC 8392) Claims Sets sometimes do not need the protection afforded by wrapping them into COSE, as is required for a true CWT. This specification defines a CBOR tag for such unprotected CWT Claims Sets (UCCS) and discusses conditions for its proper use.

// The present version (-01) has a few editorial improvements over
// -00 and attempts to address points from Thomas Fossati's
// 2021-03-16 review, for further discussion at IETF 111.

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-ietf-rats-uccs/>.

Discussion of this document takes place on the Remote Attestation
ProcedureS (rats) Working Group mailing list (<mailto: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-wg/draft-ietf-rats-uccs>.

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 16 July 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	3
1.1. Terminology	3
2. Example Use Cases	4
3. Characteristics of a Secure Channel	4
3.1. UCCS and Remote ATtestation procedureS (RATS)	5
3.2. Privacy Preserving Channels	6
4. IANA Considerations	7
5. Security Considerations	7
5.1. General Considerations	7
5.2. AES-CBC_MAC	8
5.3. AES-GCM	8
5.4. AES-CCM	9
5.5. ChaCha20 and Poly1305	9
6. References	9
6.1. Normative References	9
6.2. Informative References	10
Appendix A. CDDL	11
Appendix B. Example	13
Acknowledgements	14
Authors' Addresses	14

1. Introduction

A CBOR Web Token (CWT) as specified by [RFC8392] is always wrapped in a CBOR Object Signing and Encryption (COSE, [RFC8152]) envelope. COSE provides -- amongst other things -- the end-to-end data origin authentication and integrity protection employed by RFC 8392 and optional encryption for CWTs. Under the right circumstances (Section 3), though, a signature providing proof for authenticity and integrity can be provided through the transfer protocol and thus omitted from the information in a CWT without compromising the intended goal of authenticity and integrity. In other words, if communicating parties have a pre-existing security association they can reuse it to provide authenticity and integrity for their messages, enabling the basic principle of using resources parsimoniously. Specifically, if a mutually Secured Channel is established between two remote peers, and if that Secure Channel provides the required properties (as discussed below), it is possible to omit the protection provided by COSE, creating a use case for unprotected CWT Claims Sets. Similarly, if there is one-way authentication, the party that did not authenticate may be in a position to send authentication information through this channel that allows the already authenticated party to authenticate the other party.

This specification allocates a CBOR tag to mark Unprotected CWT Claims Sets (UCCS) as such and discusses conditions for its proper use in the scope of Remote ATtestation procedureS (RATS) and the conveyance of Evidence from an Attester to a Verifier.

This specification does not change [RFC8392]: A true CWT does not make use of the tag allocated here; the UCCS tag is an alternative to using COSE protection and a CWT tag. Consequently, within the well-defined scope of a secured channel, it can be acceptable and economic to use the contents of a CWT without its COSE container and tag it with a UCCS CBOR tag for further processing within that scope -- or to use the contents of a UCCS CBOR tag for building a CWT to be signed by some entity that can vouch for those contents.

1.1. Terminology

The term Claim is used as in [RFC7519].

The terms Claim Key, Claim Value, and CWT Claims Set are used as in [RFC8392].

The terms Attester, Attesting Environment and Verifier are used as in [I-D.ietf-rats-architecture].

UCCS: Unprotected CWT Claims Set(s); CBOR map(s) of Claims as defined by the CWT Claims Registry that are composed of pairs of Claim Keys and Claim Values.

Secure Channel: A protected communication channel between two peers that can ensure the same qualities associated for UCCS conveyance as CWT conveyance without any additional protection.

All terms referenced or defined in this section are capitalized in the remainder of this document.

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. Example Use Cases

Use cases involving the conveyance of Claims, in particular, remote attestation procedures (RATS, see [I-D.ietf-rats-architecture]) require a standardized data definition and encoding format that can be transferred and transported using different communication channels. As these are Claims, [RFC8392] is a suitable format. However, the way these Claims are secured depends on the deployment, the security capabilities of the device, as well as their software stack. For example, a Claim may be securely stored and conveyed using a device's Trusted Execution Environment (TEE, see [I-D.ietf-tee-architecture]) or especially in some resource constrained environments, the same process that provides the secure communication transport is also the delegate to compose the Claim to be conveyed. Whether it is a transfer or transport, a Secure Channel is presumed to be used for conveying such UCCS. The following sections further describe the RATS usage scenario and corresponding requirements for UCCS deployment.

3. Characteristics of a Secure Channel

A Secure Channel for the conveyance of UCCS needs to provide the security properties that would otherwise be provided by COSE for a CWT. In this regard, UCCS is similar in security considerations to JWTs [RFC8725] using the algorithm "none". RFC 8725 states:

[...] if a JWT is cryptographically protected end-to-end by a transport layer, such as TLS using cryptographically current algorithms, there may be no need to apply another layer of cryptographic protections to the JWT. In such cases, the use of the "none" algorithm can be perfectly acceptable.

The security considerations discussed, e.g., in Sections 2.1, 3.1, and 3.2 of [RFC8725] apply in an analogous way to the use of UCCS as elaborated on in this document.

Secure Channels are often set up in a handshake protocol that mutually derives a session key, where the handshake protocol establishes the (identity and thus) authenticity of one or both ends of the communication. The session key can then be used to provide confidentiality and integrity of the transfer of information inside the Secure Channel. A well-known example of a such a Secure Channel setup protocol is the TLS [RFC8446] handshake; the TLS record protocol can then be used for secure conveyance.

As UCCS were initially created for use in Remote Attestation procedureS (RATS) Secure Channels, the following subsection provides a discussion of their use in these channels. Where other environments are intended to be used to convey UCCS, similar considerations need to be documented before UCCS can be used.

3.1. UCCS and Remote Attestation procedureS (RATS)

For the purposes of this section, the Verifier is the receiver of the UCCS and the Attester is the provider of the UCCS.

Secure Channels can be transient in nature. For the purposes of this specification, the mechanisms used to establish a Secure Channel are out of scope.

As a minimum requirement in the scope of RATS Claims, the Verifier MUST authenticate the Attester as part of the establishment of the Secure Channel. Furthermore, the channel MUST provide integrity of the communication from the Attester to the Verifier. If confidentiality is also required, the receiving side needs to be authenticated as well; this can be achieved if the Verifier and the Attester mutually authenticate when establishing the Secure Channel.

The extent to which a Secure Channel can provide assurances that UCCS originate from a trustworthy attesting environment depends on the characteristics of both the cryptographic mechanisms used to establish the channel and the characteristics of the attesting environment itself.

A Secure Channel established or maintained using weak cryptography may not provide the assurance required by a relying party of the authenticity and integrity of the UCCS.

Ultimately, it is up to the Verifier's policy to determine whether to accept a UCCS from the Attester and to the type of Secure Channel it must negotiate. While the security considerations of the cryptographic algorithms used are similar to COSE, the considerations of the secure channel should also adhere to the policy configured at each of the Attester and the Verifier. However, the policy controls and definitions are out of scope for this document.

Where the security assurance required of an attesting environment by a relying party requires it, the attesting environment may be implemented using techniques designed to provide enhanced protection from an attacker wishing to tamper with or forge UCCS. A possible approach might be to implement the attesting environment in a hardened environment such as a TEE [I-D.ietf-teep-architecture] or a TPM [TPM2].

When UCCS emerge from the Secure Channel and into the Verifier, the security properties of the Secure Channel no longer apply and UCCS have the same properties as any other unprotected data in the Verifier environment. If the Verifier subsequently forwards UCCS, they are treated as though they originated within the Verifier.

As with EATs nested in other EATs (Section 3.20.1.2 of [I-D.ietf-rats-eat]), the Secure Channel does not endorse fully formed CWTs transferred through it. Effectively, the COSE envelope of a CWT shields the CWT Claims Set from the endorsement of the Secure Channel. (Note that EAT might add a nested UCCS Claim, and this statement does not apply to UCCS nested into UCCS, only to fully formed CWTs)

3.2. Privacy Preserving Channels

A Secure Channel which preserves the privacy of the Attester may provide security properties equivalent to COSE, but only inside the life-span of the session established. In general, a Verifier cannot correlate UCCS received in different sessions from the same attesting environment based on the cryptographic mechanisms used when a privacy preserving Secure Channel is employed.

In the case of a Remote Attestation, the attester must consider whether any UCCS it returns over a privacy preserving Secure Channel compromises the privacy in unacceptable ways. As an example, the use of the EAT UEID [I-D.ietf-rats-eat] Claim in UCCS over a privacy preserving Secure Channel allows a verifier to correlate UCCS from a single attesting environment across many Secure Channel sessions. This may be acceptable in some use-cases (e.g. if the attesting environment is a physical sensor in a factory) and unacceptable in others (e.g. if the attesting environment is a device belonging to a child).

4. 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
TBD601	map	Unprotected CWT Claims Set [RFCthis]

Table 1: Values for Tags

5. Security Considerations

The security considerations of [RFC8949] apply. The security considerations of [RFC8392] need to be applied analogously, replacing the role of COSE with that of the Secured Channel.

Section 3 discusses security considerations for Secure Channels, in which UCCS might be used. This document provides the CBOR tag definition for UCCS and a discussion on security consideration for the use of UCCS in Remote Attestation procedures (RATS). Uses of UCCS outside the scope of RATS are not covered by this document. The UCCS specification - and the use of the UCCS CBOR tag, correspondingly - is not intended for use in a scope where a scope-specific security consideration discussion has not been conducted, vetted and approved for that use.

5.1. General Considerations

Implementations of Secure Channels are often separate from the application logic that has security requirements on them. Similar security considerations to those described in [I-D.ietf-cose-rfc8152bis-struct] for obtaining the required levels of assurance include:

- * Implementations need to provide sufficient protection for private or secret key material used to establish or protect the Secure Channel.
- * Using a key for more than one algorithm can leak information about the key and is not recommended.
- * An algorithm used to establish or protect the Secure Channel may have limits on the number of times that a key can be used without leaking information about the key.

The Verifier needs to ensure that the management of key material used to establish or protect the Secure Channel is acceptable. This may include factors such as:

- * Ensuring that any permissions associated with key ownership are respected in the establishment of the Secure Channel.
- * Cryptographic algorithms are used appropriately.
- * Key material is used in accordance with any usage restrictions such as freshness or algorithm restrictions.
- * Ensuring that appropriate protections are in place to address potential traffic analysis attacks.

5.2. AES-CBC_MAC

- * A given key should only be used for messages of fixed or known length.
- * Different keys should be used for authentication and encryption operations.
- * A mechanism to ensure that IV cannot be modified is required.

Section 3.2.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

5.3. AES-GCM

- * The key and nonce pair are unique for every encrypted message.
- * The maximum number of messages to be encrypted for a given key is not exceeded.

Section 4.1.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

5.4. AES-CCM

- * The key and nonce pair are unique for every encrypted message.
- * The maximum number of messages to be encrypted for a given block cipher is not exceeded.
- * The number of messages both successfully and unsuccessfully decrypted is used to determine when rekeying is required.

Section 4.2.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

5.5. ChaCha20 and Poly1305

- * The nonce is unique for every encrypted message.
- * The number of messages both successfully and unsuccessfully decrypted is used to determine when rekeying is required.

Section 4.3.1 of [I-D.ietf-cose-rfc8152bis-algs] contains a detailed explanation of these considerations.

6. References

6.1. Normative References

- [IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags",
<<https://www.iana.org/assignments/cbor-tags>>.
- [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>>.
- [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>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/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/info/rfc8174>>.

- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <<https://www.rfc-editor.org/info/rfc8725>>.
- [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>>.

6.2. Informative References

- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.
- [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-14, 9 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-14.txt>>.
- [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-11, 24 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-eat-11.txt>>.
- [I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-

ietf-teep-architecture-15, 12 July 2021,
<<https://www.ietf.org/archive/id/draft-ietf-teep-architecture-15.txt>>.

- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [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/info/rfc8446>>.
- [RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <<https://www.rfc-editor.org/info/rfc8693>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [TPM2] "Trusted Platform Module Library Specification, Family 2.0, Level 00, Revision 01.59 ed., Trusted Computing Group", 2019.

Appendix A. CDDL

[RFC8392] does not define CDDL for CWT Claims sets.

This specification proposes using the definitions in Figure 1 for the claims set defined in [RFC8392]. Note that these definitions have been built such that they also can describe [RFC7519] claims sets by disabling feature "cbor" and enabling feature "json", but this flexibility is not the subject of the present specification.

```

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>

```

Figure 1: CDDL definition for Claims-Set

Specifications that define additional claims should also supply additions to the \$\$Claims-Set-Claims socket, e.g.:

```
; [RFC8747]
$$Claims-Set-Claims //= ( 8: CWT-cnf ) ; cnf
CWT-cnf = {
    (1: CWT-COSE-Key) //
    (2: CWT-Encrypted_COSE_Key) //
    (3: CWT-kid)
}

CWT-COSE-Key = COSE_Key
CWT-Encrypted_COSE_Key = COSE_Encrypt / COSE_Encrypt0
CWT-kid = bytes

; [RFC8693]
$$Claims-Set-Claims //= ( 9: CWT-scope ) ; scope
; TO DO: understand what this means:
; scope The scope of an access token as defined in [RFC6749].
; scope 9 byte string or text string [IESG] [RFC8693, Section 4.2]
CWT-scope = bytes / text

; [RFC-ietf-ace-oauth-authz-45, Section 5.10]
$$Claims-Set-Claims //= ( 38: CWT-ace-profile ) ; ace_profile
CWT-ace-profile = $CWT-ACE-Profiles /
    int .feature "ace_profile-extend"
; fill in from IANA registry
; https://www.iana.org/assignments/ace/ace.xhtml#ace-profiles :
$CWT-ACE-Profiles /= 1 ; coap_dtls

$$Claims-Set-Claims //= ( 39: CWT-cnonce ) ; cnonce
CWT-cnonce = bytes

$$Claims-Set-Claims //= ( 40: CWT-exi ) ; exi
CWT-exi = uint ; in seconds (5.10.3)

;;; insert CDDL from 9052-to-be to complete these CDDL definitions.
```

Appendix B. Example

The example CWT Claims Set from Appendix A.1 of [RFC8392] can be turned into an UCCS by enclosing it with a tag number TBD601:

```
<TBD601>(  
  {  
    / iss / 1: "coap://as.example.com",  
    / sub / 2: "erikw",  
    / aud / 3: "coap://light.example.com",  
    / exp / 4: 1444064944,  
    / nbf / 5: 1443944944,  
    / iat / 6: 1443944944,  
    / cti / 7: h'0b71'  
  }  
)
```

Acknowledgements

Laurence Lundblade suggested some improvements to the CDDL.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough
GU14 7LS
United Kingdom

Email: jodonogh@qti.qualcomm.com

Nancy Cam-Winget
Cisco Systems
3550 Cisco Way
San Jose, CA 95134
United States of America

Email: ncamwing@cisco.com

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

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

IETF	K. Moriarty
Internet-Draft	Center for Internet Security (CIS)
Intended status: Standards Track	A. Fontes
Expires: December 6, 2021	Dell Technologies
	June 4, 2021

Scalable Remote Attestation for Systems, Containers, and Applications
draft-moriarty-attestationsets-03

Abstract

This document establishes an architectural pattern whereby a remote attestation could be issued for a complete set of benchmarks or controls that are defined and grouped by an external entity, preventing the need to send over individual attestations for each item within a benchmark or control framework. This document establishes a pattern to list sets of benchmarks and controls within CWT and JWT formats for use as an Entity Attestation Token (EAT).

