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

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

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

Abstract

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

Discussion Venues

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

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

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

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

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

1.1. Terminology and Requirements Language

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

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

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

1.2. CDDL Typographical Conventions

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

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

Table 1: Type Traits & Typographical Conventions

1.3. Common Types

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

1.3.1. Non-Empty

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

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

1.3.2. Entity

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

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

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

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

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

The following describes each member of the entity-map.

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

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

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

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

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

1.3.3. Validity

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

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

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

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

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

1.3.4. UUID

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

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

1.3.5. UEID

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

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

1.3.6. OID

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

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

1.3.7. Tagged Integer Type

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

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

1.3.8. Hash Entry

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

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

2. CoRIM

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

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

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

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

2.1. CoRIM Map

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

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

The following describes each child item of this map.

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

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

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

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

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

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

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

2.1.1. Identity

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

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

2.1.2. Tags

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

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

2.1.3. Locator Map

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

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

The following describes each child element of this type.

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

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

2.1.4. Profile Types

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

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

2.1.5. Entities

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

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

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

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

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

2.2. Signed CoRIM

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

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

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

The following describes each child element of this type.

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

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

* payload: A CBOR encoded tagged CoRIM.

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

2.2.1. Protected Header Map

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

The following describes each child item of this map.

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

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

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

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

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

2.2.2. Meta Map

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

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

The following describes each child item of this group.

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

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

2.2.2.1. Signer Map

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

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

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

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

3. CoMID

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

3.1. Structure

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

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

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

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

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

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

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

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

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

3.1.1. Tag Identity

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

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

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

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

3.1.1.1. Tag ID

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

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

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

3.1.1.2. Tag Version

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

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

3.1.2. Entities

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

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

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

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

The roles defined for a CoMID entity are:

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

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

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

3.1.3. Linked Tag

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

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

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

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

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

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

The relations defined in this specification are:

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

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

3.1.4. Triples

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

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

The following describes each member of the triples-map:

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

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

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

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

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

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

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

3.1.4.1. Common Types

3.1.4.1.1. Environment

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

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

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

The following describes each member of the environment-map:

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

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

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

3.1.4.1.2. Class

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

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

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

The following describes each member of the class-map:

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

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

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

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

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

3.1.4.1.3. Instance

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

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

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

3.1.4.1.4. Group

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

The type defined for a group identified is UUID.

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

3.1.4.1.5. Measurements

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

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

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

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

The following describes each member of the measurement-map:

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

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

3.1.4.1.5.1. Measurement Keys

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

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

3.1.4.1.5.2. Measurement Values

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.1.4.1.5.3. Version

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

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

The following describes each member of the version-map:

* version (index 0): the version string

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

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

3.1.4.1.5.4. Security Version Number

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

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

3.1.4.1.5.5. Flags

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

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

The following describes each member of the flags-map:

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

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

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

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

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

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

3.1.4.1.5.6. Raw Values Types

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

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

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

3.1.4.1.5.7. Address Types

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

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

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

3.1.4.1.6. Crypto Keys

A cryptographic key can be one of the following formats:

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

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

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

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

tagged-pkix-base64-key-type = #6.554(tstr)
tagged-pkix-base64-cert-type = #6.555(tstr)
tagged-pkix-base64-cert-path-type = #6.556(tstr)
```

3.1.4.1.7. Domain Types

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

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

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

3.1.4.2. Reference Values Triple

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

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

3.1.4.3. Endorsed Values Triple

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

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

3.1.4.4. Device Identity Triple

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

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

3.1.4.5. Attestation Keys Triple

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

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

3.1.4.6. Domain Dependency Triple

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

trustworthiness properties of the subject domain exists.

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

3.1.4.7. Domain Membership Triple

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

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

3.1.4.8. CoMID-CoSWID Linking Triple

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

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

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

3.2. Extensibility

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

4. Implementation Status

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

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

4.1. Veraison

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

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

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

* Implementation's level of maturity: alpha.

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

* Version compatibility: Version -02 of the draft

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

* Implementation experience: n/a

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

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

5. Security and Privacy Considerations

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

6. IANA Considerations

6.1. New COSE Header Parameters

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

6.2. New CBOR Tags

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

6.3. New CoRIM Registries

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

6.4. New CoMID Registries

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

6.5. New Media Types

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

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

Table 2: New Media Types

6.5.1. corim-signed+cbor

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

6.5.2. corim-unsigned+cbor

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

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

6.6. CoAP Content-Formats Registration

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

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

Table 3: New Content-Formats

7. References

7.1. Normative References

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

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

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

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

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

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

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

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

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/rfc/rfc7468>.

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

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

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

[STD66] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

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

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

7.2. Informative References

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

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

Appendix A. Full CoRIM CDDL

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

corim = []
```

Acknowledgments

Carl Wallace for review and comments on this document.

Authors' Addresses

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

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

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

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

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

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

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

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

Abstract

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

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/ietf-rats-wg/draft-ietf-rats-ar4si.

Status of This Memo

This Internet-Draft will expire on 2 September 2024.

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

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

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

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

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

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

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

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

1.1. Requirements Notation

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

1.2. Terminology

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

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

Newly defined terms for this document:

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

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

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

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

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

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

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

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

2. Attestation Results for Secure Interactions

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

2.1. Information driving a Relying Party Action

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

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

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

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

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

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

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

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

The following sections detail requirements for these three categories.

2.2. Non-repudiable Identity

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

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

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

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

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

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

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

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

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

2.2.1. Attester and Attesting Environment

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

This document recognizes three general categories of Attesters.

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

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

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

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

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

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

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

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

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

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

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

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

Table 1

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

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

2.2.2. Verifier

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

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

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

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

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

2.2.3. Communicating Identity

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

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

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

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

2.3. Trustworthiness Claims

2.3.1. Design Principles

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

by a Relying Party.

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

1. Expose a small number of Trustworthiness Claims.

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

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

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

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

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

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

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

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

2.3.2. Enumeration Encoding

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

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

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

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

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

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

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

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

Warning: The Verifier warns about this aspect of trustworthiness.

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

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

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

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

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

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

2.3.3. Assigning a Trustworthiness Claim value

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

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

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

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

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

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

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

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

2.3.4. Specific Claims

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

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: The configuration is a known and approved config.

3: The configuration includes or exposes no known vulnerabilities.

32: The configuration includes or exposes known vulnerabilities.

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

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

99: Cryptographic validation of the Evidence has failed.

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

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

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

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

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

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

99: Cryptographic validation of the Evidence has failed.

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

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

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

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

99: Cryptographic validation of the Evidence has failed.

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

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

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

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

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

99: Cryptographic validation of the Evidence has failed.

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

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

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

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

99: Cryptographic validation of the Evidence has failed.

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

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

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

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

99: Cryptographic validation of the Evidence has failed.

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

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

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

96: Attester source data objects come from contraindicated sources.

99: Cryptographic validation of the Evidence has failed.

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

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

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

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

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

99: Cryptographic validation of the Evidence has failed.

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

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

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

2.3.5. Trustworthiness Vector

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

2.3.6. Trustworthiness Vector for a type of Attesting Environment

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

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

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

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

2.4. Freshness

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

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

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

3. Secure Interactions Models

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

3.1. Background-Check

3.1.1. Verifier Retrieval

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

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

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

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

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

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

3.1.2. Co-resident Verifier

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

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

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

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

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

3.2. Below Zero Trust

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

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

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

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

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

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

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

Figure 1: Below Zero Trust

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

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

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

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

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

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

    1. the verified identity of the Attesting Environment,

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

    3. a freshness proof associated with the Attestation Results,

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

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

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

    1. The Attestation Results from (2)

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

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

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

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

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

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

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

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

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

    7. Assemble the Verifier B Trustworthiness Vector

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

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

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

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

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

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

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

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

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

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

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

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

3.3. Mutual Attestation

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

3.4. Transport Protocol Integration

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

over TLS [RFC8446].

4. Privacy Considerations

Privacy Considerations Text

5. Security Considerations

Security Considerations Text

6. IANA Considerations

See Body.

7. References

7.1. Normative References

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

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

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

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

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

7.2. Informative References

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

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

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

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

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

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

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

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

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

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

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

Appendix A. Implementation Guidance

A.1. Supplementing Trustworthiness Claims

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

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

Appendix B. Supportable Trustworthiness Claims

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

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

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

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

Table 2

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

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

Step 0: set Trustworthiness Vector = Null

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

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

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

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

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

Step 6: Assemble Attestation Results, and push to Attester

End
```

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

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

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

Table 3

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

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

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

Table 4

Appendix C. Some issues being worked

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

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

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

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

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

Appendix D. Contributors

Guy Fedorkow

Email: gfedorkow@juniper.net

Dave Thaler

Email: dthaler@microsoft.com

Ned Smith

Email: ned.smith@intel.com

Lawrence Lundblade

Email: lgl@island-resort.com

Authors' Addresses

Eric Voit
Cisco Systems
Email: evoit@cisco.com

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

Thomas Hardjono
MIT
Email: hardjono@mit.edu

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

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

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

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

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

Abstract

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

Note to Readers

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

For example:

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

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

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

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

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

2. Reference Use Cases

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

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

2.1. Network Endpoint Assessment

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

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

Attester: A device desiring access to a network.

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

2.2. Confidential Machine Learning Model Protection

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

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

Attester: A device desiring to run an ML model.

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

2.3. Confidential Data Protection

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

Attester: An entity desiring to retrieve confidential data.

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

2.4. Critical Infrastructure Control

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

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

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

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

2.5. Trusted Execution Environment Provisioning

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

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

Relying Party: A TAM.

2.6. Hardware Watchdog

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

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

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

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

2.7. FIDO Biometric Authentication

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

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

Attester: FIDO Authenticator.

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

3. Architectural Overview

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

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

Figure 1: Conceptual Data Flow

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

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

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

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

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

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

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

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

3.1. Two Types of Environments of an Attester

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

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

Figure 2: Two Types of Environments

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

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

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

3.2. Layered Attestation Environments

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

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

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

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

Figure 3: Layered Attester

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

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

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

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

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

3.3. Composite Device

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

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

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

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

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

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

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

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

Figure 4: Composite Device

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

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

3.4. Implementation Considerations

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

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

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

4. Terminology

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

This document uses the following terms.

4.1. Roles

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

Produces: Evidence

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

Consumes: Attestation Results, Appraisal Policy for Attestation Results

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

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

Produces: Attestation Results

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

Produces: Appraisal Policy for Attestation Results

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

Produces: Appraisal Policy for Evidence

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

Produces: Endorsements

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

Produces: Reference Values

4.2. Artifacts

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

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

Consumed By: Verifier

Produced By: Endorser

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

Consumed By: Verifier

Produced By: Attester

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

Consumed By: Relying Party

Produced By: Verifier

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

Consumed By: Verifier

Produced By: Verifier Owner

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

Consumed by: Relying Party

Produced by: Relying Party Owner

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

Consumed By: Verifier

Produced By: Reference Value Provider

5. Topological Patterns

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

5.1. Passport Model

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

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

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

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

Three ways in which the process may fail include:

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

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

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

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

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

Figure 5: Passport Model

5.2. Background-Check Model

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

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

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

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

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

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

Figure 6: Background-Check Model

5.3. Combinations

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

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

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

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

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

Figure 7: Example Combination

6. Roles and Entities

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

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

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

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

7. Trust Model

7.1. Relying Party

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

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

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

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

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

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

7.2. Attester

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

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

7.3. Relying Party Owner

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

7.4. Verifier

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

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

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

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

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

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

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

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

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

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

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

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

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

See Section 12 for discussion on security strength.

7.5. Endorser, Reference Value Provider, and Verifier Owner

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

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

8. Conceptual Messages

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

8.1. Evidence

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

8.2. Endorsements

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

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

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

8.3. Reference Values

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

8.4. Attestation Results

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

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

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

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

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

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

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

8.5. Appraisal Policies

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

* comparison for equality against a Reference Value, or

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

* membership in a set of Reference Values, or

* a check against values in other Claims.

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

9. Claims Encoding Formats

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

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

Figure 8: Typical Resource Access

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

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

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

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

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

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

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

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

Figure 9: Multiple Attesters and Relying Parties with Different Formats

10. Freshness

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

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

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

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

10.1. Explicit Timekeeping using Synchronized Clocks

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

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

10.2. Implicit Timekeeping using Nonces

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

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

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

10.3. Implicit Timekeeping using Epoch IDs

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

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

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

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

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

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

10.4. Discussion

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

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

A more detailed discussion with examples appears in Appendix A.

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

11. Privacy Considerations

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

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

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

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

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

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

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

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

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

12. Security Considerations

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

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

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

12.1. Attester and Attestation Key Protection

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

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

12.1.1. On-Device Attester and Key Protection

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

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

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

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

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

12.1.2. Attestation Key Provisioning Processes

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

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

12.1.2.1. Off-Device Key Generation

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

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

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

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

12.1.2.2. On-Device Key Generation

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

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

12.2. Conceptual Message Protection

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

* end-to-end encryption,

* denial of service protection,

* authentication,

* auditing,

* fine grained access controls, and

* logging.

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

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

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

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

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

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

12.3. Epoch ID-based Attestation

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

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

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

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

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

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

Reordering and dropping attacks are mitigated if the transport provides the ability to detect reordering and drop. However, the delay attack described above can't be thwarted in this manner.

12.4. Trust Anchor Protection

As noted in Section 7, Verifiers and Relying Parties have trust anchor stores that must be secured. [RFC6024] contains more discussion of trust anchor store requirements for protecting public keys. Section 6 of [NIST-800-57-p1] contains a comprehensive treatment of the topic, including the protection of symmetric key material. Specifically, a trust anchor store must resist modification against unauthorized insertion, deletion, and modification. Additionally, if the trust anchor is a symmetric key,

the trust anchor store must not allow unauthorized read.

If certificates are used as trust anchors, Verifiers and Relying Parties are also responsible for validating the entire certificate path up to the trust anchor, which includes checking for certificate revocation. For an example of such a proceedure see Section 6 of [RFC5280].

13. IANA Considerations

This document does not require any actions by IANA.

14. Acknowledgments

Special thanks go to Joerg Borchert, Nancy Cam-Winget, Jessica Fitzgerald-McKay, Diego Lopez, Laurence Lundblade, Paul Rowe, Hannes Tschofenig, Frank Xia, and David Wooten.

15. Notable Contributions

Thomas Hardjono created initial versions of the terminology section in collaboration with Ned Smith. Eric Voit provided the conceptual separation between Attestation Provision Flows and Attestation Evidence Flows. Monty Wisemen created the content structure of the first three architecture drafts. Carsten Bormann provided many of the motivational building blocks with respect to the Internet Threat Model.

Peter Loscocco contributed critical review feedback as part of the weekly design team meetings that added precision and depth to several sections.

16. References

16.1. Normative References

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

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

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

16.2. Informative References

[CCC-DeepDive]
Confidential Computing Consortium, "Confidential Computing Deep Dive", n.d., <https://confidentialcomputing.io/whitepaper-02-latest>.

[CTAP] FIDO Alliance, "Client to Authenticator Protocol", n.d., <https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html>.

[I-D.birkholz-rats-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-07, 10 July 2022, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-tuda-07>.

[I-D.birkholz-rats-uccs]
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-birkholz-rats-uccs-03, 8 March 2021, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-uccs-03>.

[I-D.ietf-rats-daa]
Birkholz, H., Newton, C., Chen, L., and D. Thaler, "Direct Anonymous Attestation for the Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-daa-02, 7 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-daa-02>.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. M. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-18, 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-teep-architecture-18>.

[I-D.tschofenig-rats-psa-token]
Tschofenig, H., Frost, S., Brossard, M., Shaw, A. L., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", Work in Progress, Internet-Draft, draft-tschofenig-rats-psa-token-10, 6 September 2022, <https://datatracker.ietf.org/doc/html/draft-tschofenig-rats-psa-token-10>.

[I-D.tschofenig-tls-cwt]
Tschofenig, H. and M. Brossard, "Using CBOR Web Tokens (CWTs) in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-tschofenig-tls-cwt-02, 13 July 2020, <https://datatracker.ietf.org/doc/html/draft-tschofenig-tls-cwt-02>.

[NIST-800-57-p1]
Barker, E., "Recommendation for Key Managemement: Part 1 - General", May 2020, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>.

[OPCUA] OPC Foundation, "OPC Unified Architecture Specification, Part 2: Security Model, Release 1.03", OPC 10000-2 , 25 November 2015, <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/rfc/rfc4949>.

[RFC5209] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <https://www.rfc-editor.org/rfc/rfc5209>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC8322] Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, <https://www.rfc-editor.org/rfc/rfc8322>.

[strengthoffunction] NISC, "Strength of Function", n.d., <https://csrc.nist.gov/glossary/term/strength_of_function>.

[TCG-DICE] Trusted Computing Group, "DICE Certificate Profiles", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Certificate-Profiles-r01_3june2020-1.pdf>.

[TCG-DICE-SIBDA] Trusted Computing Group, "Symmetric Identity Based Device Attestation for DICE", 24 July 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_DICE_SymIDAttest_v1_r0p94_pubrev.pdf>.

[TCGarch] Trusted Computing Group, "Trusted Platform Module Library - Part 1: Architecture", 8 November 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf>.

[WebAuthN] W3C, "Web Authentication: An API for accessing Public Key Credentials", n.d., <https://www.w3.org/TR/webauthn-1/>.

Appendix A. Time Considerations

Section 10 discussed various issues and requirements around freshness of evidence, and summarized three approaches that might be used by different solutions to address them. This appendix provides more details with examples to help illustrate potential approaches, to inform those creating specific solutions.

The table below defines a number of relevant events, with an ID that is used in subsequent diagrams. The times of said events might be defined in terms of an absolute clock time, such as the Coordinated Universal Time timescale, or might be defined relative to some other timestamp or timeticks counter, such as a clock resetting its epoch each time it is powered on.

ID	Event	Explanation of event
VG	Value generated	A value to appear in a Claim was created. In some cases, a value may have technically existed before an Attester became aware of it but the Attester might have no idea how long it has had that value. In such a case, the Value created time is the time at which the Claim containing the copy of the value was created.
NS	Nonce sent	A nonce not predictable to an Attester (recentness & uniqueness) is sent to an Attester.
NR	Nonce relayed	A nonce is relayed to an Attester by another entity.
IR	Epoch ID received	An epoch ID is successfully received and processed by an entity.
EG	Evidence generation	An Attester creates Evidence from collected Claims.
ER	Evidence relayed	A Relying Party relays Evidence to a Verifier.
RG	Result generation	A Verifier appraises Evidence and generates an Attestation Result.
RR	Result relayed	A Relying Party relays an Attestation Result to a Relying Party.
RA	Result appraised	The Relying Party appraises Attestation Results.
OP	Operation performed	The Relying Party performs some operation requested by the Attester via a resource access protocol as depicted in Figure 8, e.g., across a session created earlier at time(RA).
RX	Result expiry	An Attestation Result should no longer be accepted, according to the Verifier that generated it.

Table 1

Using the table above, a number of hypothetical examples of how a solution might be built are illustrated below. This list is not intended to be complete, but is just representative enough to highlight various timing considerations.

All times are relative to the local clocks, indicated by an "_a" (Attester), "_v" (Verifier), or "_r" (Relying Party) suffix.

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

How and if clocks are synchronized depends upon the model.

In the figures below, curly braces indicate containment. For example, the notation Evidence{foo} indicates that 'foo' is contained in the Evidence and is thus covered by its signature.

A.1. Example 1: Timestamp-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party, which depends on using a secure clock synchronization mechanism. As a result, the receiver of a conceptual message containing a timestamp can directly compare it to its own clock and timestamps.