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 December 6, 2021.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Policy and Measurement Set Definitions	4
3. Supportability and Re-Attestation	4
4. Configuration Sets	5
5. Remediation	5
6. Security Considerations	5
7. IANA Considerations	6
8. Contributors	6
9. References	6
9.1. Normative References	6
9.2. Informative References	6
Appendix A. Change Log	7
Appendix B. Open Issues	7
Authors' Addresses	7

1. Introduction

Posture assessment has long been desired, but has been difficult to achieve due to complexities of customization requirements at each organization. By using policy and measurement sets that may be offered at various assurance levels, automating posture assessment through attestation becomes achievable for organizations of all sizes. The measurement and policy groupings may be provided by the vendor or by a neutral third party to enable ease of use and consistent implementations. This provides simpler options to enable posture assessment at selected levels by organizations without the need to have in-house expertise. The measurement and policy sets may also be customized, but not necessary to achieve posture assessment to predefined options. This document describes a method to use existing attestation formats and protocols while allowing for profiles of policies, benchmarks, and measurements at defined assurance levels that scale to provide transparency to posture assessment results with remote attestation.

By way of example, the Center for Internet Security (CIS) hosts recommended configuration settings to secure operating systems, applications, and devices in CIS Benchmarks developed with industry experts. Attestations aligned to the CIS Benchmarks or other configuration guide such as a DISA STIG could be used to assert the configuration meets expectations. This has already been done for multiple platforms to demonstrate assurance for firmware according to NIST SP 800-193, Firmware Resiliency Guidelines. In order to scale

remote attestation, a single attestation for a set of Benchmarks or policies being met may be sent to the remote attestation management system.

On traditional servers, assurance to NIST SP 800-193 is provable through attestation from a root of trust (RoT), using the Trusted Computing Group (TCG) Trusted Platform Module (TPM) chip and attestation formats. At boot, policy and measurement expectations are verified against a set of "golden policies" from collected and attested evidence. Device identity and measurements can also be attested at runtime. The attestations on evidence (e.g. hash of boot element) and verification of attestations are typically contained within a system and are limited to the control plane for management. The policy and measurement sets for comparison are protected to assure the result in the attestation verification process for the boot element. Event logs and PCR values may be exposed to provide transparency into the verified attestations. Remote attestation on systems is intended to provide an assessment of trust posture for all managed systems and across various layers in each of these systems in an environment.

There is a balance of exposure and evidence needed to assess posture when providing assurance of controls and system state. Currently, logs and TPM PCR values may be passed to provide assurance of verification of attestation evidence meeting set requirements. Providing the assurance can be accomplished with a remote attestation format such as the Entity Attestation Token (EAT) [I-D.ietf-rats-eat] and a RESTful interface such as ROLIE or RedFish. Policy definition blocks may be scoped to control measurement sets, where the EAT asserts compliance to the policy or measurement block specified and may include claims with the log and PCR value evidence. Measurement and Policy sets may be published and maintained by separate entities (e.g. CIS Benchmarks, DISA STIGs). The policy and measurement sets should be maintained separately even if associated with the same benchmark or control set. This avoids the need to transition the verifying entity to a remote system for individual policy and measurements which are performed locally for more immediate remediation as well as other functions.

Examples of measurement and policy sets include, but are not limited to:

- o Hardware attribute certificates
- o Hardware Attribute Certificate Comparison Results
- o Reference Integrity Measurements for firmware

- o Operating system Benchmarks at Specified Assurance Levels
- o Application hardening Benchmarks at Specified Assurance Levels
- o Container security Benchmarks at Specified Assurance Levels

Scale, ease of use, full automation, and consistency for customer consumption of a remote attestation function or service are essential toward the goal of consistently securing systems against known threats and vulnerabilities. Mitigations may be baked into policy. Measurement verification sets and the attestation that the sets meet expected policies and measurements are conveyed in an Entity Attestation Token made available to a RESTful interface in aggregate for the systems managed.

2. Policy and Measurement Set Definitions

This document defines EAT claims in the JWT [RFC7519] and CWT [RFC8392] registries to provide attestation to a set of verified claims within a defined grouping. The trustworthiness will be conveyed on original verified evidence as well as the attestation on the grouping.

rmat	Claim	Long Name	Description	Fo
	MPS	Measurement or Policy Set	Name for the MPS	
	LEM	Log Evidence of MPS	Log File or URI	
	PCR	TPM PCR Values		
	FMA	Format of MPS Attestations	Format of included attestations	
	HSH	Hash Value/Message Digest	Hash value of configuration set	

3. Supportability and Re-Attestation

The remote attestation framework shall include provisions within the system and attestation authority to allow for Product modification.

Over its lifecycle, the Product may experience modification due to: maintenance, failures, upgrades, expansion, moves, etc..

The customer can choose to:

- o Run remote attestation after product modification, or
- o Not take action and remain un-protected

In the case of Re-Attestation:

- o framework needs to invalidate previous TPM PCR values and tokens,
- o framework needs to collect new measurements,
- o framework needs to maintain history or allow for history to be logged to enable change traceability attestation, and
- o framework needs to notify that the previous attestation has been invalidated

4. Configuration Sets

In some cases, it may be difficult to attest to configuration settings for the initial or subsequent attestation and verification processes. The use of an expected hash value for configuration settings can be used to compare the attested configuration set. In this case, the creator of the attestation verification measurements would define a set of values for which a message digest would be created and then signed by the attester. The expected measurements would include the expected hash value for comparison. The configuration set could be the full attestation set to a Benchmark or a defined subset.

5. Remediation

If policy and configuration settings or measurements attested do not meet expected values, remediation is desirable. Automated remediation performed with alignment to zero trust architecture principles would require that the remediation be performed prior to any relying component executing. The relying component would verify before continuing in a zero trust architecture.

Ideally, remediation would occur on system as part of the process to attest to a set of attestations, similar to how attestation is performed for firmware in the boot process. If automated remediation is not possible, an alert should be generated to allow for notification of the variance from expected values.

6. Security Considerations

This document establishes a pattern to list sets of benchmarks and controls within CWT and JWT formats. The contents of the benchmarks and controls are out of scope for this document. This establishes an architectural pattern whereby a remote attestation could be issued for a complete set of benchmarks or controls as defined and grouped by external entities, preventing the need to send over individual

attestations for each item within a benchmark or control framework. This document does not add security considerations over what has been described in the EAT, JWT, or CWT specifications.

7. IANA Considerations

This memo includes no request to IANA, yet. This will list the initial registration sets to the JWT and CWT registries if adopted.

8. Contributors

Thank you to reviewers and contributors who helped to improve this document. Thank you to Nick Grobelney, Dell Technologies, for your review and contribution to separate out the policy and measurement sets. Thank you, Samant Kakarla and Huijun Xie from Dell Technologies, for your detailed review and corrections on boot process details. Section 3 has been contributed by Rudy Bauer from Dell as well and an author will be added on the next revision.

9. References

9.1. Normative References

- [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>>.
- [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>>.
- [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>>.

9.2. Informative References

- [I-D.ietf-rats-eat] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-09 (work in progress), March 2021.

Appendix A. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

Appendix B. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

Authors' Addresses

Kathleen M. Moriarty
Center for Internet Security (CIS)
31 Tech Valley Drive
East Greenbush, NY
US

EMail: Kathleen.Moriarty.ietf@gmail.com

Antonio Fontes
Dell Technologies
176 South Street
Hopkinton, MA
US

EMail: Antonio.Fontes@dell.com

IETF
Internet-Draft
Intended status: Standards Track
Expires: 13 June 2022

K. Moriarty
Center for Internet Security (CIS)
A. Fontes
Dell Technologies
10 December 2021

Scalable Remote Attestation for Systems, Containers, and Applications
draft-moriarty-attestationsets-04

Abstract

This document establishes an architectural pattern whereby a remote attestation could be issued for a complete set of benchmarks or controls that are defined and grouped by an external entity, preventing the need to send over individual attestations for each item within a benchmark or control framework. This document establishes a pattern to list sets of benchmarks and controls within CWT and JWT formats for use as an Entity Attestation Token (EAT).

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 June 2022.

Copyright Notice

Copyright (c) 2021 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. Policy and Measurement Set Definitions	4
3. Supportability and Re-Attestation	4
4. Configuration Sets	5
5. Remediation	5
6. Security Considerations	6
7. IANA Considerations	6
8. Contributors	6
9. References	6
9.1. Normative References	6
9.2. Informative References	6
Appendix A. Change Log	7
Appendix B. Open Issues	7
Authors' Addresses	7

1. Introduction

Posture assessment has long been desired, but has been difficult to achieve due to complexities of customization requirements at each organization. By using policy and measurement sets that may be offered at various assurance levels, automating posture assessment through attestation becomes achievable for organizations of all sizes. The measurement and policy groupings may be provided by the vendor or by a neutral third party to enable ease of use and consistent implementations. This provides simpler options to enable posture assessment at selected levels by organizations without the need to have in-house expertise. The measurement and policy sets may also be customized, but not necessary to achieve posture assessment to predefined options. This document describes a method to use existing attestation formats and protocols while allowing for profiles of policies, benchmarks, and measurements at defined assurance levels that scale to provide transparency to posture assessment results with remote attestation.

By way of example, the Center for Internet Security (CIS) hosts recommended configuration settings to secure operating systems, applications, and devices in CIS Benchmarks developed with industry

experts. Attestations aligned to the CIS Benchmarks or other configuration guide such as a DISA STIG could be used to assert the configuration meets expectations. This has already been done for multiple platforms to demonstrate assurance for firmware according to NIST SP 800-193, Firmware Resiliency Guidelines. In order to scale remote attestation, a single attestation for a set of Benchmarks or policies being met may be sent to the remote atteststation management system.

On traditional servers, assurance to NIST SP 800-193 is provable through attestation from a root of trust (RoT), using the Trusted Computing Group (TCG) Trusted Platform Module (TPM) chip and attestation formats. At boot, policy and measurement expectations are verified against a set of "golden policies" from collected and attested evidence. Device identity and measurements can also be attested at runtime. The attestations on evidence (e.g. hash of boot element) and verification of attestations are typically contained within a system and are limited to the control plane for management. The policy and measurement sets for comparison are protected to assure the result in the attestation verification process for boot element. Event logs and PCR values may be exposed to provide transparency into the verified attestations. Remote attestation on systems is intended to provide an assessment of posture for all managed systems and across various layers in each of these systems in an environment.

There is a balance of exposure and evidence needed to assess posture when providing assurance of controls and system state. Currently, logs and TPM PCR values may be passed to provide assurance of verification of attestation evidence meeting set requirements. Providing the assurance can be accomplished with a remote attestation format such as the Entity Attestation Token (EAT) [I-D.ietf-rats-eat] and a RESTful interface such as ROLIE or RedFish. Policy definition blocks may be scoped to control measurement sets, where the EAT asserts compliance to the policy or measurement block specified and may include claims with the log and PCR value evidence. Measurement and Policy sets may be published and maintained by separate entities (e.g. CIS Benchmarks, DISA STIGs). The policy and measurement sets should be maintained separately even if associated with the same benchmark or control set. This avoids the need to transition the verifying entity to a remote system for individual policy and measurements which are performed locally for more immediate remediation as well as other functions.

Examples of measurement and policy sets include, but are not limited to:

- * Hardware attribute certificates
- * Hardware Attribute Certificate Comparison Results
- * Reference Integrity Measurements for firmware
- * Operating system benchmarks at Specified Assurance Levels
- * Application hardening Benchmarks at Specified Assurance Levels
- * Container security benchmarks at Specified Assurance Levels

Scale, ease of use, full automation, and consistency for customer consumption of a remote attestation function or service are essential toward the goal of consistently securing systems against known threats and vulnerabilities. Mitigations may be baked into policy. Measurement verification sets and the attestation that the sets meet expected policies and measurements are conveyed in an Entity Attestation Token made available to a RESTful interface in aggregate for the systems managed.

2. Policy and Measurement Set Definitions

This document defines EAT claims in the JWT [RFC7519] and CWT [RFC8392] registries to provide attestation to a set of verified claims within a defined grouping. The trustworthiness will be conveyed on original verified evidence as well as the attestation on the grouping.

Format	Claim	Long Name	Description	Format
	MPS	Measurement or Policy Set	Name for the MPS	
	LEM	Log Evidence of MPS	Log File or URI	
	PCR	TPM PCR Values		
	FMA	Format of MPS Attestations	Format of included attestations	
	HSB	Hash Value/Message Digest	Hash value of configuration set	

3. Supportability and Re-Attestation

The remote attestation framework shall include provisions within the system and attestation authority to allow for Product modification.

Over its lifecycle, the Product may experience modification due to: maintenance, failures, upgrades, expansion, moves, etc..

The customer can chose to:

- * Run remote attestation after product modification, or
- * Not take action and remain un-protected

In the case of Re-Attestation:

- * framework needs to invalidate previous TPM PCR values and tokens,
- * framework needs to collect new measurements,
- * framework needs to maintain history or allow for history to be logged to enable change traceability attestation, and
- * framework needs to notify that the previous attestation has been invalidated

4. Configuration Sets

In some cases, it may be difficult to attest to configuration settings for the initial or subsequent attestation and verification processes. The use of an expected hash value for configuration settings can be used to compare the attested configuration set. In this case, the creator of the attestation verification measurements would define a set of values for which a message digest would be created and then signed by the attestor. The expected measurements would include the expected hash value for comparison. The configuration set could be the full attestation set to a Benchmark or a defined subset.

5. Remediation

If policy and configration settings or measurements attested do not meet expected values, remediation is desireable. Automated remediation performed with alignment to zero trust architecture principles would require that the remeidation be performed prior to any relying component executing. The relying component would verifiy before continuing in a zero trust architecture.

Ideally, remediation would occur on system as part of the process to attest to a set of attestations, similar to how attestation is performed for firmware in the boot process. If automated remediation is not possible, an alert should be generated to allow for notification of the variance from expected values.

6. Security Considerations

This document establishes a pattern to list sets of benchmarks and controls within CWT and JWT formats. The contents of the benchmarks and controls are out of scope for this document. This establishes an architectural pattern whereby a remote attestation could be issued for a complete set of benchmarks or controls as defined and grouped by external entities, preventing the need to send over individual attestations for each item within a benchmark or control framework. This document does not add security consideration over what has been described in the EAT, JWT, or CWT specifications.

7. IANA Considerations

This memo includes no request to IANA, yet. This will list the initial registration sets to the JWT and CWT registries if adopted.

8. Contributors

Thank you to reviewers and contributors who helped to improve this document. Thank you to Nick Grobelney, Dell Technologies, for your review and contribution to separate out the policy and measurement sets. Thank you, Samant Kakarla and Huijun Xie from Dell Technologies, for your detailed review and corrections on boot process details. Section 3 has been contributed by Rudy Bauer from Dell as well and an author will be added on the next revision.

9. References

9.1. Normative References

- [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>>.
- [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>>.
- [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>>.

9.2. Informative References

[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-11, 24 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-eat-11.txt>>.

Appendix A. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

Appendix B. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

Authors' Addresses

Kathleen M. Moriarty
Center for Internet Security (CIS)
31 Tech Valley Drive
East Greenbush, NY,
United States of America

Email: Kathleen.Moriarty.ietf@gmail.com

Antonio Fontes
Dell Technologies
176 South Street
Hopkinton, MA,
United States of America

Email: Antonio.Fontes@dell.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 11, 2022

E. Voit
Cisco
H. Birkholz
Fraunhofer SIT
T. Hardjono
MIT
T. Fossati
Arm Limited
V. Scarlata
Intel
September 07, 2021

Attestation Results for Secure Interactions
draft-voit-rats-attestation-results-02

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.