```
      .----------.                      .----------.  .---------------.
      | Attester |                      | Verifier |  | Relying Party |
      '----+-----'                      '-----+----'  '-------+-------'
           |                                  |                |
        time(VG_a)                            |                |
           |                                  |                |
           ~                                  ~                ~
           |                                  |                |
        time(EG_a)                            |                |
           |                                  |                |
           +------Evidence{time(EG_a)}------>|                |
           |                                  |                |
           |                              time(RG_v)           |
           |                                  |                |
           |<-----Attestation Result---------+                |
           |      {time(RG_v),time(RX_v)}     |                |
           ~                                                   ~
           |                                                   |
           +--Attestation Result{time(RG_v),time(RX_v)}--> time(RA_r)
           |                                                   |
           ~                                                   ~
           |                                                   |
           |                                              time(OP_r)
```

The Verifier can check whether the Evidence is fresh when appraising it at time(RG_v) by checking time(RG_v) - time(EG_a) < Threshold, where the Verifier's threshold is large enough to account for the maximum permitted clock skew between the Verifier and the Attester.

If time(VG_a) is also included in the Evidence along with the Claim value generated at that time, and the Verifier decides that it can trust the time(VG_a) value, the Verifier can also determine whether the Claim value is recent by checking time(RG_v) - time(VG_a) < Threshold. The threshold is decided by the Appraisal Policy for Evidence, and again needs to take into account the maximum permitted clock skew between the Verifier and the Attester.

The Attester does not consume the Attestation Result, but might cache it.

The Relying Party can check whether the Attestation Result is fresh when appraising it at time(RA_r) by checking time(RA_r) - time(RG_v) < Threshold, where the Relying Party's threshold is large enough to account for the maximum permitted clock skew between the Relying Party and the Verifier. The result might then be used for some time (e.g., throughout the lifetime of a connection established at time(RA_r)). The Relying Party must be careful, however, to not allow continued use beyond the period for which it deems the

Attestation Result to remain fresh enough. Thus, it might allow use (at time(OP_r)) as long as time(OP_r) - time(RG_v) < Threshold. However, if the Attestation Result contains an expiry time time(RX_v) then it could explicitly check time(OP_r) < time(RX_v).

A.2. Example 2: Nonce-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses nonces instead of timestamps. Compared to the timestamp-based example, it requires an extra round trip to retrieve a nonce, and requires that the Verifier and Relying Party track state to remember the nonce for some period of time.

The advantage is that it does not require that any clocks are synchronized. As a result, the receiver of a conceptual message containing a timestamp cannot directly compare it to its own clock or timestamps. Thus, we use a suffix ("a" for Attester, "v" for Verifier, and "r" for Relying Party) on the IDs below indicating which clock generated them, since times from different clocks cannot be compared. Only the delta between two events from the sender can be used by the receiver.

```
      .----------.                      .----------.  .---------------.
      | Attester |                      | Verifier |  | Relying Party |
      '----+-----'                      '-----+----'  '-------+-------'
           |                                  |                |
        time(VG_a)                            |                |
           |                                  |                |
           ~                                  ~                ~
           |                                  |                |
           |<--Nonce1---------------------time(NS_v)          |
           |                                  |                |
        time(EG_a)                            |                |
           |                                  |                |
           +---Evidence--------------------->|                |
           | {Nonce1, time(EG_a)-time(VG_a)} |                |
           |                                  |                |
           |                              time(RG_v)          |
           |                                  |                |
           |<--Attestation Result------------+                |
           |   {time(RX_v)-time(RG_v)}        |                |
           ~                                                   ~
           |                                                   |
           |<--Nonce2-----------------------------------time(NS_r)
           |                                                   |
        time(RR_a)                                             |
           |                                                   |
           +--[Attestation Result{time(RX_v)-time(RG_v)}, -->|time(RA_r)
           |        Nonce2, time(RR_a)-time(EG_a)]             |
           |                                                   |
           ~                                                   ~
           |                                                   |
           |                                             time(OP_r)
```

In this example solution, the Verifier can check whether the Evidence is fresh at time(RG_v) by verifying that time(RG_v)-time(NS_v) < Threshold.

The Verifier cannot, however, simply rely on a Nonce to determine whether the value of a Claim is recent, since the Claim value might have been generated long before the nonce was sent by the Verifier. However, if the Verifier decides that the Attester can be trusted to correctly provide the delta time(EG_a)-time(VG_a), then it can determine recency by checking time(RG_v)-time(NS_v) + time(EG_a)-time(VG_a) < Threshold.

Similarly if, based on an Attestation Result from a Verifier it trusts, the Relying Party decides that the Attester can be trusted to correctly provide time deltas, then it can determine whether the Attestation Result is fresh by checking time(OP_r)-time(NS_r) +

time(RR_a)-time(EG_a) < Threshold. Although the Nonce2 and time(RR_a)-time(EG_a) values cannot be inside the Attestation Result, they might be signed by the Attester such that the Attestation Result vouches for the Attester's signing capability.

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of time(RX_v)-time(RG_v), then the Relying Party can check time(OP_r)-time(NS_r) < time(RX_v)-time(RG_v).

A.3. Example 3: Epoch ID-based Passport Model Example

The example in Figure 10 illustrates a hypothetical Passport Model solution that uses epoch IDs instead of nonces or timestamps.

The Epoch ID Distributor broadcasts epoch ID I which starts a new epoch E for a protocol participant upon reception at time(IR).

The Attester generates Evidence incorporating epoch ID I and conveys it to the Verifier.

The Verifier appraises that the received epoch ID I is "fresh" according to the definition provided in Section 10.3 whereby retries are required in the case of mismatching epoch IDs, and generates an Attestation Result. The Attestation Result is conveyed to the Attester.

After the transmission of epoch ID I' a new epoch E' is established when I' is received by each protocol participant. The Attester relays the Attestation Result obtained during epoch E (associated with epoch ID I) to the Relying Party using the epoch ID for the current epoch I'. If the Relying Party had not yet received I', then the Attestation Result would be rejected, but in this example, it is received.

In the illustrated scenario, the epoch ID for relaying an Attestation Result to the Relying Party is current, while a previous epoch ID was used to generate Verifier evaluated evidence. This indicates that at least one epoch transition has occurred, and the Attestation Results may only be as fresh as the previous epoch. If the Relying Party remembers the previous epoch ID I during an epoch window as discussed in Section 10.3, and the message is received during that window, the Attestation Result is accepted as fresh, and otherwise it is rejected as stale.

```
                  .-------------.
.----------.      | Epoch ID    |    .----------.  .---------------.
| Attester |      | Distributor |    | Verifier |  | Relying Party |
'----+-----'      '------+------'    '-----+----'  '-------+-------'
     |                   |                 |               |
  time(VG_a)             |                 |               |
     |                   |                 |               |
     ~                   |                 ~               ~
     |                   |                 |               |
  time(IR_a) <-----I--o--I------> time(IR_v) ---> time(IR_r)
     |                   |                 |               |
  time(EG_a)             |                 |               |
     |                   |                 |               |
     +---Evidence--------------------->|               |
     |   {I,time(EG_a)-time(VG_a)}     |               |
     |                   |             |               |
     |                   |           time(RG_v)          |
     |                   |             |               |
     |<--Attestation Result------------+               |
     |   {I,time(RX_v)-time(RG_v)}     |               |
     |                   |             |               |
  time(IR'_a) <----I'-o--I' ----> time(IR'_v) --> time(IR'_r)
     |                   |             |               |
     +---[Attestation Result--------------------> time(RA_r)
     |   {I,time(RX_v)-time(RG_v)},I'] |               |
     |                   |             |               |
     ~                   |             ~               ~
     |                   |             |               |
     |                   |             |          time(OP_r)
```

Figure 10: Epoch ID-based Passport Model

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

The following example illustrates a hypothetical Background-Check Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party. The Attester conveys Evidence to the Relying Party, which treats it as opaque and simply forwards it on to the Verifier.

```
  .----------.         .---------------.                    .----------.
  | Attester |         | Relying Party |                    | Verifier |
  '-------+--'         '-------+-------'                    '----+-----'
          |                    |                                 |
    time(VG_a)                 |                                 |
          |                    |                                 |
          ~                    ~                                 ~
          |                    |                                 |
    time(EG_a)                 |                                 |
          |                    |                                 |
          +----Evidence------->|                                 |
          |   {time(EG_a)}     |                                 |
          |               time(ER_r) ---Evidence{time(EG_a)}---->|
          |                    |                                 |
          |                    |                            time(RG_v)
          |                    |                                 |
          |               time(RA_r) <---Attestation Result------+
          |                    |           {time(RX_v)}          |
          ~                    ~                                 ~
          |                    |                                 |
          |                  time(OP_r)                          |
```

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

A.5. Example 5: Nonce-based Background-Check Model Example

The following example illustrates a hypothetical Background-Check Model solution that uses nonces and thus does not require that any clocks are synchronized. In this example solution, a nonce is generated by a Verifier at the request of a Relying Party, when the Relying Party needs to send one to an Attester.