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 March 11, 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Requirements Notation	4
1.2. Terminology	4
2. Attestation Results for Secure Interactions	5
2.1. Information driving a Relying Party Action	5
2.2. Non-repudiable Identity	6
2.2.1. Attester and Attesting Environment	7
2.2.2. Verifier	9
2.2.3. Communicating Identity	10
2.3. Trustworthiness Claims	10
2.3.1. Design Principles	10
2.3.2. Enumeration Encoding	12
2.3.3. Assigning a Trustworthiness Claim value	13
2.3.4. Specific Claims	13
2.3.5. Trustworthiness Vector	17
2.3.6. Trustworthiness Vector for a type of Attesting Environment	17
2.4. Freshness	18
3. Secure Interactions Models	18
3.1. Pure Background-Check retrieval	19
3.2. Attestation Result Augmented Evidence	19
3.3. Mutual Attestation	24
3.4. Transport Protocol Integration	24
4. Privacy Considerations	24
5. Security Considerations	24
6. IANA Considerations	24
7. References	24
7.1. Normative References	24
7.2. Informative References	25
Appendix A. Supportable Trustworthiness Claims	26
A.1. Supportable Trustworthiness Claims for HSM-based CC	26
A.2. Supportable Trustworthiness Claims for process-based CC . .	28
A.3. Supportable Trustworthiness Claims for VM-based CC	29
Appendix B. Some issues being worked	30
Appendix C. Contributors	31
Authors' Addresses	31

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 [I-D.ietf-rats-architecture] 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:

- o What Attestation Results (AR) might a Relying Party be willing to trust from a specific Verifier?
- o What information does a Relying Party need before allowing interactions or choosing policies to apply to a connection?
- o 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.)

- o 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 heterogeneous 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 [I-D.ietf-rats-architecture]: Appraisal Policy for Attestation Results, Attester, Attesting Environment, Claims, Evidence, Relying Party, Target Environment and Verifier.

[I-D.ietf-rats-architecture] 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: Affirmed, 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:

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

- o Establish a transport connection between an Attester and a specific context within a Relying Party (e.g., a TEE, or Virtual Routing Function (VRF).)
- o 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 connection.
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 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 will be established with a Relying Party 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 will be accomplished via two items.

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:

- o 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?
- o Does the Relying Party only accept connections from a verified-authentic software build from a specific software developer?
- o 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 [I-D.ietf-rats-architecture] 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 continually hashes security measurements in a way which prevents an Attester from lying about measurements which have been extended into the Attesting Environment (e.g., 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:

- o chip-vendor: the vendor of the hardware chip used for the Attesting Environment (e.g., a primary Endorsement Key from a TPM)
- o chip-hardware: specific hardware with specific firmware from an 'ae-vendor'
- o 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.))
- o target-developer: the organizational unit responsible for a particular 'target-environment' (e.g., MRSIGNER [SGX])
- o 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

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 be 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 Verifier identities defined in this document.

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

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

- o Values 1 to 31: A standards enumerated reason for affirming.
- o Values -2 to -32: A non-standard reason for affirming.

Warning: The Verifier warns about this aspect of trustworthiness.

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

- o Values 96 to 127: A standards enumerated reason for the contraindication.
- o Values -97 to -128: A non-standard reason for the contraindication.

None: The Verifier makes no assertions about this Trustworthiness Claim.

- o Value 0: 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.
- o Value -1: An unexpected error occurred during the Verifier's appraisal processing. Note: while no claim is being made, the Relying Party MAY make a distinction between a Trustworthiness

Claim with enumeration '-1', and no Trustworthiness Claim being provided.

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: The configuration is a known and approved config

2: The configuration includes or exposes no known vulnerabilities

32: The configuration includes or exposes known vulnerabilities

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

-1: Unexpected error

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

2: 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.

-1: Unexpected error

file-system: A Verifier has evaluated the Attester's file system.

0: No assertion

1: 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

-1: Unexpected error

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: 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.

-1: Unexpected error

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: 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.

-1: Unexpected error

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

0: No assertion

1: 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.

-1: Unexpected error

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

0: No assertion

1: 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.

-1: Unexpected error

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: 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.

-1: Unexpected error

It is possible for additional 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 A.2 and Appendix A.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 A.

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 [I-D.ietf-rats-architecture], 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. Pure Background-Check retrieval

It is possible to for a Relying Party to follow the Background-Check Model defined in Section 5.2 of [I-D.ietf-rats-architecture]. 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:

- o Verifier scale: if the Attester has many Relying Parties, a Verifier appraising that Attester could be frequently be queried based on the same Evidence.
- o Information leak: Evidence which the Attester might consider private can be visible to the Relying Party. Hiding that Evidence could devalue any resulting appraisal.
- o 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.

3.2. Attestation Result Augmented Evidence

There is a hybrid alternative for the establishment and maintenance of trustworthiness between an Attester and a Relying Party which is not adversely impacted by the potential drawbacks with pure background-check. In this alternative, 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.birkholz-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 [I-D.ietf-rats-architecture] 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 [I-D.ietf-rats-architecture] in the following way. "Verifier A" below corresponds to the "Verifier" Figure 5 within [I-D.ietf-rats-architecture]. And "Relying Party/Verifier B" below corresponds to the union of the "Relying Party" and "Verifier" boxes within Figure 6 of [I-D.ietf-rats-architecture]. 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.

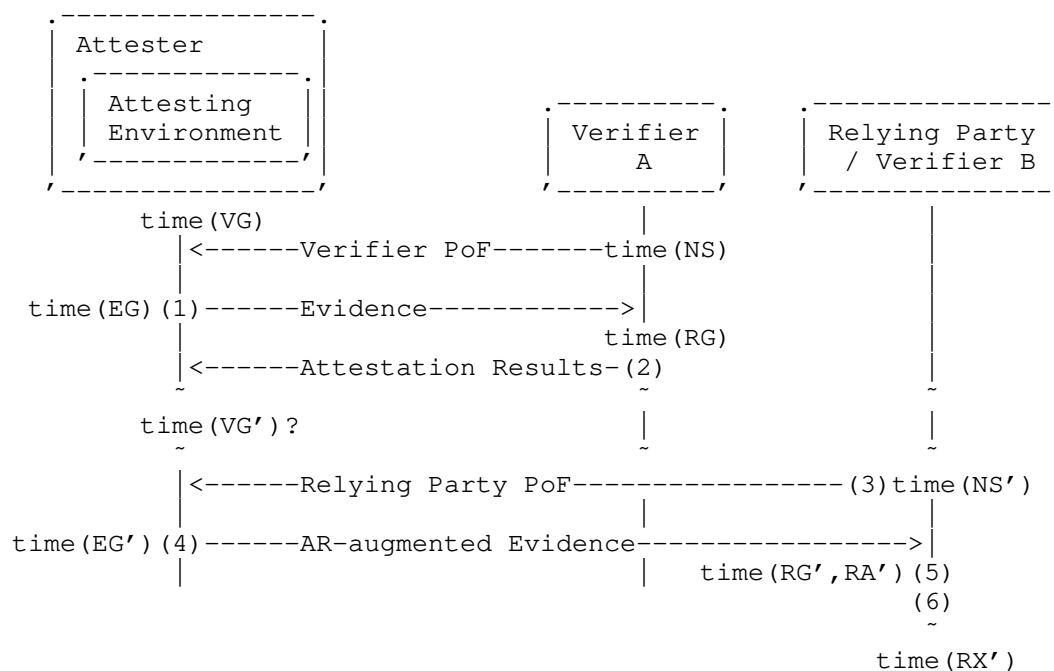


Figure 1: Secure Interactions Model

The interaction model depicted above includes specific time related events from Appendix A of [I-D.ietf-rats-architecture]. 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 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) through (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:
 1. Prune any Trustworthiness Claims not used in the Appraisal Policy for Attestation Results.
 2. Allow the information exchange from the Attester into a Relying Party context where the Verifier B appraised Trustworthiness Vector includes all the mandatory Trustworthiness Claims of value "1", and none of the disqualifying Trustworthiness Claims containing values that are not "0" or "1".
 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):

- o life-cycle events, e.g. a change to an Authentication Secret of the Attester or an update of a software component.
- o uptime-cycle events, e.g. a hard reset or a re-initialization of an Attester.
- o 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/>>.

[I-D.ietf-rats-architecture]

Fraunhofer SIT, Microsoft, Sandelman Software Works, Intel Corporation, and Huawei Technologies, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-12 (work in progress), April 2021.

[OMTP-ATE]

"Open Mobile Terminal Platform - Advanced Trusted Environment", May 2009, <<https://www.gsma.com/newsroom/wp-content/uploads/2012/03/omtpadvancedtrustedenvironmentomtptrlv11.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/info/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/info/rfc8174>>.

7.2. Informative References

[I-D.birkholz-rats-network-device-subscription]

Fraunhofer SIT, Cisco Systems, Inc., and Huawei Technologies, "Attestation Event Stream Subscription", draft-birkholz-rats-network-device-subscription-03 (work in progress), August 2021.

[I-D.tschofenig-rats-psa-token]

Arm Limited, Arm Limited, Arm Limited, Arm Limited, and Arm Limited, "Arm's Platform Security Architecture (PSA) Attestation Token", draft-tschofenig-rats-psa-token-08 (work in progress), March 2021.

[IEEE802.1AR]

"802.1AR: Secure Device Identity", 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/info/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/info/rfc8446>>.

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

Appendix A. 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.

A.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.

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

A.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 its 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 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

A.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

Appendix B. 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 C. Contributors

Guy Fedorkow

Email: gfedorkow@juniper.net

Dave Thaler

Email: dthaler@microsoft.com

Ned Smith

Email: ned.smith@intel.com

Authors' Addresses

Eric Voit

Cisco Systems

Email: evoit@cisco.com

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Thomas Hardjono
MIT

Email: hardjono@mit.edu

Thomas Fossati
Arm Limited

Email: Thomas.Fossati@arm.com

Vincent Scarlata
Intel

Email: vincent.r.scarlata@intel.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 11, 2022

E. Voit
C. Gaddam
Cisco
G. Fedorkow
Juniper
H. Birkholz
Fraunhofer SIT
September 07, 2021

Trusted Path Routing
draft-voit-rats-trustworthy-path-routing-04

Abstract

There are end-users who believe encryption technologies like IPsec alone are insufficient to protect the confidentiality of their highly sensitive traffic flows. These end-users want their flows to traverse devices which have been freshly appraised and verified for trustworthiness. This specification describes Trusted Path Routing. Trusted Path Routing protects sensitive flows as they transit a network by forwarding traffic to/from sensitive subnets across network devices recently appraised as trustworthy.

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 March 11, 2022.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
2.1. Terms	3
2.2. Requirements Notation	4
3. Implementation Prerequisites	4
4. End-to-end Solution	5
4.1. Network Topology Assembly	5
4.2. Attestation Information Flows	6
4.2.1. Step 1	8
4.2.2. Step 2	8
4.2.3. Step 3	12
4.2.4. Step 4	12
4.2.5. Step 5	14
4.2.6. Step 6	16
5. YANG Module	16
6. Security Considerations	27
7. References	27
7.1. Normative References	27
7.2. Informative References	28
Appendix A. Acknowledgements	29
Appendix B. Change Log	29
Appendix C. Open Questions	30
Authors' Addresses	31

1. Introduction

There are end-users who believe encryption technologies like IPSec alone are insufficient to protect the confidentiality of their highly sensitive traffic flows. These customers want their highly sensitive flows to be transported over only network devices recently verified as trustworthy.

By using a router's embedded TPM based cryptoprocessors in conjunction with the Remote Attestation context established by [attestation-results], a network provider can identify potentially compromised devices as well as potentially exploitable (or even exploited) vulnerabilities. Using this knowledge, it is then

possible to redirect sensitive flows around these devices while other remediations are potentially considered by Network Operations.

Trusted Path Routing allows the establishing Trusted Topologies which only include trust-verified network devices. Membership in a Trusted Topology is established and maintained via an exchange of Stamped Passports at the link layer between peering network devices. As links to Attesting Devices are appraised as meeting at least a minimum set of formally defined Trustworthiness Claims, the links are then included as members of this Trusted Topology. Routing protocols are then used to propagate topology state throughout a network.

IP Packets to and from end-user designated Sensitive Subnets are then forwarded into this Trusted Topology at each network boundary. This is done by an end user identifying sensitive IP subnets where flows with applications using these IP subnets need enhanced privacy guarantees. Trusted Path Routing passes flows to/from these Sensitive Subnets over a Trusted Topology able to meet these guarantees. The Trusted Topology itself consists of the interconnection of network devices where each potentially transited device has been verified as achieving a specific set of Trustworthiness Claims during its most recent trustworthiness appraisal. Interesting sets of Trustworthiness Claims might be marketed to end-users in the following ways:

- o all transited devices have booted with known hardware and firmware
- o all transited devices are from a specific set of vendors and are running known software containing the latest patches
- o no guarantees provided

2. Terminology

2.1. Terms

The following terms are imported from [RATS-Arch]: Attester, Evidence, Passport, Relying Party, and Verifier.

The following terms are imported from [attestation-results]: Trustworthiness Claim, Trustworthiness Vector, AR-augmented Evidence

Newly defined terms for this document:

Attested Device - a network connected Attester where a Verifier's most recent appraisal of Evidence has returned a Trustworthiness Vector.

Stamped Passport - AR-augmented Evidence which can take two forms. First if the Attester uses a TPM2, the the Verifier Proof-of-Freshness includes the <clock>, <reset-counter>, <restart-counter> and <safe> objects from a recent TPM2 quote made by that Attester, and the Relying Party Proof-of-Freshness is returned along with the timeticks as objects embedded within the most recent TPM quote signed by the same TPM2. Second, if the Attester uses a TPM1.2: the Verifier Proof-of-Freshness includes a global timestamp from that Verifier, and the Relying Party Proof-of-Freshness is embedded within a more recent TPM quote signed by the same TPM Attesting Environment.

Sensitive Subnet - an IP address range where IP packets to or from that range desire confidentially guarantees beyond those of non-identified subnets. In practice, flows to or from a Sensitive Subnet must only have their IP headers and encapsulated payloads accessible/visible only by Attested Devices supporting one or more Trustworthiness Vectors.

Transparently-Transited Device - a network device within an network domain where any packets originally passed into that network domain are completely opaque on that network device at Layer 3 and above.

Trusted Topology - a topology which includes only Attested Devices and Transparently-Transited Devices.

2.2. 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. Implementation Prerequisites

The specification is a valid instance of [attestation-results]. This specification works under the following protocol and preconfiguration prerequisite assumptions:

- o All Attested Devices support the TPM remote attestation profile as laid out in [RATS-Device], and include either [TPM2.0] or [TPM1.2].
- o One or more Verifier A's as defined in [attestation-results] 'Interaction Model' continuously appraise each of the Attested

Devices in a network domain, and these Verifiers return the Attestation Results back to each originating Attested Device.

- o The Attested Devices are connected via link layer protocols such as [MACSEC] or [IEEE-802.1X].
- o Each Attester can pass a Stamped Passport to a Relying Party / Verifier B as defined in [attestation-results] 'Interaction Model' within [RFC3748] over that link layer protocol.
- o A Trusted Topology such as [I-D.ietf-lsr-flex-algo] exists in an IGP domain for the forwarding of Sensitive Subnet traffic. This Topology will carry traffic across a set of Attested Devices which currently meet at a defined set of Trustworthiness Vectors.
- o A Relying Party is able to use mechanisms such as [I-D.ietf-lsr-flex-algo]'s affinity to include/exclude links as part of the Trusted Topology based on the appraisal of a Stamped Passport.
- o Customer designated Sensitive Subnets and their requested Trustworthiness Vectors have been identified and associated with external interfaces to/from Attested Devices at the edge of a network. Traffic to a Sensitive Subnet can be passed into the Trusted Topology by the Attested Device.
- o Relying Party/Verifier B trusts information signed by Verifier A. Verifier B has also been pre-provisioned with certificates or public keys necessary to confirm that Stamped Passports came from Verifier A.