```
   .----------.         .---------------.                .----------.
   | Attester |         | Relying Party |                | Verifier |
   '----+-----'         '-------+-------'                '----+-----'
        |                       |                              |
     time(VG_a)                 |                              |
        |                       |                              |
        ~                       ~                              ~
        |                       |                              |
        |                       |<-------Nonce-----------time(NS_v)
        |                       |                              |
        |<---Nonce-----------time(NR_r)                        |
        |                       |                              |
     time(EG_a)                 |                              |
        |                       |                              |
        +----Evidence{Nonce}--->|                              |
        |                       |                              |
        |                    time(ER_r) ---Evidence{Nonce}--->|
        |                       |                              |
        |                       |                           time(RG_v)
        |                       |                              |
        |                   ime(RA_r) <---Attestation Result--+
        |                       |    {time(RX_v)-time(RG_v)}   |
        ~                       ~                              ~
        |                       |                              |
        |                    time(OP_r)                        |
```

The Verifier can check whether the Evidence is fresh, and whether a Claim value is recent, the same as in Example 2 above.

However, unlike in Example 2, the Relying Party can use the Nonce to determine whether the Attestation Result is fresh, by verifying that time(OP_r)-time(NR_r) < Threshold.

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of time(RX_v)-time(RG_v), then the Relying Party can check time(OP_r)-time(ER_r) < time(RX_v)-time(RG_v).

Contributors

Monty Wiseman
Email: montywiseman32@gmail.com

Liang Xia
Email: frank.xialiang@huawei.com

Laurence Lundblade
Email: lgl@island-resort.com

Eliot Lear
Email: elear@cisco.com

Jessica Fitzgerald-McKay

Sarah C. Helbe

Andrew Guinn

Peter Loscocco
Email: pete.loscocco@gmail.com

Eric Voit

Thomas Fossati
Email: thomas.fossati@arm.com

Paul Rowe

Carsten Bormann
Email: cabo@tzi.org

Giri Mandyam
Email: mandyam@qti.qualcomm.com

Kathleen Moriarty
Email: kathleen.moriarty.ietf@gmail.com

Guy Fedorkow
Email: gfedorkow@juniper.net

Simon Frost

Email: Simon.Frost@arm.com

Authors' Addresses

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

Dave Thaler
Microsoft
United States of America
Email: dthaler@microsoft.com

Michael Richardson
Sandelman Software Works
Canada
Email: mcr+ietf@sandelman.ca

Ned Smith
Intel Corporation
United States of America
Email: ned.smith@intel.com

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

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 5 September 2024

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

4 March 2024

Direct Anonymous Attestation for the Remote Attestation Procedures Architecture
draft-ietf-rats-daa-05

Abstract

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

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-daa/.

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/. Subscribe at https://www.ietf.org/mailman/listinfo/rats/.

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

Status of This Memo

Copyright Notice

Table of Contents

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, the protocol entity DAA Issuer as described in [DAA] is introduced and its duties as well as its mappings 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, Endorser, Conceptual Message, Evidence, Attestation Result, Attesting Environment.

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

Two protocols as described in [DAA] are illustrated: the Join Protocol and the DAA-Signing Protocol. This section specifies the mapping of the protocol entity DAA Issuer described in [DAA] as an actor in the Join Protocol as well as an actor in the corresponding DAA-Signing Protocol to roles specified in the RATS Architecture.

In the Join Protocol, the protocol entity DAA Issuer takes on the RATS roles of Verifier and associated Relying Party. The mapping is illustrated in Figure Figure 1.

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

Figure 1: RATS Architecture for the Join Protocol

The Join Protocol is essentially an enrollment protocol that consumes Evidence from the Attester (therefore the mapping to the Verifier role). Corresponding Appraisal Policies for Evidence specific to the Join Protocol are used to produce Attestation Results to decide whether to issue a DAA credential to an Attester or not (therefore the mapping to the Relying Party role).

In the DAA-Signing Protocol, the RATS role Endorser is then taken on by the DAA Issuer protocol entity. The mapping is illustrated in Figure Figure 2.

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

Figure 2: RATS Architecture for the DAA-Signing Protocol

The DAA Issuer acts as the Endorser for the Group Public Key that is used by the Verifier for the appraisal of evidence of anonymized Attesters that use the DAA credentials and associated key material to produce Evidence.

In consequence, DAA provides 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 it 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 it now provides Endorsements to the DAA Issuer rather than directly to the Verifier. These Endorsements 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, randomization ensures that the Attester’s identity cannot be revealed to anyone, including the DAA 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 randomized DAA credential.

The following information elements define extensions for corresponding information elements defined in [I-D.ietf-rats-reference-interaction-models], which 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 randomized 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 randomized each time it used.

6. Privacy Considerations

As outlined above, 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 [PBA].

7. Security Considerations

The anonymity property of DAA makes revocation difficult. Well known solutions include:

1. Rogue Attester revocation -- if 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 considerations for DAA over and above those specified 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. References

9.1. Normative References

[DAA] Brickell, E., Camenisch, J., and L. Chen, "Direct anonymous attestation", DOI 10.1145/1030083.1030103, Proceedings of the 11th ACM conference on Computer and communications security, October 2004, <https://doi.org/10.1145/1030083.1030103>.

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

[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://doi.org/10.17487/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://doi.org/10.17487/RFC8174>.

9.2. Informative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-22, 28 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-22>.

[I-D.ietf-rats-reference-interaction-models]
Birkholz, H., Eckel, M., Pan, W., and E. Voit, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-ietf-rats-reference-interaction-models-08, 10 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-reference-interaction-models-08>.

[PBA] Chen, L., Löhr, H., Manulis, M., and A. Sadeghi, "Property-Based Attestation without a Trusted Third Party", DOI 10.1007/978-3-540-85886-7_3, Lecture Notes in Computer Science pp. 31-46, September 2008, <https://doi.org/10.1007/978-3-540-85886-7_3>.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@ietf.contact

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
Internet-Draft
Intended status: Standards Track
Expires: 18 July 2024

L. Lundblade
Security Theory LLC
G. Mandyam

J. O'Donoghue
Qualcomm Technologies Inc.
C. Wallace
Red Hound Software, Inc.
15 January 2024

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-25

Abstract

An Entity Attestation Token (EAT) provides an attested claims set that describes state and characteristics of an entity, a device like a smartphone, IoT device, network equipment or such. This claims set is used by a relying party, server or service to determine the type and degree of trust placed in the entity.

An EAT is either a CBOR Web Token (CWT) or JSON Web Token (JWT) with attestation-oriented claims.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

An Entity Attestation Token (EAT) is a message made up of claims about an entity. An entity may be a device, some hardware or some software. The claims are ultimately used by a relying party who decides if and how it will interact with the entity. The relying party may choose to trust, not trust or partially trust the entity. For example, partial trust may be allowing a monetary transaction only up to a limit.

The security model and goal for attestation are unique and are not the same as for other security standards like those for server authentication, user authentication and secured messaging. To give an example of one aspect of the difference, consider the association and life-cycle of key material. For authentication, keys are associated with a user or service and set up by actions performed by a user or an operator of a service. For attestation, the keys are associated with specific devices and are configured by device manufacturers. The reader is assumed to be familiar with the goals and security model for attestation as described in RATS Architecture [RFC9334] and are not repeated here.

This document defines some common claims that are potentially of broad use. EAT additionally allows proprietary claims and for further claims to be standardized. Here are some examples:

* Make and model of manufactured consumer device

* Make and model of a chip or processor, particularly for a security-oriented chip

* Identification and measurement of the software running on a device

* Configuration and state of a device

* Environmental characteristics of a device like its Global Positioning Sytem (GPS) location

* Formal certifications received

EAT is constructed to support a wide range of use cases.

No single set of claims can accommodate all use cases so EAT is constructed as a framework for defining specific attestation tokens for specific use cases. In particular, EAT provides a profile mechanism to be able to clearly specify the claims needed, the cryptographic algorithms that should be used, and other characteristics for a particular token and use case. Section 6 describes profile contents and provides a profile that is suitable for constrained device use cases.

The entity's EAT implementation generates the claims and typically signs them with an attestation key. It is responsible for protecting the attestation key. Some EAT implementations will use components with very high resistance to attack like Trusted Platform Modules or Secure Elements. Others may rely solely on simple software defenses.

Nesting of tokens and claims sets is accommodated for composite devices that have multiple subsystems.

An EAT may be encoded in either JavaScript Object Notation (JSON) [RFC8259] or Concise Binary Object Representation (CBOR) [RFC8949] as needed for each use case. EAT is built on CBOR Web Token (CWT) [RFC8392] and JSON Web Token (JWT) [RFC7519] and inherits all their characteristics and their security mechanisms. Like CWT and JWT, EAT does not imply any message flow.

Following is a very simple example. It is JSON format for easy reading, but could also be CBOR. Only the Claims-Set, the payload for the JWT, is shown.

```
{
    "eat_nonce": "MIDBNH28iioisjPy",
    "ueid":      "AgAEizrK3Q",
    "oemid":     76543,
    "swname":    "Acme IoT OS",
    "swversion": "3.1.4"
}
```

This example has a nonce for freshness. This nonce is the base64url encoding of a 12 byte random binary byte string. The ueid is effectively a serial number uniquely identifying the device. This ueid is the base64url encoding of a 48-bit MAC address preceded by the type byte 0x02. The oemid identifies the manufacturer using a Private Enterprise Number [PEN]. The software is identified by a simple string name and version. It could be identified by a full manifest, but this is a minimal example.

1.1. Entity Overview

This document uses the term "entity" to refer to the target of an EAT. Most of the claims defined in this document are claims about an entity. An entity is equivalent to a target environment in an attester as defined in [RFC9334].

Layered attestation and composite devices, as described in [RFC9334], are supported by a submodule mechanism (see Section 4.2.18). Submodules allow nesting of EATs and of claims-sets so that such hierarchies can be modeled.

An entity is the same as a "system component", as defined in the Internet Security Glossary [RFC4949].

Note that [RFC4949] defines "entity" and "system entity" as synonyms, and that they may be a person or organization in addition to being a system component. In the EAT context, "entity" never refers to a person or organization. The hardware and software that implement a web site server or service may be an entity in the EAT sense, but the organization that operates, maintains or hosts the web site is not an entity.

Some examples of entities:

* A Secure Element

* A Trusted Execution Environment (TEE)

* A network card in a router

* A router, perhaps with each network card in the router a submodule

* An Internet of Things (IoT) device

* An individual process

* An app on a smartphone

* A smartphone with many submodules for its many subsystems

* A subsystem in a smartphone like the modem or the camera

An entity may have strong security defenses against hardware invasive attacks. It may also have low security, having no special security defenses. There is no minimum security requirement to be an entity.

1.2. EAT as a Framework

EAT is a framework for defining attestation tokens for specific use cases, not a specific token definition. While EAT is based on and compatible with CWT and JWT, it can also be described as:

* An identification and type system for claims in claims-sets

* Definitions of common attestation-oriented claims

* Claims defined in CDDL and serialized using CBOR or JSON

* Security envelopes based on CBOR Object Signing and Encryption (COSE) and Javascript Object Signing and Encryption (JOSE)

* Nesting of claims sets and tokens to represent complex and compound devices

* A profile mechanism for specifying and identifying specific tokens for specific use cases

EAT uses the name/value pairs the same as CWT and JWT to identify individual claims. Section 4 defines common attestation-oriented claims that are added to the CWT and JWT IANA registries. As with CWT and JWT, no claims are mandatory and claims not recognized should be ignored.

Unlike, but compatible with CWT and JWT, EAT defines claims using Concise Data Definition Language (CDDL) [RFC8610]. In most cases the same CDDL definition is used for both the CBOR/CWT serialization and the JSON/JWT serialization.

Like CWT and JWT, EAT uses COSE and JOSE to provide authenticity, integrity and optionally confidentiality. EAT places no new restrictions on cryptographic algorithms, retaining all the cryptographic flexibility of CWT, COSE, JWT and JOSE.

EAT defines a means for nesting tokens and claims sets to accommodate composite devices that have multiple subsystems and multiple attesters. Tokens with security envelopes or bare claims sets may be embedded in an enclosing token. The nested token and the enclosing token do not have to use the same encoding (e.g., a CWT may be enclosed in a JWT).

EAT adds the ability to detach claims sets and send them separately from a security-enveloped EAT that contains a digest of the detached claims set.

This document registers no media or content types for the identification of the type of EAT, its serialization encoding or security envelope. The definition and registration of EAT media types is addressed in [EAT.media-types].

Finally, the notion of an EAT profile is introduced that facilitates the creation of narrowed definitions of EATs for specific use cases in follow-on documents. One basic profile for constrained devices is normatively defined.

1.3. Operating Model and RATS Architecture

EAT follows the operational model described in Figure 1 in RATS Architecture [RFC9334]. To summarize, an attester generates evidence in the form of a claims set describing various characteristics of an entity. Evidence is usually signed by a key that proves the attester and the evidence it produces are authentic. The claims set includes a nonce or some other means to assure freshness.

A verifier confirms an EAT is valid by verifying the signature and may vet some claims using reference values. The verifier then produces attestation results, which may also be represented as an EAT. The attestation results are provided to the relying party, which is the ultimate consumer of the Remote Attestation Procedure. The relying party uses the attestation results as needed for its use case, perhaps allowing an entity to access a network, allowing a financial transaction or such. In some cases, the verifier and relying party are not distinct entities.

1.3.1. Relationship between Evidence and Attestation Results

Any claim defined in this document or in the IANA CWT or JWT registry may be used in evidence or attestation results. The relationship of claims in attestation results to evidence is fundamentally governed by the verifier and the verifier’s policy.

A common use case is for the verifier and its policy to perform checks, calculations and processing with evidence as the input to produce a summary result in attestation results that indicates the overall health and status of the entity. For example, measurements in evidence may be compared to reference values the results of which are represented as a simple pass/fail in attestation results.

It is also possible that some claims in the Evidence will be forwarded unmodified to the relying party in attestation results. This forwarding is subject to the verifier’s implementation and policy. The relying party should be aware of the verifier’s policy to know what checks it has performed on claims it forwards.

The verifier may modify claims it forwards, for example, to implement a privacy preservation functionality. It is also possible the verifier will put claims in the attestation results that give details about the entity that it has computed or looked up in a database. For example, the verifier may be able to put an "oemid" claim in the attestation results by performing a look up based on a "ueid" claim (e.g., serial number) it received in evidence.

This specification does not establish any normative rules for the verifier to follow, as these are a matter of local policy. It is up to each relying party to understand the processing rules of each verifier to know how to interpret claims in attestation results.

2. Terminology

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

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

The examples in Appendix A use CBOR diagnostic notation defined in Section 8 of [RFC8949] and Appendix G of [RFC8610].

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

base64url-encoded: base64url-encoded is as described in [RFC7515], i.e., using URL- and filename-safe character set [RFC4648] with all trailing '=' characters omitted and without the inclusion of any line breaks, whitespace, or other additional characters.

Claim: A piece of information asserted about a subject. A claim is represented as pair with a value and either a name or key to identify it.

Claim Name: A unique text string that identifies the claim. It is used as the claim name for JSON encoding.

Claim Key: The CBOR map key used to identify a claim. (The term "Claim Key" comes from CWT. This document, like COSE, uses the term "label" to refer to CBOR map keys to avoid confusion with cryptographic keys.)

Claim Value: The value portion of the claim. A claim value can be

any CBOR data item or JSON value.

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

This document reuses terminology from RATS Architecure [RFC9334]:

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

Verifier: A role that appraises the validity of evidence about an attester and produces attestation results to be used by a relying party.

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

Evidence: A set of claims generated by an attester to be appraised by a verifier. Evidence may include configuration data, measurements, telemetry, or inferences.

Attestation Results: The output generated by a verifier, typically including information about an attester, where the verifier vouches for the validity of the results

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

Endorsement: A secure statement that an Endorser vouches for the integrity of an attester’s various capabilities such as claims collection and evidence signing.

This document reuses terminology from CDDL [RFC8610]:

Group Socket: refers to the mechanism by which a CDDL definition is extended, as described in [RFC8610] and [RFC9165]

3. Top-Level Token Definition

An "EAT" is an encoded (serialized) message the purpose of which is to transfer a Claims-Set between two parties. An EAT MUST always contain a Claims-Set. In this document an EAT is always a CWT or JWT.

An EAT MUST have authenticity and integrity protection. CWT and JWT provide that in this document.

Further documents may define other encodings and security mechanims for EAT.

The identification of a protocol element as an EAT follows the general conventions used for CWTs and JWTs. Identification depends on the protocol carrying the EAT. In some cases it may be by media type (e.g., in a HTTP Content-Type field). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This document also defines another message, the detached EAT bundle (see Section 5), which holds a collection of detached claims sets and an EAT that provides integrity and authenticity protection for them. Detached EAT bundles can be either CBOR or JSON encoded.

The following CDDL defines the top-level $EAT-CBOR-Tagged-Token, $EAT-CBOR-Untagged-Token and $EAT-JSON-Token-Formats sockets (see Section 3.9 of [RFC8610]), enabling future token formats to be defined. Any new format that plugs into one or more of these sockets MUST be defined by an IETF standards action. Of particular use may be a token type that provides no direct authenticity or integrity protection for use with transports mechanisms that do provide the necessary security services [UCCS].

Nesting of EATs is allowed and defined in Section 4.2.18.3. This includes the nesting of an EAT that is a different format than the enclosing EAT, i.e., the nested EAT may be encoded using CBOR and the enclosing EAT encoded using JSON or vice versa. The definition of Nested-Token references the CDDL defined in this section. When new token formats are defined, the means for identification in a nested token MUST also be defined.

The top-level CDDL type for CBOR-encoded EATs is EAT-CBOR-Token and for JSON is EAT-JSON-Token (while CDDL and CDDL tools provide enough support for shared definitions of most items in this document, they dont provide enough support for this sharing at the top level).

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

$EAT-CBOR-Tagged-Token /= CWT-Tagged-Message
$EAT-CBOR-Tagged-Token /= BUNDLE-Tagged-Message

$EAT-CBOR-Untagged-Token /= CWT-Untagged-Message
$EAT-CBOR-Untagged-Token /= BUNDLE-Untagged-Message

EAT-JSON-Token = $EAT-JSON-Token-Formats

$EAT-JSON-Token-Formats /= JWT-Message
$EAT-JSON-Token-Formats /= BUNDLE-Untagged-Message
```

4. The Claims

This section describes new claims defined for attestation that are to be added to the CWT [IANA.CWT.Claims] and JWT [IANA.JWT.Claims] IANA registries.