4. End-to-end Solution

4.1. Network Topology Assembly

To be included in a Trusted Topology, Stamped Passports are shared between Attested Devices (such as routers). Upon receiving and appraising the Stamped Passport as part of link layer authentication, the Relying Party Attested Device decides if this link should be added as an active adjacency for a particular Trusted Topology. In Figure 1 below, this might be done by applying an Appraisal Policy for Attestation Results which requires any Attesting Device be most recently appraised with the Trustworthiness Claim 'hw-authentic'. If Attested Device 'x' has been appraised with 'hw-verification-fail' is would not become part of the Trustworthy Topology.

When enough links have been successfully added, the Trusted Topology will support edge-to-edge forwarding as routing protocols flood the adjacency information across the network domain.

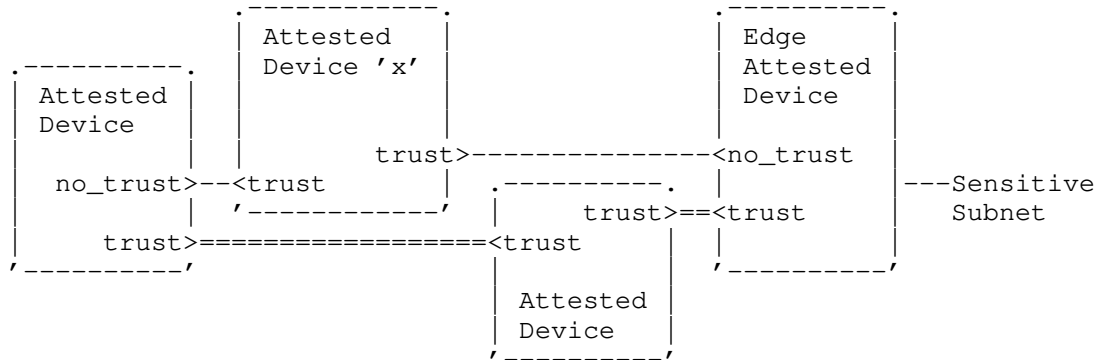


Figure 1: Trusted Path Topology Assembly

As the process described above repeats over time across the set of links within a network domain, Trusted Topologies can be extended and maintained. Traffic to and from Sensitive Subnets is then identified at the edges of the network domain and passed into this Trusted Topology. Traffic exchanged with Sensitive Subnets can then be forwarded across that Trusted Topology from all edges of the network domain.

4.2. Attestation Information Flows

Critical to the establishment and maintenance of a Trusted Topology is the Stamped Passport. A Stamped Passport is comprised of Evidence from both an Attester and a Verifier. A Stamped Passport is a valid type of AR-augmented evidence as described in [attestation-results].

Stamped Passports are exchanged between adjacent network devices over a link layer protocols like 802.1x or MACSEC. As both sides of a link may need might need to appraise the other, independent Stamped Passports will often be transmitted from either side of the link. Additionally, as link layer protocols will continuously re-authenticate the link, this allows for fresh Stamped Passports to be constantly appraised by either side of the connection.

Each Stamped Passport will include the most recent Verifier provided Attestation Results, as well as the most recent TPM Quote for that Attester. Upon receiving this information as part of link layer authentication, the Relying Party Router appraises the results and decides if this link should be added to a Trusted Topology.

Figure 2 describes this flow of information using the time definitions described in [RATS-Arch], and the information flows defined in Section 7 of [RATS-Interactions]. This figure is also a valid embodiment of the "Interaction Model" described within [attestation-results]. (Note that the Relying Party must also be an Attested Device in order to attract Sensitive Subnet traffic which may flow from the Attester.)

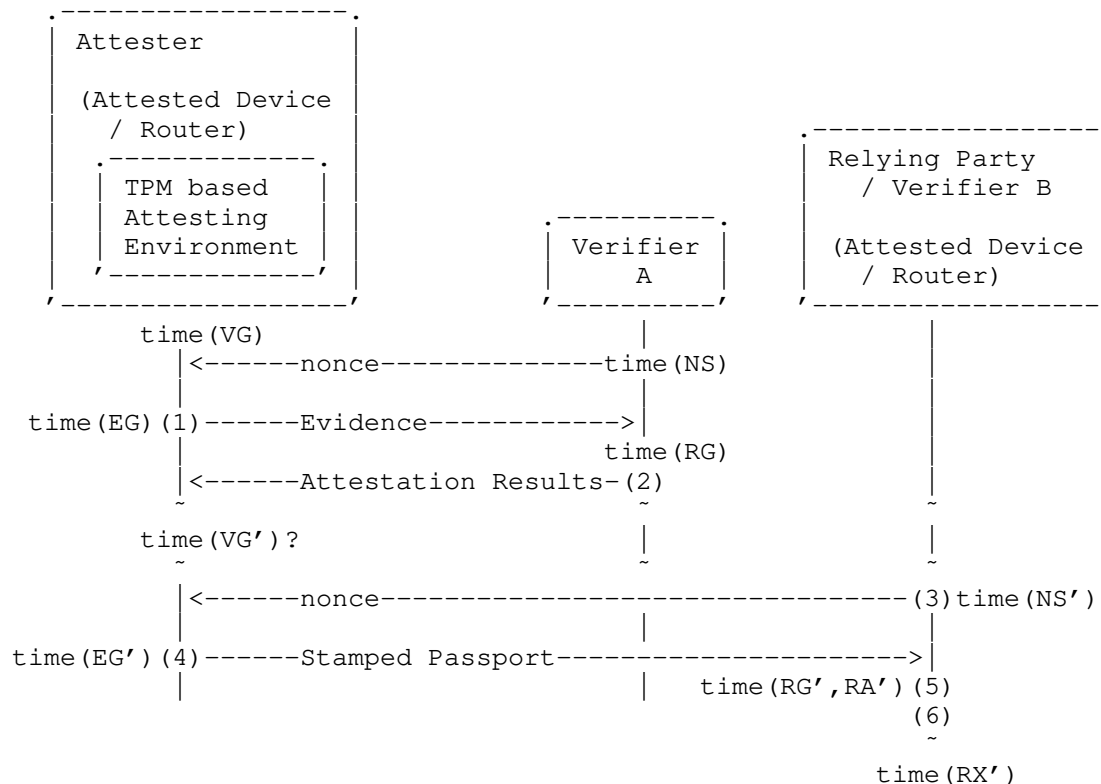


Figure 2: Trusted Path Timing

To summarize Figure 2 above, Evidence about a specific Attester is generated. Some subset of this evidence will be in the form of PCR quotes which are signed by a TPM that exists as the Attester's Attesting Environment. This Evidence will be deliberated to and appraised by Verifier A. Verifier A will then appraise the Attester and give it a Trustworthiness Vector. This Trustworthiness Vector is then signed by Verifier A and be returned as Attestation Results to the Attester. Later, when a request comes in from a Relying Party, the Attester assembles and returns a Stamped Passport. The Stamped Passport contains all the information necessary for Verifier B to

appraise the most recent Trustworthiness Vector of the Attester. Based on the Verifier B appraisal, the link will be included or not in a Trusted Topology maintained on the Relying Party.

More details on the mechanisms used in the construction, verification, and transmitting of the Stamped Passport are listed below. These numbers match to both the numbered steps of Figure 2 and numbered steps described in Section 3 of [attestation-results]:

4.2.1. Step 1

Evidence about and Attester is generated. A portion of this Evidence will include a PCR quote signed by a TPM private LDevID key that exists within the Attester's TPM based Attesting Environment. The Attester sends a signed TPM Quote which includes PCR measurements to Verifier A at time(EG).

There are two alternatives for Verifier A to acquire this signed Evidence:

- o Subscription to the <attestation> stream defined in [stream-subscription]. Note: this method is recommended as it will minimize the interval between when a PCR change is made in a TPM, and when the PCR change appraisal is incorporated within a subsequent Stamped Passport.
- o Periodic polling of RPC <tpm20-challenge-response-attestation> or the RPC <tpm12-challenge-response-attestation> which are defined in [RATS-YANG].

4.2.2. Step 2

Verifier A appraises the Evidence from Step 1. A portion of this appraisal process will follow the appraisal process flow described below. This appraisal process MUST be able to set at least the following set of Trustworthiness Claims from [attestation-results]: 'hardware', 'instance-identity', and 'executables'. The establishment of a Trustworthiness Vector uses the following Figure 3 logic on the Verifier:

Start: TPM Quote Received, log received, or appraisal timer expired
for the the Attesting network device.

Appraisal 0: set Trustworthiness Vector = Null

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

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

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

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

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

End

Figure 3: Verifier A Appraisal Flow

After the appraisal and generation of the Trustworthiness Vector, the following are assembled as the set of Attestation Results from this particular appraisal cycle:

(2.1) the Public Attestation Key which was used to validate the TPM Quote of Step 1. This is encoded by <public-key>, <public-key-format>, and <public-key-algorithm-type>.

(2.2) the appraised Trustworthiness Vector of the Attester as calculated in Figure 3

(2.3) the PCR state information from the TPM Quote of (1) plus the time information associated with the TPM Quote of (1). Specifically if the Attester has a TPM2, then the values of the TPM PCRs are included (i.e., <TPM2B_DIGEST>, <tpm20-hash-algo>, and <pcr-index>), as are the timing counters from the TPM (i.e., <clock>, <reset-counter>, <restart-counter>, and <safe>). Likewise if the Attester has a TPM1.2, the TPM PCR values of the <pcr-index> and <pcr-value> are included. Timing information comes from the Verifier itself via the <timestamp> object.

(2.4) a Verifier A signature across (2.1) through (2.3). This signature is encoded by <verifier-signature>, <verifier-key-algorithm-type>, and <verifier-signature-key-name>.

Immediately subsequent to each Verifier appraisal cycle of an Attester, these Attestation Results MUST be pushed to the Attesting Router. This is done via a datastore write to the following YANG model on the Attester. A YANG tree showing the relevant YANG objects is below. The YANG model describing each of these objects is described later in the document. Note however that although the YANG model shows the specific objects which are needed, the specific set of objects needs to be encoded in CDDL. This makes the payload going over TLS more efficient. Look for this encoding in a new version of the draft which is coming shortly.

```

module: ietf-trustworthiness-claims
+--rw attestation-results!
+--rw (tpm-specification-version)?
+--:(tpm20-attestation-results-cddl) {taa:tpm20}?
|   +--rw tpm20-attestation-results-cddl
|   |   +--rw trustworthiness-vector
|   |   |   +--rw hardware?          hardware
|   |   |   +--rw instance-identity?  instance-identity
|   |   |   +--rw executables?        executables
|   |   |   +--rw configuration?      configuration
|   |   +--rw tpm20-pcr-selection* [tpm20-hash-algo]
|   |   |   +--rw tpm20-hash-algo      identityref
|   |   |   +--rw pcr-index*          tpm:pcr
|   |   +--rw TPM2B_DIGEST              binary
|   |   +--rw clock                      uint64
|   |   +--rw reset-counter              uint32
|   |   +--rw restart-counter            uint32
|   |   +--rw safe                      boolean
|   |   +--rw attester-certificate-name
|   |   |   tpm:certificate-name-ref
|   |   +--rw appraisal-timestamp
|   |   |   yang:date-and-time
|   |   +--rw verifier-algorithm-type    identityref
|   |   +--rw verifier-signature         binary
|   |   +--rw verifier-certificate-keystore-ref
|   |   |   tpm:certificate-name-ref
+--:(tpm12-attestation-results-cddl) {taa:tpm12}?
|   +--rw tpm12-attestation-results-cddl
|   |   +--rw trustworthiness-vector
|   |   |   +--rw hardware?          hardware
|   |   |   +--rw instance-identity?  instance-identity
|   |   |   +--rw executables?        executables
|   |   |   +--rw configuration?      configuration
|   |   +--rw pcr-index*              pcr
|   |   +--rw tpm12-pcr-value*         binary
|   |   +--rw tpm12-hash-algo          identityref
|   |   +--rw TPM12-quote-timestamp
|   |   |   yang:date-and-time
|   |   +--rw attester-certificate-name
|   |   |   tpm:certificate-name-ref
|   |   +--rw appraisal-timestamp
|   |   |   yang:date-and-time
|   |   +--rw verifier-algorithm-type    identityref
|   |   +--rw verifier-signature         binary
|   |   +--rw verifier-certificate-keystore-ref
|   |   |   tpm:certificate-name-ref

```

Figure 4: Attestation Results Tree

4.2.3. Step 3

At time(NS') some form of time-based freshness (such as a nonce or Epoch Handle [RATS-Interactions]) will be generated in a way which makes it available to the Relying Party. Soon after time(NS'), a Relying Party will make a Link Layer authentication request to an Attester via either [MACSEC] or [IEEE-802.1X]. This connection request MUST expect the return of [RFC3748] credentials from the Attester.

4.2.4. Step 4

Upon receipt of the Link Layer request from Step 3, a Stamped Passport is generated and sent to the Relying Party. The Stamped Passport MUST include the following:

(4.1) The Attestation Results from Step 2

(4.2) New signed, verifiably fresh PCR measurements based on a TPM quote at time(EG') which incorporates the freshness information known by the Relying Party from Step 3. If it is a nonce, the freshness information will have been delivered as part of the link layer connection request in Steps 3.

Stamped Passports contain following objects, defined in this document via YANG. A subsequent draft will convert the objects below into CDDL format so that the objects can efficiently be passed over EAP.

If an Attester includes a TPM2, these YANG objects are:

```

+---n tpm20-stamped-passport
+--ro attestation-results
|   +--ro trustworthiness-vector
|   |   +--ro hardware?          hardware
|   |   +--ro instance-identity? instance-identity
|   |   +--ro executables?       executables
|   |   +--ro configuration?     configuration
|   +--ro tpm20-pcr-selection* [tpm20-hash-algo]
|   |   +--ro tpm20-hash-algo    identityref
|   |   +--ro pcr-index*        tpm:pcr
|   +--ro TPM2B_DIGEST           binary
|   +--ro clock                  uint64
|   +--ro reset-counter          uint32
|   +--ro restart-counter        uint32
|   +--ro safe                   boolean
|   +--ro attester-certificate-name
|   |   tpm:certificate-name-ref
|   +--ro appraisal-timestamp
|   |   yang:date-and-time
|   +--ro verifier-algorithm-type identityref
|   +--ro verifier-signature     binary
|   +--ro verifier-certificate-keystore-ref
|   |   tpm:certificate-name-ref
+--ro tpm20-quote
|   +--ro TPMS_QUOTE_INFO        binary
|   +--ro quote-signature        binary

```

Figure 5: YANG Tree for a TPM2 Stamped Passport

Note that where a TPM2.0 is used, the PCR numbers and hash algorithms quoted in Step 1 MUST match the PCR numbers and hash algorithms quoted in this step.

And if the Attester is a TPM1.2, the YANG object are:

```

+---n tpm12-stamped-passport
+--ro attestation-results
|   +--ro trustworthiness-vector
|   |   +--ro hardware?          hardware
|   |   +--ro instance-identity? instance-identity
|   |   +--ro executables?       executables
|   |   +--ro configuration?     configuration
|   +--ro pcr-index*             pcr
|   +--ro tpm12-pcr-value*        binary
|   +--ro tpm12-hash-algo         identityref
|   +--ro TPM12-quote-timestamp
|   |   yang:date-and-time
|   +--ro attester-certificate-name
|   |   tpm:certificate-name-ref
|   +--ro appraisal-timestamp
|   |   yang:date-and-time
|   +--ro verifier-algorithm-type identityref
|   +--ro verifier-signature      binary
|   +--ro verifier-certificate-keystore-ref
|   |   tpm:certificate-name-ref
+--ro tpm12-quote
+--ro TPM_QUOTE2?    binary

```

Figure 6: YANG Tree for a TPM1.2 Stamped Passport

With either of these passport formats, if the TPM quote is verifiably fresh, then the state of the Attester can be appraised by a network peer.

Note that with [MACSEC] or [IEEE-802.1X], Step 3 plus Step 4 will repeat periodically independently of any subsequent iteration Steps 1 and Step 2. This allows for periodic reauthentication of the link layer in a way not bound to the updating of Verifier A's Attestation Results.

4.2.5. Step 5

Upon receipt of the Stamped Passport generated in Step 4, the Relying Party appraises this Stamped Passport as per its Appraisal Policy for Attestation Results. The result of this application will determine how the Stamped Passport will impact adjacencies within a Trusted Topology. The decision process is as follows:

- (5.1) Verify that (4.2) includes the freshness context from Step 3.
- (5.2) Use a local certificate to validate the signature (4.1).
- (5.3) Verify that the hash from (4.2) matches (4.1)

(5.4) Use the identity of (2.1) to validate the signature of (4.2).

(5.5) 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.

(5.6) Compare the time(EG) TPM state to the time(EG') TPM state

o If TPM2.0

1. If the <TPM2B_DIGEST>, <reset-counter>, <restart-counter> and <safe> are equal between the Attestation Results and the TPM Quote at time(EG') then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to Step 6.
2. If the <reset-counter>, <restart-counter> and <safe> are equal between the Attestation Results and the TPM Quote at time(EG'), and the <clock> object from time(EG') has not incremented by an unacceptable number of seconds since the Attestation Result, then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to Step 6.)
3. Assign the link a null Trustworthiness Vector.

o If TPM1.2

1. If the <pcr-index>'s and <tpm12-pcr-value>'s are equal between the Attestation Results and the TPM Quote at time(EG'), then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to step (6).
2. If the time hasn't incremented an unacceptable number of seconds from the Attestation Results <timestamp> and the system clock of the Relying Party, then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to step 6.)
3. Assign the link a null Trustworthiness Vector.

(5.7) Assemble the Verifier B Trustworthiness Vector

1. Copy Verifier A Trustworthiness Vector to Verifier B Trustworthiness Vector
2. Prune any Trustworthiness Claims the Relying Party doesn't accept from this Verifier.

4.2.6. Step 6

After the Trustworthiness Vector has been validated or reset, based on the link's Trustworthiness Vector, the Relying Party adjusts the link affinity of the corresponding ISIS [I-D.ietf-lsr-flex-algo] topology. ISIS will then replicate the link state across the IGP domain. Traffic will then avoid links which do not have a qualifying Trustworthiness Vector.

5. YANG Module

This YANG module imports modules from [RATS-YANG], [crypto-types] and [RFC6021].

```
<CODE BEGINS> ietf-trustworthiness-claims@2021-09-03.yang
module ietf-trustworthiness-claims {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-trustworthiness-claims";
  prefix tc;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-tcg-algs {
    prefix taa;
    reference
      "draft-ietf-rats-yang-tpm-charra";
  }
  import ietf-tpm-remote-attestation {
    prefix tpm;
    reference
      "draft-ietf-rats-yang-tpm-charra";
  }

  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) 2020 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; see the RFC itself for full legal notices.";

```
revision 2021-09-03 {  
  description  
    "Initial version.";  
  reference  
    "draft-voit-rats-trustworthy-path-routing";  
}
```

```
/*  
 * TYPEDEF  
 */
```

```
typedef trustworthiness-claim {  
  type int8;  
  description  
    "A Verifier asserted value designed to enable a common  
    understanding of a Verifier trustworthiness appraisal. The  
    value assignments for this 8 bit signed integer will follow  
    these guidelines:  
  
    Affirming: The Verifier affirms the Attester support for this  
    aspect of trustworthiness  
      - Values 1 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 63: A standards enumerated reason for the  
        warning.  
      - Values -33 to -64: A non-standard reason for the warning.  
  
    Contraindicated: The Verifier asserts the Attester is explicitly  
    untrustworthy in regard to this aspect.  
      - Values 64 to 127: A standards enumerated reason for the  
        contraindication.  
      - Values -65 to -128: A non-standard reason for the  
        contraindication.
```

None: The Verifier makes no assertions about this Trustworthiness Claim. The following values are reserved with the following meanings across all instances of trustworthiness-claim.

- Value 0: 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: An unexpected error occurred during the Verifier's appraisal processing. Note: while no claim is being made, the Relying Party MAY make a distinction between a Trustworthiness Claim with enumeration '-1', and no Trustworthiness Claim being provided.";

}

```
typedef hardware {
    type trustworthiness-claim;
    description
        "A Verifier has appraised any Attester hardware and firmware
        which are able to expose fingerprints of their identity and
        running code.

        The following are specific reserved values of hardware and
        the meanings of these reserved values:

        0: No assertion

        1: 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.

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

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

        -1:Unexpected error.";
```

}

```
typedef instance-identity {
    type trustworthiness-claim;
    description
        "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.)

The following are specific reserved values of instance-identity and the meanings of these reserved values:

0: No assertion

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

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

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

-1: Unexpected error.";

}

```
typedef executables {
    type trustworthiness-claim;
    description
        "A Verifier has appraised and evaluated relevant runtime files,
        scripts, and/or other objects which have been loaded into the
        Target environment's memory.
```

The following are specific reserved values of executables and the meanings of these reserved values:

0: No assertion

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

2: 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.

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

```
-1:Unexpected error.";  
}
```

```
typedef configuration {  
  type trustworthiness-claim;  
  description  
    "A Verifier has appraised an Attester's configuration, and is  
    able to make conclusions regarding the exposure of known  
    vulnerabilities.
```

The following are specific reserved values of configuration and the meanings of these reserved values:

0: No assertion

1: The configuration is a known and approved config

2: The configuration includes or exposes no known vulnerabilities

32:The configuration includes or exposes known vulnerabilities

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

```
-1:Unexpected error.";  
}
```

```
/*  
 * GROUPINGS  
 */
```

```
grouping trustworthiness-vector {  
  description  
    "Allows the inclusion of a Trustworthiness Vector into  
    other constructs.";  
  container trustworthiness-vector {  
    description  
      "One or more Trustworthiness Claims assigned which expose the  
      Verifiers evaluation of the Evidence associated with the  
      AIK which signed as associated TPM Quote.";  
    leaf hardware {  
      type hardware;
```

```
        description
            "An 8 bit signed integter encoded per the typedef.";
    }
    leaf instance-identity {
        type instance-identity;
        description
            "An 8 bit signed integter encoded per the typedef.";
    }
    leaf executables {
        type executables;
        description
            "An 8 bit signed integter encoded per the typedef.";
    }
    leaf configuration {
        type configuration;
        description
            "An 8 bit signed integter encoded per the typedef.";
    }
}

grouping verifier-evidence {
    description
        "Evidence generated by the Verifier.";
    leaf appraisal-timestamp {
        type yang:date-and-time;
        mandatory true;
        description
            "The timestamp of the Verifier's appraisal. This can be used
            by a Relying Party to determine the freshness of the
            attestation results.";
    }
    leaf verifier-algorithm-type {
        type identityref {
            base taa:asymmetric;
        }
        mandatory true;
        description
            "Platform asymmetric algorithm used in the Verifier signature
            process.";
    }
    leaf verifier-signature {
        type binary;
        mandatory true;
        description
            "Signature of the Verifier across all the current objects in
            the attestation-results container except for 'verifier-
            signature' and 'verifier-certificate-keystore-ref'.
```

```
        This assumes CDDL encoding of the objects in the current
        order of this YANG model.";
    }
    leaf verifier-certificate-keystore-ref {
        type tpm:certificate-name-ref;
        mandatory true;
        description
            "A reference to a specific certificate to an asymmetric key
            in the Keystore for the Verifier which can be used to validate
            the 'verifier-signature'. Note that the
            'name' reference must be globally unique so that it can be
            read by the Relying Party in a way which identifies a
            specific Verifier.";
    }
}

grouping tpm20-cddl-attestation-results {
    description
        "Elements combined into a CDDL representation for TPM2.0.";
    uses trustworthiness-vector;
    list tpm20-pcr-selection {
        key "tpm20-hash-algo";
        description
            "Specifies the list of PCRs and Hash Algorithms used by the
            Verifier.";
        reference
            "https://www.trustedcomputinggroup.org/wp-content/uploads/
            TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.9.7";
        uses tpm:tpm20-hash-algo;
        leaf-list pcr-index {
            type tpm:pcr;
            description
                "The numbers of the PCRs associated with the TPM2B_DIGEST.";
        }
    }
    leaf TPM2B_DIGEST {
        mandatory true;
        type binary;
        description
            "A hash of the latest PCR values (and the hash algorithm used)
            which have been returned from a Verifier for the selected PCRs
            identified within TPML_PCR_SELECTION.";
        reference
            "https://www.trustedcomputinggroup.org/wp-content/uploads/
            TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.12.1";
    }
    leaf clock {
        mandatory true;
    }
}
```

```
type uint64;
description
    "Clock is a monotonically increasing counter that advances
    whenever power is applied to a TPM2. The value of Clock is
    incremented each millisecond.";
reference
    "https://www.trustedcomputinggroup.org/wp-content/uploads/
    TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.11.2";
}
leaf reset-counter {
    mandatory true;
    type uint32;
    description
        "This counter increments on each TPM Reset. The most common
        TPM Reset would be due to a hardware power cycle.";
    reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.11.3";
}
leaf restart-counter {
    mandatory true;
    type uint32;
    description
        "This counter shall increment by one for each TPM Restart or
        TPM Resume. The restartCount shall be reset to zero on a TPM
        Reset.";
    reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.11.4";
}
leaf safe {
    mandatory true;
    type boolean;
    description
        "This parameter is set to YES when the value reported in Clock
        is guaranteed to be unique for the current Owner. It is set to
        NO when the value of Clock may have been reported in a previous
        attestation or access.";
    reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.11.5";
}
leaf attester-certificate-name {
    mandatory true;
    description
        "The Attester is associated with these results.";
    type tpm:certificate-name-ref;
}
```

```
    uses verifier-evidence;
  }

grouping tpm12-cddl-attestation-results {
  description
    "Elements combined into a CDDL representation for TPM1.2.";
  uses trustworthiness-vector;
  uses tpm:tpm12-pcr-selection;
  leaf-list tpm12-pcr-value {
    type binary;
    description
      "The list of TPM_PCRVALUES from each PCR selected in sequence
      of tpm12-pcr-selection.";
    reference
      "https://www.trustedcomputinggroup.org/wp-content/uploads/
      TPM-Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf
      Section 10.9.7";
  }
  uses tpm:tpm12-hash-algo {
    refine "tpm12-hash-algo" {
      mandatory true;
    }
  }
  leaf TPM12-quote-timestamp {
    type yang:date-and-time;
    mandatory true;
    description
      "The timestamp for when the indicator of freshness (such as a
      nonce) was generated. This is the indicator of freshness
      which was used in the generation of the TPM1.2 quote. This
      timestamp can be used by a Relying Party to determine the
      freshness of the attestation results.";
  }
  leaf attester-certificate-name {
    mandatory true;
    description
      "The Attester is associated with these results.";
    type tpm:certificate-name-ref;
  }
  uses verifier-evidence;
}

/*
 * NOTIFICATIONS
 */

notification tpm20-stamped-passport {
  description
```



```
    "The augmentation of the most recent Attestation Results
    delivered from a Verifier with a TPM2.0 Quote.";
  container attestation-results {
    description
      "The latest Verifier delivered Attestation Results.";
    uses tpm20-cddl-attestation-results;
  }
  container tpm20-quote {
    description
      "The TPM2.0 quote delivered in response to a connectivity
      request.";
    leaf TPMS_QUOTE_INFO {
      type binary;
      mandatory true;
      description
        "A hash of the latest PCR values (and the hash algorithm
        used) which have been returned from a Verifier for the
        selected PCRs and Hash Algorithms.";
      reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.12.1";
    }
    leaf quote-signature {
      type binary;
      mandatory true;
      description
        "Quote signature returned by TPM Quote. The signature was
        generated using the key associated with the
        certificate 'name'.";
      reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 11.2.1";
    }
  }
}

notification tpm12-stamped-passport {
  description
    "The augmentation of the most recent Attestation Results
    delivered from a Verifier with a TPM1.2 Quote.";
  container attestation-results {
    description
      "The latest Verifier delivered Attestation Results.";
    uses tpm12-cddl-attestation-results;
  }
  container tpm12-quote {
    description
      "The TPM1.2 quote delivered in response to a connectivity
```

```
    request.";
  leaf TPM_QUOTE2 {
    type binary;
    description
      "Result of a TPM1.2 Quote2 operation. This includes PCRs,
       signatures, locality, the provided nonce and other data which
       can be further parsed to appraise the Attester.";
    reference
      "TPM1.2 commands rev116 July 2007, Section 16.5";
  }
}

/*
 * DATA NODES
 */

container attestation-results {
  presence
    "Indicates that Verifier has appraised the security posture of
     the Attester, and returned the results within this container.";
  description
    "Retains the most recent Attestation Results for this Attester.
     It must only be written by a Verifier which is to be trusted by a
     Relying Party.";

  choice tpm-specification-version {
    description
      "Identifies the cryptoprocessor API set which drove the
       Attestation Results.";
    case tpm20-attestation-results-cddl {
      if-feature "taa:tpm20";
      description
        "Attestation Results which are returned from the
         evaluation of Evidence which includes a TPM2 quote.";
      container tpm20-attestation-results-cddl {
        description
          "Attestation Results which are returned from the
           evaluation of Evidence which includes a TPM2 quote.";
        uses tpm20-cddl-attestation-results;
      }
    }
    case tpm12-attestation-results-cddl {
      if-feature "taa:tpm12";
      description
        "Attestation Results which are returned from the
         evaluation of Evidence which includes a TPM1.2 quote.";
      container tpm12-attestation-results-cddl {
```

```
        description
          "Attestation Results which are returned from the
           evaluation of Evidence which includes a TPM1.2 quote.";
          uses tpml2-cddl-attestation-results;
        }
      }
    }
  }
}
<CODE ENDS>
```

6. Security Considerations

Verifiers are limited to the Evidence available for appraisal from a Router. Although the state of the art is improving, some exploits may not be visible via Evidence.

Only security measurements which are placed into PCRs are capable of being exposed via TPM Quote at time(EG').

Successful attacks on an Verifier have the potential of affecting traffic on the Trusted Topology.

For Trusted Path Routing, links which are part of the FlexAlgo are visible across the entire IGP domain. Therefore a compromised device will know when it is being bypassed.

Access control for the objects in Figure 4 should be tightly controlled so that it becomes difficult for the Stamped Passport to become a denial of service vector.

7. References

7.1. Normative References

- [attestation-results]
"Attestation Results for Connectivity", April 2021,
<[https://tools.ietf.org/html/
draft-voit-rats-attestation-results-00](https://tools.ietf.org/html/draft-voit-rats-attestation-results-00)>.
- [crypto-types]
"Common YANG Data Types for Cryptography", May 2020,
<[https://datatracker.ietf.org/doc/
draft-ietf-netconf-crypto-types/](https://datatracker.ietf.org/doc/draft-ietf-netconf-crypto-types/)>.

- [RATS-Arch] "Remote Attestation Procedures Architecture", March 2020, <<https://tools.ietf.org/html/draft-ietf-rats-architecture-02>>.
- [RATS-YANG] "A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs", June 2020, <<https://datatracker.ietf.org/doc/draft-ietf-rats-yang-tpm-charra/>>.
- [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>>.
- [RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <<https://www.rfc-editor.org/info/rfc6021>>.
- [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>>.
- [TPM1.2] TCG, ., "TPM 1.2 Main Specification", October 2003, <<https://trustedcomputinggroup.org/resource/tpm-main-specification/>>.
- [TPM2.0] TCG, ., "TPM 2.0 Library Specification", March 2013, <<https://trustedcomputinggroup.org/resource/tpm-library-specification/>>.