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

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

The set of claims that an EAT must contain to be considered valid is context dependent and is outside the scope of this specification. Specific applications of EATs will require implementations to understand and process some claims in particular ways. However, in the absence of such requirements, all claims that are not understood by implementations MUST be ignored.

CDDL, along with a text description, is used to define each claim independent of encoding. Each claim is defined as a CDDL group. In Section 7 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

Each claim defined in this document is added to the $$Claims-Set-Claims group socket. Claims defined by other specifications MUST also be added to the $$Claims-Set-Claims group socket.

All claims in an EAT MUST use the same encoding except where otherwise explicitly stated (e.g., in a CBOR-encoded token, all claims must be CBOR-encoded).

This specification includes a CDDL definition of most of what is defined in [RFC8392]. Similarly, this specification includes CDDL for most of what is defined in [RFC7519]. These definitions are in Appendix D and are not normative.

Each claim described has a unique text string and integer that identifies it. CBOR-encoded tokens MUST use only the integer for claim keys. JSON-encoded tokens MUST use only the text string for claim names.

4.1. eat_nonce (EAT Nonce) Claim

An EAT nonce is either a byte or text string or an array of byte or text strings. The array option supports multistage EAT verification and consumption.

A claim named "nonce" was defined and registered with IANA for JWT, but MUST NOT be used because it does not support multiple nonces. No previous "nonce" claim was defined for CWT. To distinguish from the previously defined JWT "nonce" claim, this claim is named "eat_nonce" in JSON-encoded EATs. The CWT nonce defined here is intended for general purpose use and retains the "Nonce" claim name instead of an EAT-specific name.

An EAT nonce MUST have at least 64 bits of entropy. A maximum EAT nonce size is set to limit the memory required for an implementation. All receivers MUST be able to accommodate the maximum size.

In CBOR, an EAT nonce is a byte string between 8 and 64 bytes in length. In JSON, an EAT nonce is a text string between 8 and 88 bytes in length.

```
$$Claims-Set-Claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

nonce-type = JC< tstr .size (8..88), bstr .size (8..64)>
```

4.2. Claims Describing the Entity

The claims in this section describe the entity itself. They describe the entity whether they occur in evidence or occur in attestation results. See Section 1.3.1 for discussion on how attestation results relate to evidence.

4.2.1. ueid (Universal Entity ID) Claim

The "ueid" claim conveys a UEID, which identifies an individual manufactured entity like a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire entity or a submodule. It does not identify types, models or classes of entities. It is akin to a serial number, though it does not have to be sequential.

UEIDs MUST be universally and globally unique across manufacturers and countries, as described in Section 4.2.1.1. UEIDs MUST also be unique across protocols and systems, as tokens are intended to be embedded in many different protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries should have the same UEID (if they are not global and universal in this way, then relying parties receiving them will have to track other characteristics of the entity to keep entities distinct between manufacturers).

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

There are privacy considerations for UEIDs. See Section 8.1.

A Device Identifier URN is registered for UEIDs. See Section 10.3.

```
$$Claims-Set-Claims //= (ueid-label => ueid-type)

ueid-type = JC<base64-url-text .size (10..44) , bstr .size (7..33)>
```

4.2.1.1. Rules for Creating UEIDs

These rules are solely for the creation of UEIDs. The EAT consumer need not have any awareness of them.

A UEID is constructed of a single type byte followed by the unique bytes for that type. The type byte assures global uniqueness of a UEID even if the unique bytes for different types are accidentally the same.

UEIDS are variable length to accommodate the types defined here and future-defined types.

UEIDs SHOULD NOT be longer than 33 bytes. If they are longer, there is no guarantee that a receiver will be able to accept them. See Appendix B.

A UEID is permanent. It MUST never change for a given entity.

The different types of UEIDs 1) accommodate different manufacturing processes, 2) accommodate small UEIDs, 3) provide an option that doesn't require registration fees and central administration.

In the unlikely event that a new UEID type is needed, it MUST be defined in a standards-track update to this document.

A manufacturer of entities MAY use different types for different products. They MAY also change from one type to another for a given product or use one type for some items of a given produce and another type for other.

Type Byte	Type Name	Specification
0x01	RAND	This is a 128, 192 or 256-bit random number generated once and stored in the entity. This may be constructed by concatenating enough identifiers to make up an equivalent number of random bits and then feeding the concatenation through a cryptographic hash function. It may also be a cryptographic quality random number generated once at the beginning of the life of the entity and stored. It MUST NOT be smaller than 128 bits. See the length analysis in Appendix B.
0x02	IEEE EUI	This makes use of the device identification scheme operated by the IEEE. An EUI is either an EUI-48, EUI-60 or EUI-64 and made up of an OUI, OUI-36 or a CID, different registered company identifiers, and some unique per-entity identifier. EUIs are often the same as or similar to MAC addresses. This type includes MAC-48, an obsolete name for EUI-48. (Note that while entities with multiple network interfaces may have multiple MAC addresses, there is only one UEID for an entity; changeable MAC addresses that don't meet the permanence requirements in this document MUST NOT be used for the UEID or SUEID) [IEEE.802-2001], [OUI.Guide].
0x03	IMEI	This makes use of the International Mobile Equipment Identity (IMEI) scheme operated by the GSMA. This is a 14-digit identifier consisting of an 8-digit Type Allocation Code (TAC) and a 6-digit serial number allocated by the manufacturer, which SHALL be encoded as byte string of length 14 with each byte as the digit's value (not the ASCII encoding of the digit; the digit 3 encodes as 0x03, not 0x33). The IMEI value encoded SHALL NOT include Luhn checksum or SVN information. See [ThreeGPP.IMEI].

Table 1: UEID Composition Types

4.2.1.2. Rules for Consuming UEIDs

For the consumer, a UEID is solely a globally unique opaque identifier. The consumer does not and should not have any awareness of the rules and structure used to achieve global uniqueness.

All implementations MUST be able to receive UEIDs up to 33 bytes long. 33 bytes is the longest defined in this document and gives necessary entropy for probabilistic uniqueness.

The consumer of a UEID MUST treat it as a completely opaque string of bytes and MUST NOT make any use of its internal structure. The reasons for this are:

* UEIDs types vary freely from one manufacturer to the next.

* New types of UEIDs may be defined.

* The manufacturer of an entity is allowed to change from one type of UEID to another anytime they want.

For example, when the consumer receives a type 0x02 UEID, they should not use the OUI part to identify the manufacturer of the device because there is no guarantee all UEIDs will be type 0x02. Different manufacturers may use different types. A manufacturer may make some of their product with one type and others with a different type or even change to a different type for newer versions of their product. Instead, the consumer should use the "oemid" claim.

4.2.2. sueids (Semi-permanent UEIDs) Claim (SUEIDs)

The "sueids" claim conveys one or more semi-permanent UEIDs (SUEIDs). An SUEID has the same format, characteristics and requirements as a UEID, but MAY change to a different value on entity life-cycle events. An entity MAY have both a UEID and SUEIDs, neither, one or the other.

Examples of life-cycle events are change of ownership, factory reset and on-boarding into an IoT device management system. It is beyond the scope of this document to specify particular types of SUEIDs and the life-cycle events that trigger their change. An EAT profile MAY provide this specification.

There MAY be multiple SUEIDs. Each has a text string label the purpose of which is to distinguish it from others. The label MAY name the purpose, application or type of the SUEID. For example, the label for the SUEID used by XYZ Onboarding Protocol could thus be "XYZ". It is beyond the scope of this document to specify any SUEID labeling schemes. They are use case specific and MAY be specified in an EAT profile.

If there is only one SUEID, the claim remains a map and there still MUST be a label.

An SUEID provides functionality similar to an IEEE LDevID [IEEE.802.1AR].

There are privacy considerations for SUEIDs. See Section 8.1.