7.2. Informative References

- [I-D.ietf-lsr-flex-algo] Cisco Systems, Juniper Networks, Cisco Systems, Cisco Systems, and Edward Jones, "IGP Flexible Algorithm", draft-ietf-lsr-flex-algo-17 (work in progress), July 2021.
- [IEEE-802.1X] Parsons, G., "802.1AE: MAC Security (MACsec)", January 2020, <https://standards.ieee.org/standard/802_1X-2010.html>.
- [MACSEC] Seaman, M., "802.1AE: MAC Security (MACsec)", January 2006, <<https://1.ieee802.org/security/802-1ae/>>.

[RATS-Device]

"Network Device Remote Integrity Verification", n.d.,
<[https://datatracker.ietf.org/doc/
draft-ietf-rats-tpm-based-network-device-attest](https://datatracker.ietf.org/doc/draft-ietf-rats-tpm-based-network-device-attest)>.

[RATS-Interactions]

"Reference Interaction Models for Remote Attestation
Procedures", June 2020, <[https://ietf-rats.github.io/
draft-birkholz-rats-reference-interaction-model/draft-
birkholz-rats-reference-interaction-model.html#section-7](https://ietf-rats.github.io/draft-birkholz-rats-reference-interaction-model/draft-birkholz-rats-reference-interaction-model.html#section-7)>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H.
Levkowetz, Ed., "Extensible Authentication Protocol
(EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004,
<<https://www.rfc-editor.org/info/rfc3748>>.

[stream-subscription]

"Attestation Event Stream Subscription", June 2020,
<[https://datatracker.ietf.org/doc/
draft-birkholz-rats-network-device-subscription](https://datatracker.ietf.org/doc/draft-birkholz-rats-network-device-subscription)>.

Appendix A. Acknowledgements

Peter Psenak, Shwetha Bhandari, Adwaith Gautham, Annu Singh, Sujal Sheth, Nancy Cam Winget, and Ned Smith.

Appendix B. Change Log

[THIS SECTION TO BE REMOVED BY THE RFC EDITOR.]

v03-v04

- o Moved in concert with draft-voit-rats-attestation-results as Trustworthiness Claims became 8 bit signed integers.

v02-v03

- o Integrated [attestation-results] as prerequisite context.
- o Totally rearranged content. But there were not meaningful process changes.
- o Redid YANG model, and highlighted CDDL needs.

v01-v02

- o Minor tweaks such as renaming and removal of a few trustworthiness-claims

v00-v01

- o Minor tweaks

v02-v00 of draft-voit-rats-trustworthy-path-routing-00

- o file rename was due to an IETF tool submission glitch
- o The Attester's AIK is included within the Stamped Passport. This eliminates the need to provision to AIK certificate on the Relying Party.
- o Removed Centralized variant
- o Added timing diagram, and moved content around to match

v01-v02 of draft-voit-rats-trusted-path-routing

- o Extracted the attestation stream, and placed into draft-birkholz-rats-network-device-subscription
- o Introduced the Trustworthiness Vector

v00-v01 of draft-voit-rats-trusted-path-routing

- o Move all FlexAlgo terminology to allow passport definition to be more generic.
- o Edited Figure 1 so that (4) points to the egress router.
- o Added text freshness mechanisms, and articulated configured subscription support.
- o Minor YANG model clarifications.
- o Added a few open questions which Frank thinks interesting to work.

Appendix C. Open Questions

(1) When there is no available Trusted Topology?

Do we need functional requirements on how to handle traffic to/from Sensitive Subnets when no Trusted Topology exists between IGP edges? The network typically can make this unnecessary. For example it is possible to construct a local IPSec tunnel to make untrusted devices appear as Transparently-Transited Devices. This way Secure Subnets could be tunneled between FlexAlgo nodes where an end-to-end path

doesn't currently exist. However there still is a corner case where all IGP egress points are not considered sufficiently trustworthy.

(2) Extension of the Stamped Passport?

Format of the reference to the 'verifier-certificate-name' based on WG desire to include more information in the Stamped Passport. Also we need to make sure that the keystore referenced names are globally unique, else we will need to include a node name in the object set.

(3) Encoding of objects in CDDL. A Verifier will want to sign encoded objects rather than YANG structures. It is most efficient to encode the Attestation Results once on the Verifier, and push these down via a YANG model to the Attester.

Authors' Addresses

Eric Voit
Cisco Systems, Inc.
8135 Maple Lawn Blvd
Fulton, Maryland 20759
USA

Email: evoit@cisco.com

Chennakesava Reddy Gaddam
Cisco Systems, Inc.
Cessna Business Park, Kadubeesanahalli
Bangalore, Karnataka 560103
India

Email: chgaddam@cisco.com

Guy C. Fedorkow
Juniper Networks
10 Technology Park Drive
Westford, Massachusetts 01886
USA

Email: gfedorkow@juniper.net

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2022

E. Voit
C. Gaddam
Cisco
G. Fedorkow
Juniper
H. Birkholz
Fraunhofer SIT
M. Chen
China Mobile
2 March 2022

Trusted Path Routing
draft-voit-rats-trustworthy-path-routing-05

Abstract

There are end-users who believe encryption technologies like IPSec alone are insufficient to protect the confidentiality of their highly sensitive traffic flows. These end-users want their flows to traverse devices which have been freshly appraised and verified for trustworthiness. This specification describes Trusted Path Routing. Trusted Path Routing protects sensitive flows as they transit a network by forwarding traffic to/from sensitive subnets across network devices recently appraised as trustworthy.

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 3 September 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
2. Terminology	3
2.1. Terms	3
2.2. Requirements Notation	4
3. Implementation Prerequisites	4
4. End-to-end Solution	5
4.1. Network Topology Assembly	5
4.2. Attestation Information Flows	6
4.2.1. Step 1	9
4.2.2. Step 2	9
4.2.3. Step 3	13
4.2.4. Step 4	13
4.2.5. Step 5	15
4.2.6. Step 6	17
5. YANG Module	17
6. Security Considerations	28
7. References	28
7.1. Normative References	28
7.2. Informative References	29
Appendix A. Acknowledgements	30
Appendix B. Change Log	30
Appendix C. Open Questions	32
Authors' Addresses	32

1. Introduction

There are end-users who believe encryption technologies like IPSec alone are insufficient to protect the confidentiality of their highly sensitive traffic flows. These customers want their highly sensitive flows to be transported over only network devices recently verified as trustworthy.

By using a router's embedded TPM based cryptoprocessors in conjunction with the Remote Attestation context established by [attestation-results], a network provider can identify potentially compromised devices as well as potentially exploitable (or even exploited) vulnerabilities. Using this knowledge, it is then possible to redirect sensitive flows around these devices while other remediations are potentially considered by Network Operations.

Trusted Path Routing allows the establishing Trusted Topologies which only include trust-verified network devices. Membership in a Trusted Topology is established and maintained via an exchange of Stamped Passports at the link layer between peering network devices. As links to Attesting Devices are appraised as meeting at least a minimum set of formally defined Trustworthiness Claims, the links are then included as members of this Trusted Topology. Routing protocols are then used to propagate topology state throughout a network.

IP Packets to and from end-user designated Sensitive Subnets are then forwarded into this Trusted Topology at each network boundary. This is done by an end user identifying sensitive IP subnets where flows with applications using these IP subnets need enhanced privacy guarantees. Trusted Path Routing passes flows to/from these Sensitive Subnets over a Trusted Topology able to meet these guarantees. The Trusted Topology itself consists of the interconnection of network devices where each potentially transited device has been verified as achieving a specific set of Trustworthiness Claims during its most recent trustworthiness appraisal. Interesting sets of Trustworthiness Claims might be marketed to end-users in the following ways:

- * all transited devices have booted with known hardware and firmware
- * all transited devices are from a specific set of vendors and are running known software containing the latest patches
- * no guarantees provided

2. Terminology

2.1. Terms

The following terms are imported from [RATS-Arch]: Attester, Evidence, Passport, Relying Party, and Verifier.

The following terms are imported from [attestation-results]: Trustworthiness Claim, Trustworthiness Vector, AR-augmented Evidence

Newly defined terms for this document:

Attested Device -- a network connected Attester where a Verifier's most recent appraisal of Evidence has returned a Trustworthiness Vector.

Stamped Passport -- AR-augmented Evidence which can take two forms. First if the Attester uses a TPM2, the the Verifier Proof-of-Freshness includes the <clock>, <reset-counter>, <restart-counter> and <safe> objects from a recent TPM2 quote made by that Attester, and the Relying Party Proof-of-Freshness is returned along with the timeticks as objects embedded within the most recent TPM quote signed by the same TPM2. Second, if the Attester uses a TPM1.2: the Verifier Proof-of-Freshness includes a global timestamp from that Verifier, and the Relying Party Proof-of-Freshness is embedded within a more recent TPM quote signed by the same TPM Attesting Environment.

Sensitive Subnet -- an IP address range where IP packets to or from that range desire confidentially guarantees beyond those of non-identified subnets. In practice, flows to or from a Sensitive Subnet must only have their IP headers and encapsulated payloads accessible/visible only by Attested Devices supporting one or more Trustworthiness Vectors.

Transparently-Transited Device -- a network device within an network domain where any packets originally passed into that network domain are completely opaque on that network device at Layer 3 and above.

Trusted Topology -- a topology which includes only Attested Devices and Transparently-Transited Devices.

2.2. 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. Implementation Prerequisites

The specification is a valid instance of [attestation-results]. This specification works under the following protocol and preconfiguration prerequisite assumptions:

- * All Attested Devices support the TPM remote attestation profile as laid out in [RATS-Device], and include either [TPM2.0] or [TPM1.2].

- * One or more Verifier A's as defined in [attestation-results] 'Interaction Model' continuously appraise each of the Attested Devices in a network domain, and these Verifiers return the Attestation Results back to each originating Attested Device.
- * The Attested Devices are connected via link layer protocols such as [MACSEC] or [IEEE-802.1X].
- * Each Attester can pass a Stamped Passport to a Relying Party / Verifier B as defined in [attestation-results] 'Interaction Model' within [RFC3748] over that link layer protocol.
- * A Trusted Topology such as [I-D.ietf-lsr-flex-algo] exists in an IGP domain for the forwarding of Sensitive Subnet traffic. This Topology will carry traffic across a set of Attested Devices which currently meet at a defined set of Trustworthiness Vectors.
- * A Relying Party is able to use mechanisms such as [I-D.ietf-lsr-flex-algo]'s affinity to include/exclude links as part of the Trusted Topology based on the appraisal of a Stamped Passport.
- * Customer designated Sensitive Subnets and their requested Trustworthiness Vectors have been identified and associated with external interfaces to/from Attested Devices at the edge of a network. Traffic to a Sensitive Subnet can be passed into the Trusted Topology by the Attested Device.
- * Relying Party/Verifier B trusts information signed by Verifier A. Verifier B has also been pre-provisioned with certificates or public keys necessary to confirm that Stamped Passports came from Verifier A.

4. End-to-end Solution

4.1. Network Topology Assembly

To be included in a Trusted Topology, Stamped Passports are shared between Attested Devices (such as routers) as part of link layer authentication. Upon receiving and appraising the Stamped Passport during the link layer authentication phase, the Relying Party Attested Device decides if this link should be added as an active adjacency for a particular Trusted Topology. In Figure 1 below, this might be done by applying an Appraisal Policy for Attestation Results. The policy within each device might specify the evaluation of a 'hardware' claim as defined in [attestation-results], Section 2.3.4. With the appraisal, an Attesting Device be most recently appraised with the 'hardware' Trustworthiness Claim in the

'affirming' range. If Attested Device has been appraised outside that range, it would not become part of the Trustworthy Topology.

When enough links have been successfully added, the Trusted Topology will support edge-to-edge forwarding as routing protocols flood the adjacency information across the network domain.

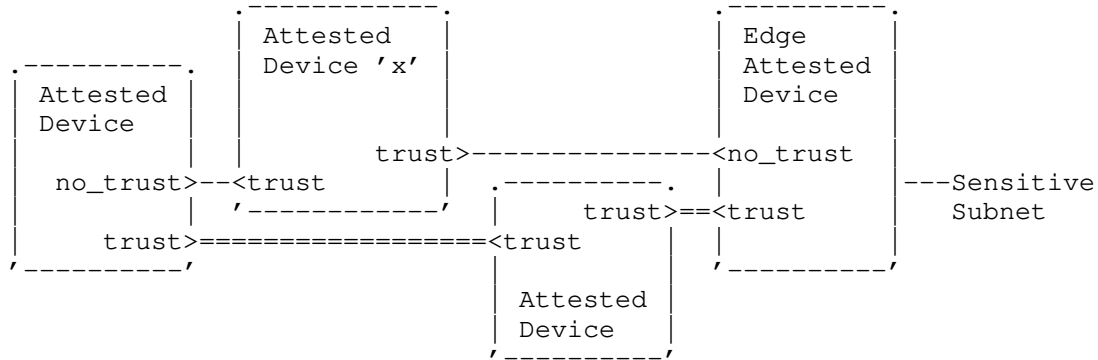


Figure 1: Trusted Path Topology Assembly

As the process described above repeats over time across the set of links within a network domain, Trusted Topologies can be extended and maintained. Traffic to and from Sensitive Subnets is then identified at the edges of the network domain and passed into this Trusted Topology. Traffic exchanged with Sensitive Subnets can then be forwarded across that Trusted Topology from all edges of the network domain. After the initial Trusted Topology establishment, new and existing devices will continue to provide incremental Stamped Passports. As each link is added/removed from the Trusted Topology, the topology will adjust itself accordingly.

Ultimately from an operator and users point of view, the delivered network will be more secure and therefore the service provided more valuable. As network operators attach great importance to the innate security of links, also delivering security for transited network and networking devices will also prove valuable.

4.2. Attestation Information Flows

Critical to the establishment and maintenance of a Trusted Topology is the Stamped Passport. A Stamped Passport is comprised of Evidence from both an Attester and a Verifier. A Stamped Passport is a valid type of AR-augmented evidence as described in [attestation-results].

Stamped Passports are exchanged between adjacent network devices over a link layer protocols like 802.1x or MACSEC. As both sides of a link may need might need to appraise the other, independent Stamped Passports will often be transmitted from either side of the link. Additionally, as link layer protocols will continuously re-authenticate the link, this allows for fresh Stamped Passports to be constantly appraised by either side of the connection.

Each Stamped Passport will include the most recent Verifier provided Attestation Results, as well as the most recent TPM Quote for that Attester. Upon receiving this information as part of link layer authentication, the Relying Party Router appraises the results and decides if this link should be added to a Trusted Topology.

Figure 2 describes this flow of information using the time definitions described in [RATS-Arch], and the information flows defined in Section 7 of [RATS-Interactions]. This figure is also a valid embodiment of the "Interaction Model" described within [attestation-results]. (Note that the Relying Party must also be an Attested Device in order to attract Sensitive Subnet traffic which may flow from the Attester.)

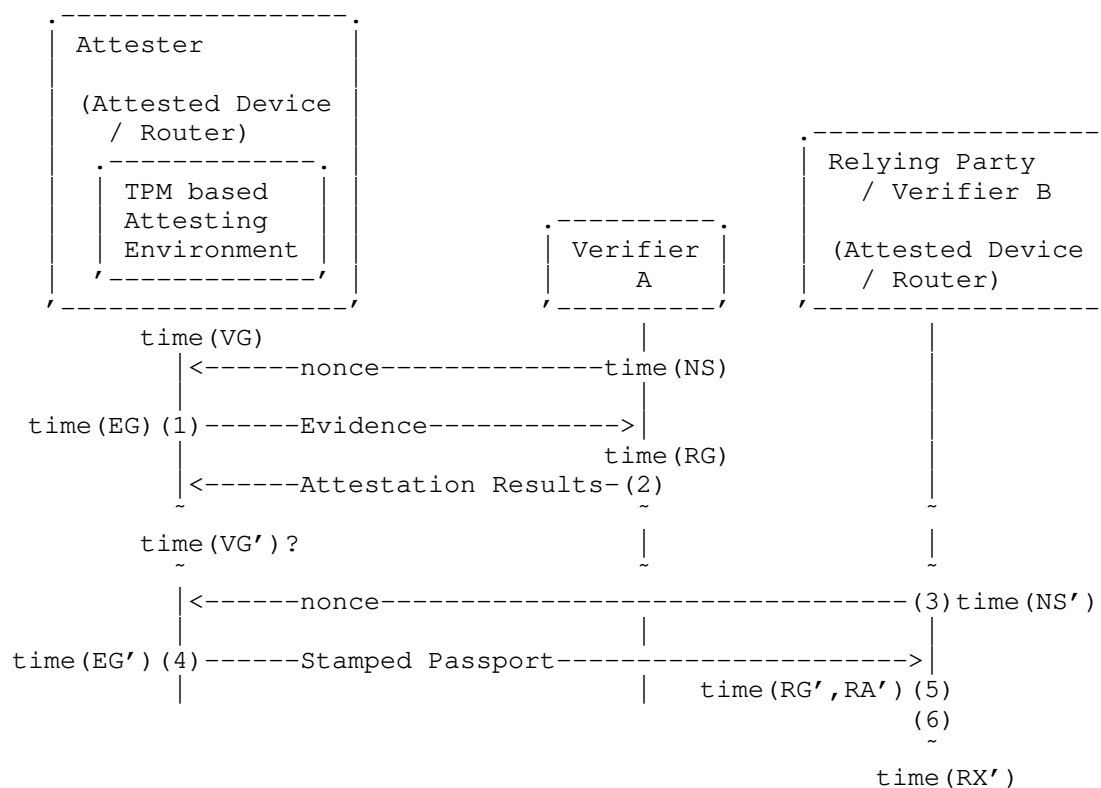


Figure 2: Trusted Path Timing

To summarize Figure 2 above, Evidence about a specific Attester is generated. Some subset of this evidence will be in the form of PCR quotes which are signed by a TPM that exists as the Attester's Attesting Environment. This Evidence will be deliberated to and appraised by Verifier A. Verifier A will then appraise the Attester and give it a Trustworthiness Vector. This Trustworthiness Vector is then signed by Verifier A and be returned as Attestation Results to the Attester. Later, when a request comes in from a Relying Party, the Attester assembles and returns a Stamped Passport. The Stamped Passport contains all the information necessary for Verifier B to appraise the most recent Trustworthiness Vector of the Attester. Based on the Verifier B appraisal, the link will be included or not in a Trusted Topology maintained on the Relying Party.

More details on the mechanisms used in the construction, verification, and transmitting of the Stamped Passport are listed below. These numbers match to both the numbered steps of Figure 2 and numbered steps described in Section 3 of [attestation-results]:

4.2.1. Step 1

Evidence about and Attester is generated. A portion of this Evidence will include a PCR quote signed by a TPM private LDevID key that exists within the Attester's TPM based Attesting Environment. The Attester sends a signed TPM Quote which includes PCR measurements to Verifier A at time(EG).

There are two alternatives for Verifier A to acquire this signed Evidence:

- * Subscription to the <attestation> stream defined in [stream-subscription]. Note: this method is recommended as it will minimize the interval between when a PCR change is made in a TPM, and when the PCR change appraisal is incorporated within a subsequent Stamped Passport.
- * Periodic polling of RPC <tpm20-challenge-response-attestation> or the RPC <tpm12-challenge-response-attestation> which are defined in [RATS-YANG].

4.2.2. Step 2

Verifier A appraises the Evidence from Step 1. A portion of this appraisal process will follow the appraisal process flow described below. This appraisal process MUST be able to set at least the following set of Trustworthiness Claims from [attestation-results]: 'hardware', 'instance-identity', and 'executables'. The establishment of a Trustworthiness Vector uses the following Figure 3 logic on the Verifier:

Start: TPM Quote Received, log received, or appraisal timer expired
for the the Attesting network device.

Appraisal 0: set Trustworthiness Vector = Null

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

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

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

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

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

End

Figure 3: Verifier A Appraisal Flow

After the appraisal and generation of the Trustworthiness Vector, the following are assembled as the set of Attestation Results from this particular appraisal cycle:

(2.1) the Public Attestation Key which was used to validate the TPM Quote of Step 1. This is encoded by <public-key>, <public-key-format>, and <public-key-algorithm-type>.

(2.2) the appraised Trustworthiness Vector of the Attester as calculated in Figure 3

(2.3) the PCR state information from the TPM Quote of (1) plus the time information associated with the TPM Quote of (1). Specifically if the Attester has a TPM2, then the values of the TPM PCRs are included (i.e., <TPM2B_DIGEST>, <tpm20-hash-algo>, and <pcr-index>), as are the timing counters from the TPM (i.e., <clock>, <reset-counter>, <restart-counter>, and <safe>). Likewise if the Attester has a TPM1.2, the TPM PCR values of the <pcr-index> and <pcr-value> are included. Timing information comes from the Verifier itself via the <timestamp> object.

(2.4) a Verifier A signature across (2.1) through (2.3). This signature is encoded by <verifier-signature>, <verifier-key-algorithm-type>, and <verifier-signature-key-name>.

Immediately subsequent to each Verifier appraisal cycle of an Attester, these Attestation Results MUST be pushed to the Attesting Router. This is done via a datastore write to the following YANG model on the Attester. A YANG tree showing the relevant YANG objects is below. The YANG model describing each of these objects is described later in the document. Note however that although the YANG model shows the specific objects which are needed, the specific set of objects needs to be encoded in CDDL. This makes the payload going over TLS more efficient. Look for this encoding in a new version of the draft which is pending.

```

module: ietf-trustworthiness-claims
+--rw attestation-results!
  +--rw (tpm-specification-version)?
    +--:(tpm20-attestation-results-cddl) {taa:tpm20}?
      +--rw tpm20-attestation-results-cddl
        +--rw trustworthiness-vector
          +--rw hardware? hardware
          +--rw instance-identity? instance-identity
          +--rw executables? executables
          +--rw configuration? configuration
          +--rw tpm20-pcr-selection* [tpm20-hash-algo]
          +--rw tpm20-hash-algo identityref
          +--rw pcr-index* tpm:pcr
          +--rw TPM2B_DIGEST binary
          +--rw clock uint64
          +--rw reset-counter uint32
          +--rw restart-counter uint32
          +--rw safe boolean
          +--rw attester-certificate-name
          | tpm:certificate-name-ref
          +--rw appraisal-timestamp
          | yang:date-and-time
          +--rw verifier-algorithm-type identityref
          +--rw verifier-signature binary
          +--rw verifier-certificate-keystore-ref
          | tpm:certificate-name-ref
    +--:(tpm12-attestation-results-cddl) {taa:tpm12}?
      +--rw tpm12-attestation-results-cddl
        +--rw trustworthiness-vector
          +--rw hardware? hardware
          +--rw instance-identity? instance-identity
          +--rw executables? executables
          +--rw configuration? configuration
          +--rw pcr-index* pcr
          +--rw tpm12-pcr-value* binary
          +--rw tpm12-hash-algo identityref
          +--rw TPM12-quote-timestamp
          | yang:date-and-time
          +--rw attester-certificate-name
          | tpm:certificate-name-ref
          +--rw appraisal-timestamp
          | yang:date-and-time
          +--rw verifier-algorithm-type identityref
          +--rw verifier-signature binary
          +--rw verifier-certificate-keystore-ref
          | tpm:certificate-name-ref

```

Figure 4: Attestation Results Tree

4.2.3. Step 3

At time(NS') some form of time-based freshness (such as a nonce or Epoch Handle [RATS-Interactions]) will be generated in a way which makes it available to the Relying Party. Soon after time(NS'), a Relying Party will make a Link Layer authentication request to an Attester via a either [MACSEC] or [IEEE-802.1X]. This connection request MUST expect the return of [RFC3748] credentials from the Attester.

4.2.4. Step 4

Upon receipt of the Link Layer request from Step 3, a Stamped Passport is generated and sent to the Relying Party. The Stamped Passport MUST include the following:

(4.1) The Attestation Results from Step 2

(4.2) New signed, verifiably fresh PCR measurements based on a TPM quote at time(EG') which incorporates the freshness information known by the Relying Party from Step 3. If it is a nonce, the freshness information will have been delivered as part of the link layer connection request in Steps 3.

Stamped Passports contain following objects, defined in this document via YANG. A subsequent draft will convert the objects below into CDDL format so that the objects can efficiently be passed over EAP.

If an Attester includes a TPM2, these YANG objects are:

```

+---n tpm20-stamped-passport
  +--ro attestation-results
    +--ro trustworthiness-vector
      +--ro hardware?          hardware
      +--ro instance-identity? instance-identity
      +--ro executables?      executables
      +--ro configuration?    configuration
    +--ro tpm20-pcr-selection* [tpm20-hash-algo]
      +--ro tpm20-hash-algo    identityref
      +--ro pcr-index*        tpm:pcr
    +--ro TPM2B_DIGEST          binary
    +--ro clock                 uint64
    +--ro reset-counter         uint32
    +--ro restart-counter       uint32
    +--ro safe                  boolean
    +--ro attester-certificate-name
      | tpm:certificate-name-ref
    +--ro appraisal-timestamp
      | yang:date-and-time
    +--ro verifier-algorithm-type identityref
    +--ro verifier-signature      binary
    +--ro verifier-certificate-keystore-ref
      | tpm:certificate-name-ref
  +--ro tpm20-quote
    +--ro TPMS_QUOTE_INFO        binary
    +--ro quote-signature        binary

```

Figure 5: YANG Tree for a TPM2 Stamped Passport

Note that where a TPM2.0 is used, the PCR numbers and hash algorithms quoted in Step 1 MUST match the PCR numbers and hash algorithms quoted in this step.

And if the Attester is a TPM1.2, the YANG object are:

```

+---n tpm12-stamped-passport
+--ro attestation-results
|   +--ro trustworthiness-vector
|   |   +--ro hardware?          hardware
|   |   +--ro instance-identity? instance-identity
|   |   +--ro executables?       executables
|   |   +--ro configuration?     configuration
|   +--ro pcr-index*              pcr
|   +--ro tpm12-pcr-value*         binary
|   +--ro tpm12-hash-algo          identityref
|   +--ro TPM12-quote-timestamp
|   |   yang:date-and-time
|   +--ro attester-certificate-name
|   |   tpm:certificate-name-ref
|   +--ro appraisal-timestamp
|   |   yang:date-and-time
|   +--ro verifier-algorithm-type  identityref
|   +--ro verifier-signature       binary
|   +--ro verifier-certificate-keystore-ref
|   |   tpm:certificate-name-ref
+--ro tpm12-quote
+--ro TPM_QUOTE2?  binary

```

Figure 6: YANG Tree for a TPM1.2 Stamped Passport

With either of these passport formats, if the TPM quote is verifiably fresh, then the state of the Attester can be appraised by a network peer.

Note that with [MACSEC] or [IEEE-802.1X], Step 3 plus Step 4 will repeat periodically independently of any subsequent iteration Steps 1 and Step 2. This allows for periodic reauthentication of the link layer in a way not bound to the updating of Verifier A's Attestation Results.

4.2.5. Step 5

Upon receipt of the Stamped Passport generated in Step 4, the Relying Party appraises this Stamped Passport as per its Appraisal Policy for Attestation Results. The result of this application will determine how the Stamped Passport will impact adjacencies within a Trusted Topology. The decision process is as follows:

- (5.1) Verify that (4.2) includes the freshness context from Step 3.
- (5.2) Use a local certificate to validate the signature (4.1).
- (5.3) Verify that the hash from (4.2) matches (4.1)

(5.4) Use the identity of (2.1) to validate the signature of (4.2).

(5.5) 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.

(5.6) Compare the time(EG) TPM state to the time(EG') TPM state

* If TPM2.0

1. If the <TPM2B_DIGEST>, <reset-counter>, <restart-counter> and <safe> are equal between the Attestation Results and the TPM Quote at time(EG') then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to Step 6.
2. If the <reset-counter>, <restart-counter> and <safe> are equal between the Attestation Results and the TPM Quote at time(EG'), and the <clock> object from time(EG') has not incremented by an unacceptable number of seconds since the Attestation Result, then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to Step 6.)
3. Assign the link a null Trustworthiness Vector.

* If TPM1.2

1. If the <pcr-index>'s and <tpm12-pcr-value>'s are equal between the Attestation Results and the TPM Quote at time(EG'), then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to step (6).
2. If the time hasn't incremented an unacceptable number of seconds from the Attestation Results <timestamp> and the system clock of the Relying Party, then Relying Party can accept (2.1) as the link's Trustworthiness Vector. Jump to step 6.)
3. Assign the link a null Trustworthiness Vector.

(5.7) Assemble the Verifier B Trustworthiness Vector

1. Copy Verifier A Trustworthiness Vector to Verifier B Trustworthiness Vector
2. Prune any Trustworthiness Claims the Relying Party doesn't accept from this Verifier.

4.2.6. Step 6

After the Trustworthiness Vector has been validated or reset, based on the link's Trustworthiness Vector, the Relying Party adjusts the link affinity of the corresponding ISIS [I-D.ietf-lsr-flex-algo] topology. ISIS will then replicate the link state across the IGP domain. Traffic will then avoid links which do not have a qualifying Trustworthiness Vector.

5. YANG Module

This YANG module imports modules from [RATS-YANG], [crypto-types] and [RFC6021].

```
<CODE BEGINS> ietf-trustworthiness-claims@2021-11-03.yang
module ietf-trustworthiness-claims {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-trustworthiness-claims";
  prefix tc;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-tcg-algs {
    prefix taa;
    reference
      "draft-ietf-rats-yang-tpm-charra";
  }
  import ietf-tpm-remote-attestation {
    prefix tpm;
    reference
      "draft-ietf-rats-yang-tpm-charra";
  }

  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) 2020 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; see the RFC itself for full legal notices.";

```
revision 2021-11-03 {
  description
    "Initial version.";
  reference
    "draft-voit-rats-trustworthy-path-routing";
}

/*
 * TYPEDEF
 */

typedef trustworthiness-claim {
  type int8;
  description
    "A Verifier asserted value designed to enable a common
    understanding of a Verifier trustworthiness appraisal. The
    value assignments for this 8 bit signed integer will follow
    these guidelines:

    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 63: A standards enumerated reason for the
        warning.
      - Values -33 to -64: A non-standard reason for the warning.

    Contraindicated: The Verifier asserts the Attester is explicitly
    untrustworthy in regard to this aspect.
      - Values 64 to 127: A standards enumerated reason for the
        contraindication.
      - Values -65 to -128: A non-standard reason for the
        contraindication.
```

None: The Verifier makes no assertions about this Trustworthiness Claim. The following values are reserved with the following meanings across all instances of trustworthiness-claim.

- Value 0: 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 unexpected elements which the Verifier is unable to parse. An example might be that the wrong type of Evidence has been delivered.
- Value -1: An unexpected error occurred during the Verifier's appraisal processing. Note: while no claim is being made, the Relying Party MAY make a distinction between a Trustworthiness Claim with enumeration '-1', and no Trustworthiness Claim being provided.";

}

```
typedef hardware {
  type trustworthiness-claim;
  description
    "A Verifier has appraised any Attester hardware and firmware
    which are able to expose fingerprints of their identity and
    running code.
```

The following are specific reserved values of hardware and the meanings of these reserved values:

0: No assertion

1: The Verifier cannot parse unexpected Evidence

-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.";

}

```
typedef instance-identity {
    type trustworthiness-claim;
    description
        "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.)

        The following are specific reserved values of instance-identity
        and the meanings of these reserved values:

        0: No assertion

        1: The Verifier cannot parse unexpected Evidence

        -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."
}

typedef executables {
    type trustworthiness-claim;
    description
        "A Verifier has appraised and evaluated relevant runtime files,
        scripts, and/or other objects which have been loaded into the
        Target environment's memory.

        The following are specific reserved values of executables and
        the meanings of these reserved values:

        0: No assertion

        1: The Verifier cannot parse unexpected Evidence

        -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.";
}

typedef configuration {
  type trustworthiness-claim;
  description
    "A Verifier has appraised an Attester's configuration, and is
    able to make conclusions regarding the exposure of known
    vulnerabilities.

    The following are specific reserved values of configuration and
    the meanings of these reserved values:

    0: No assertion

    1: The Verifier cannot parse unexpected Evidence

    -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

    64:The configuration is unsupportable as it exposes unacceptable
    security vulnerabilities.";
}

/*
 * GROUPINGS
 */

```

```
grouping trustworthiness-vector {
  description
    "Allows the inclusion of a Trustworthiness Vector into
    other constructs.";
  container trustworthiness-vector {
    description
      "One or more Trustworthiness Claims assigned which expose the
      Verifiers evaluation of the Evidence associated with the
      AIK which signed as associated TPM Quote.";
    leaf hardware {
      type hardware;
      description
        "An 8 bit signed integter encoded per the typedef.";
    }
    leaf instance-identity {
      type instance-identity;
      description
        "An 8 bit signed integter encoded per the typedef.";
    }
    leaf executables {
      type executables;
      description
        "An 8 bit signed integter encoded per the typedef.";
    }
    leaf configuration {
      type configuration;
      description
        "An 8 bit signed integter encoded per the typedef.";
    }
  }
}

grouping verifier-evidence {
  description
    "Evidence generated by the Verifier.";
  leaf appraisal-timestamp {
    type yang:date-and-time;
    mandatory true;
    description
      "The timestamp of the Verifier's appraisal. This can be used
      by a Relying Party to determine the freshness of the
      attestation results.";
  }
  leaf verifier-algorithm-type {
    type identityref {
      base taa:asymmetric;
    }
    mandatory true;
  }
}
```

```
    description
      "Platform asymmetric algorithm used in the Verifier signature
       process.";
  }
  leaf verifier-signature {
    type binary;
    mandatory true;
    description
      "Signature of the Verifier across all the current objects in
       the attestation-results container except for 'verifier-
       signature' and 'verifier-certificate-keystore-ref'.
       This assumes CDDL encoding of the objects in the current
       order of this YANG model.";
  }
  leaf verifier-certificate-keystore-ref {
    type tpm:certificate-name-ref;
    mandatory true;
    description
      "A reference to a specific certificate to an asymmetric key
       in the Keystore for the Verifier which can be used to validate
       the 'verifier-signature'. Note that the
       'name' reference must be globally unique so that it can be
       read by the Relying Party in a way which identifies a
       specific Verifier.";
  }
}

grouping tpm20-cddl-attestation-results {
  description
    "Elements combined into a CDDL representation for TPM2.0.";
  uses trustworthiness-vector;
  list tpm20-pcr-selection {
    key "tpm20-hash-algo";
    description
      "Specifies the list of PCRs and Hash Algorithms used by the
       Verifier.";
    reference
      "https://www.trustedcomputinggroup.org/wp-content/uploads/
       TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.9.7";
    uses tpm:tpm20-hash-algo;
    leaf-list pcr-index {
      type tpm:pcr;
      description
        "The numbers of the PCRs associated with the TPM2B_DIGEST.";
    }
  }
  leaf TPM2B_DIGEST {
    mandatory true;
  }
}
```

```
type binary;
description
  "A hash of the latest PCR values (and the hash algorithm used)
  which have been returned from a Verifier for the selected PCRs
  identified within TPML_PCR_SELECTION.";
reference
  "https://www.trustedcomputinggroup.org/wp-content/uploads/
  TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.12.1";
}
leaf clock {
  mandatory true;
  type uint64;
  description
    "Clock is a monotonically increasing counter that advances
    whenever power is applied to a TPM2. The value of Clock is
    incremented each millisecond.";
  reference
    "https://www.trustedcomputinggroup.org/wp-content/uploads/
    TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.11.2";
}
leaf reset-counter {
  mandatory true;
  type uint32;
  description
    "This counter increments on each TPM Reset. The most common
    TPM Reset would be due to a hardware power cycle.";
  reference
    "https://www.trustedcomputinggroup.org/wp-content/uploads/
    TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.11.3";
}
leaf restart-counter {
  mandatory true;
  type uint32;
  description
    "This counter shall increment by one for each TPM Restart or
    TPM Resume. The restartCount shall be reset to zero on a TPM
    Reset.";
  reference
    "https://www.trustedcomputinggroup.org/wp-content/uploads/
    TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.11.4";
}
leaf safe {
  mandatory true;
  type boolean;
  description
    "This parameter is set to YES when the value reported in Clock
    is guaranteed to be unique for the current Owner. It is set to
    NO when the value of Clock may have been reported in a previous
```



```
        attestation or access.";
    reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.11.5";
}
leaf attester-certificate-name {
    mandatory true;
    description
        "The Attester is associated with these results.";
    type tpm:certificate-name-ref;
}
uses verifier-evidence;
}

grouping tpml2-cddl-attestation-results {
    description
        "Elements combined into a CDDL representation for TPM1.2.";
    uses trustworthiness-vector;
    uses tpm:tpml2-pcr-selection;
    leaf-list tpml2-pcr-value {
        type binary;
        description
            "The list of TPM_PCRVALUES from each PCR selected in sequence
            of tpml2-pcr-selection.";
        reference
            "https://www.trustedcomputinggroup.org/wp-content/uploads/
            TPM-Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf
            Section 10.9.7";
    }
    uses tpm:tpml2-hash-algo {
        refine "tpml2-hash-algo" {
            mandatory true;
        }
    }
    leaf TPM12-quote-timestamp {
        type yang:date-and-time;
        mandatory true;
        description
            "The timestamp for when the indicator of freshness (such as a
            nonce) was generated. This is the indicator of freshness
            which was used in the generation of the TPM1.2 quote. This
            timestamp can be used by a Relying Party to determine the
            freshness of the attestation results.";
    }
    leaf attester-certificate-name {
        mandatory true;
        description
            "The Attester is associated with these results.";
```

```
    type tpm:certificate-name-ref;
  }
  uses verifier-evidence;
}

/*
 * NOTIFICATIONS
 */

notification tpm20-stamped-passport {
  description
    "The augmentation of the most recent Attestation Results
    delivered from a Verifier with a TPM2.0 Quote.";
  container attestation-results {
    description
      "The latest Verifier delivered Attestation Results.";
    uses tpm20-cddl-attestation-results;
  }
  container tpm20-quote {
    description
      "The TPM2.0 quote delivered in response to a connectivity
      request.";
    leaf TPMS_QUOTE_INFO {
      type binary;
      mandatory true;
      description
        "A hash of the latest PCR values (and the hash algorithm
        used) which have been returned from a Verifier for the
        selected PCRs and Hash Algorithms.";
      reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.12.1";
    }
    leaf quote-signature {
      type binary;
      mandatory true;
      description
        "Quote signature returned by TPM Quote. The signature was
        generated using the key associated with the
        certificate 'name'.";
      reference
        "https://www.trustedcomputinggroup.org/wp-content/uploads/
        TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 11.2.1";
    }
  }
}

notification tpm12-stamped-passport {
```

```
description
  "The augmentation of the most recent Attestation Results
  delivered from a Verifier with a TPM1.2 Quote.";
container attestation-results {
  description
    "The latest Verifier delivered Attestation Results.";
  uses tpm12-cddl-attestation-results;
}
container tpm12-quote {
  description
    "The TPM1.2 quote delivered in response to a connectivity
    request.";
  leaf TPM_QUOTE2 {
    type binary;
    description
      "Result of a TPM1.2 Quote2 operation. This includes PCRs,
      signatures, locality, the provided nonce and other data which
      can be further parsed to appraise the Attester.";
    reference
      "TPM1.2 commands rev116 July 2007, Section 16.5";
  }
}
}

/*
 * DATA NODES
 */

container attestation-results {
  presence
    "Indicates that Verifier has appraised the security posture of
    the Attester, and returned the results within this container.";
  description
    "Retains the most recent Attestation Results for this Attester.
    It must only be written by a Verifier which is to be trusted by a
    Relying Party.";

  choice tpm-specification-version {
    description
      "Identifies the cryptoprocessor API set which drove the
      Attestation Results.";
    case tpm20-attestation-results-cddl {
      if-feature "taa:tpm20";
      description
        "Attestation Results which are returned from the
        evaluation of Evidence which includes a TPM2 quote.";
      container tpm20-attestation-results-cddl {
        description
```

```
        "Attestation Results which are returned from the
          evaluation of Evidence which includes a TPM2 quote.";
        uses tpm20-cddl-attestation-results;
      }
    }
  case tpm12-attestation-results-cddl {
    if-feature "taa:tpm12";
    description
      "Attestation Results which are returned from the
        evaluation of Evidence which includes a TPM1.2 quote.";
    container tpm12-attestation-results-cddl {
      description
        "Attestation Results which are returned from the
          evaluation of Evidence which includes a TPM1.2 quote.";
      uses tpm12-cddl-attestation-results;
    }
  }
}
}
}
<CODE ENDS>
```

6. Security Considerations

Verifiers are limited to the Evidence available for appraisal from a Router. Although the state of the art is improving, some exploits may not be visible via Evidence.

Only security measurements which are placed into PCRs are capable of being exposed via TPM Quote at time(EG').

Successful attacks on an Verifier have the potential of affecting traffic on the Trusted Topology.

For Trusted Path Routing, links which are part of the FlexAlgo are visible across the entire IGP domain. Therefore a compromised device will know when it is being bypassed.

Access control for the objects in Figure 4 should be tightly controlled so that it becomes difficult for the Stamped Passport to become a denial of service vector.

7. References

7.1. Normative References

- [attestation-results]
"Attestation Results for Connectivity", 2 December 2021,
<<https://datatracker.ietf.org/doc/draft-ietf-rats-ar4si/>>.
- [crypto-types]
"Common YANG Data Types for Cryptography", 17 December
2021, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-crypto-types/>>.
- [RATS-Arch]
"Remote Attestation Procedures Architecture", 8 February
2022, <<https://datatracker.ietf.org/doc/draft-ietf-rats-architecture/>>.
- [RATS-YANG]
"A YANG Data Model for Challenge-Response-based Remote
Attestation Procedures using TPMs", 28 February 2022,
<<https://datatracker.ietf.org/doc/draft-ietf-rats-yang-tpm-charra/>>.
- [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>>.
- [RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types",
RFC 6021, DOI 10.17487/RFC6021, October 2010,
<<https://www.rfc-editor.org/info/rfc6021>>.
- [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>>.
- [TPM1.2] TCG, "TPM 1.2 Main Specification", 2 October 2003,
<<https://trustedcomputinggroup.org/resource/tpm-main-specification/>>.
- [TPM2.0] TCG, "TPM 2.0 Library Specification", 15 March 2013,
<<https://trustedcomputinggroup.org/resource/tpm-library-specification/>>.

7.2. Informative References

- [I-D.ietf-lsr-flex-algo]
Psenak, P., Hegde, S., Filsfils, C., Talaulikar, K., and A. Gulko, "IGP Flexible Algorithm", Work in Progress, Internet-Draft, draft-ietf-lsr-flex-algo-18, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-lsr-flex-algo-18.txt>>.
- [IEEE-802.1X]
Parsons, G., "802.1AE: MAC Security (MACsec)", 1 January 2020, <https://standards.ieee.org/standard/802_1X-2010.html>.
- [MACSEC] Seaman, M., "802.1AE: MAC Security (MACsec)", 1 January 2006, <<https://1.ieee802.org/security/802-1ae/>>.
- [RATS-Device]
"Network Device Remote Integrity Verification", n.d., <<https://datatracker.ietf.org/doc/draft-ietf-rats-tpm-based-network-device-attest>>.
- [RATS-Interactions]
"Reference Interaction Models for Remote Attestation Procedures", 26 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-rats-reference-interaction-models>>.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [stream-subscription]
"Attestation Event Stream Subscription", 16 October 2021, <<https://datatracker.ietf.org/doc/draft-ietf-rats-network-device-subscription/>>.

Appendix A. Acknowledgements

Peter Psenak, Shwetha Bhandari, Adwaith Gautham, Annu Singh, Sujal Sheth, Nancy Cam Winget, and Ned Smith.

Appendix B. Change Log

[THIS SECTION TO BE REMOVED BY THE RFC EDITOR.]

v04-v05

* Tweaks to text

- * Added text which was morphed from that provided by Meiling

v03-v04

- * YANG model updated in concert with draft-voit-rats-attestation-results as Trustworthiness Claim values are added.

v03-v04

- * Moved in concert with draft-voit-rats-attestation-results as Trustworthiness Claims became 8 bit signed integers.

v02-v03

- * Integrated [attestation-results] as prerequisite context.
- * Totally rearranged content. But there were not meaningful process changes.
- * Redid YANG model, and highlighted CDDL needs.

v01-v02

- * Minor tweaks such as renaming and removal of a few trustworthiness-claims

v00-v01

- * Minor tweaks

v02-v00 of draft-voit-rats-trustworthy-path-routing-00

- * file rename was due to an IETF tool submission glitch
- * The Attester's AIK is included within the Stamped Passport. This eliminates the need to provision to AIK certificate on the Relying Party.
- * Removed Centralized variant
- * Added timing diagram, and moved content around to match

v01-v02 of draft-voit-rats-trusted-path-routing

- * Extracted the attestation stream, and placed into draft-birkholz-rats-network-device-subscription
- * Introduced the Trustworthiness Vector

v00-v01 of draft-voit-rats-trusted-path-routing

- * Move all FlexAlgo terminology to allow passport definition to be more generic.
- * Edited Figure 1 so that (4) points to the egress router.
- * Added text freshness mechanisms, and articulated configured subscription support.
- * Minor YANG model clarifications.
- * Added a few open questions which Frank thinks interesting to work.

Appendix C. Open Questions

(1) When there is no available Trusted Topology?

Do we need functional requirements on how to handle traffic to/from Sensitive Subnets when no Trusted Topology exists between IGP edges? The network typically can make this unnecessary. For example it is possible to construct a local IPSec tunnel to make untrusted devices appear as Transparently-Transited Devices. This way Secure Subnets could be tunneled between FlexAlgo nodes where an end-to-end path doesn't currently exist. However there still is a corner case where all IGP egress points are not considered sufficiently trustworthy.

(2) Extension of the Stamped Passport?

Format of the reference to the 'verifier-certificate-name' based on WG desire to include more information in the Stamped Passport. Also we need to make sure that the keystore referenced names are globally unique, else we will need to include a node name in the object set.

(3) Encoding of objects in CDDL. A Verifier will want to sign encoded objects rather than YANG structures. It is most efficient to encode the Attestation Results once on the Verifier, and push these down via a YANG model to the Attester.

Authors' Addresses

Eric Voit
Cisco Systems, Inc.
8135 Maple Lawn Blvd
Fulton, Maryland 20759
United States of America
Email: evoit@cisco.com

Chennakesava Reddy Gaddam
Cisco Systems, Inc.
Cessna Business Park, Kadubeesanahalli
Bangalore 560103
Karnataka
India
Email: chgaddam@cisco.com

Guy C. Fedorkow
Juniper Networks
10 Technology Park Drive
Westford, Massachusetts 01886
United States of America
Email: gfedorkow@juniper.net

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

Meiling Chen
China Mobile
Email: chenmeiling@chinamobile.com