A Device Identifier URN is registered for SUEIDs. See Section 10.3.

```
$$Claims-Set-Claims //= (sueids-label => sueids-type)

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

4.2.3. oemid (Hardware OEM Identification) Claim

The "oemid" claim identifies the Original Equipment Manufacturer (OEM) of the hardware. Any of the three forms described below MAY be used at the convenience of the claim sender. The receiver of this claim MUST be able to handle all three forms.

Note that the "hwmodel" claim in Section 4.2.4, the "oemboot" claim in Section 4.2.8 and "dbgstat" claim in Section 4.2.9 depend on this claim.

Sometimes one manufacturer will acquire or merge with another. Depending on the situation and use case newly manfactured devices may continue to use the old OEM ID or switch to a new one. This is left to the discretion of the manufacturers, but they should consider how it affects the above-mentioned claims and the attestation eco-system for their devices. The considerations are the same for all three forms of this claim.

4.2.3.1. Random Number Based OEM ID

The random number based OEM ID MUST always be 16 bytes (128 bits) long.

The OEM may create their own ID by using a cryptographic-quality random number generator. They would perform this only once in the life of the company to generate the single ID for said company. They would use that same ID in every entity they make. This uniquely identifies the OEM on a statistical basis and is large enough should there be ten billion companies.

In JSON-encoded tokens this MUST be base64url-encoded.

4.2.3.2. IEEE Based OEM ID

The IEEE operates a global registry for MAC addresses and company IDs. This claim uses that database to identify OEMs. The contents of the claim may be either an IEEE MA-L, MA-M, MA-S or an IEEE CID [IEEE-RA]. An MA-L, formerly known as an OUI, is a 24-bit value used as the first half of a MAC address. MA-M similarly is a 28-bit value uses as the first part of a MAC address, and MA-S, formerly known as OUI-36, a 36-bit value. Many companies already have purchased one of these. A CID is also a 24-bit value from the same space as an MA-L, but not for use as a MAC address. IEEE has published Guidelines for Use of EUI, OUI, and CID [OUI.Guide] and provides a lookup service [OUI.Lookup].

Companies that have more than one of these IDs or MAC address blocks SHOULD select one and prefer that for all their entities.

Commonly, these are expressed in Hexadecimal Representation as described in [IEEE.802-2001]. It is also called the Canonical format. When this claim is encoded the order of bytes in the bstr are the same as the order in the Hexadecimal Representation. For example, an MA-L like "AC-DE-48" would be encoded in 3 bytes with values 0xAC, 0xDE, 0x48.

This format is always 3 bytes in size in CBOR.

In JSON-encoded tokens, this MUST be base64url-encoded and always 4 bytes.

4.2.3.3. IANA Private Enterprise Number Based OEM ID

IANA maintains a registry for Private Enterprise Numbers (PEN) [PEN]. A PEN is an integer that identifies an enterprise and may be used to construct an object identifier (OID) relative to the following OID arc that is managed by IANA: iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1).

For EAT purposes, only the integer value assigned by IANA as the PEN is relevant, not the full OID value.

In CBOR this value MUST be encoded as a major type 0 integer and is typically 3 bytes. In JSON, this value MUST be encoded as a number.

```
$$Claims-Set-Claims //= (
    oemid-label => oemid-pen / oemid-ieee / oemid-random
)

oemid-pen = int

oemid-ieee = JC<oemid-ieee-json, oemid-ieee-cbor>
oemid-ieee-cbor = bstr .size 3
oemid-ieee-json = base64-url-text .size 4

oemid-random = JC<oemid-random-json, oemid-random-cbor>
oemid-random-cbor = bstr .size 16
oemid-random-json = base64-url-text .size 24
```

4.2.4. hwmodel (Hardware Model) Claim

The "hwmodel" claim differentiates hardware models, products and variants manufactured by a particular OEM, the one identified by OEM ID in Section 4.2.3. It MUST be unique within a given OEM ID. The concatenation of the OEM ID and "hwmodel" give a global identifier of a particular product. The "hwmodel" claim MUST only be present if an "oemid" claim described in Section 4.2.3 is present.

The granularity of the model identification is for each OEM to decide. It may be very granular, perhaps including some version information. It may be very general, perhaps only indicating top-level products.

The "hwmodel" claim is for use in protocols and not for human consumption. The format and encoding of this claim should not be human-readable to discourage use other than in protocols. If this claim is to be derived from an already-in-use human-readable identifier, it can be run through a hash function.

There is no minimum length so that an OEM with a very small number of models can use a one-byte encoding. The maximum length is 32 bytes. All receivers of this claim MUST be able to receive this maximum size.

The receiver of this claim MUST treat it as a completely opaque string of bytes, even if there is some apparent naming or structure. The OEM is free to alter the internal structure of these bytes as long as the claim continues to uniquely identify its models.

```
$$Claims-Set-Claims //= (
    hardware-model-label => hardware-model-type
)

hardware-model-type = JC<base64-url-text .size (4..44),
                         bytes .size (1..32)>
```

4.2.5. hwversion (Hardware Version) Claim

The "hwversion" claim is a text string the format of which is set by each manufacturer. The structure and sorting order of this text string can be specified using the version-scheme item from CoSWID [RFC9393]. It is useful to know how to sort versions so the newer can be distinguished from the older. A "hwversion" claim MUST only be present if a "hwmodel" claim described in Section 4.2.4 is present.

```
$$Claims-Set-Claims //=  (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version:  tstr,
    ? scheme:  $version-scheme
]
```

4.2.6. swname (Software Name) Claim

The "swname" claim contains a very simple free-form text value for naming the software used by the entity. Intentionally, no general rules or structure are set. This will make it unsuitable for use cases that wish precise naming.

If precise and rigourous naming of the software for the entity is needed, the "manifests" claim described in Section 4.2.15 may be used instead.

```
$$Claims-Set-Claims //= ( sw-name-label => tstr )
```

4.2.7. swversion (Software Version) Claim

The "swversion" claim makes use of the CoSWID version-scheme defined in [RFC9393] to give a simple version for the software. A "swversion" claim MUST only be present if a "swname" claim described in Section 4.2.6 is present.

The "manifests" claim Section 4.2.15 may be instead if this is too simple.

```
$$Claims-Set-Claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version:  tstr
    ? scheme:  $version-scheme
]
```

4.2.8. oemboot (OEM Authorized Boot) Claim

An "oemboot" claim with value of true indicates the entity booted with software authorized by the manufacturer of the entity as indicated by the "oemid" claim described in Section 4.2.3. It indicates the firmware and operating system are fully under control of the OEM and may not be replaced by the end user or even the enterprise that owns the device. The means of control may be by cryptographic authentication of the software, by the software being in Read-Only Memory (ROM), a combination of the two or other. If this claim is present the "oemid" claim MUST be present.

```
$$Claims-Set-Claims //= (oem-boot-label => bool)
```

4.2.9. dbgstat (Debug Status) Claim

The "dbgstat" claim applies to entity-wide or submodule-wide debug facilities of the entity like [JTAG] and diagnostic hardware built into chips. It applies to any software debug facilities related to privileged software that allows system-wide memory inspection, tracing or modification of non-system software like user mode applications.

This characterization assumes that debug facilities can be enabled and disabled in a dynamic way or be disabled in some permanent way, such that no enabling is possible. An example of dynamic enabling is one where some authentication is required to enable debugging. An example of permanent disabling is blowing a hardware fuse in a chip. The specific type of the mechanism is not taken into account. For example, it does not matter if authentication is by a global password or by per-entity public keys.

As with all claims, the absence of the "dbgstat" claim means it is not reported.

This claim is not extensible so as to provide a common interoperable description of debug status. If a particular implementation considers this claim to be inadequate, it can define its own proprietary claim. It may consider including both this claim as a coarse indication of debug status and its own proprietary claim as a refined indication.

The higher levels of debug disabling requires that all debug disabling of the levels below it be in effect. Since the lowest level requires that all of the target's debug be currently disabled, all other levels require that too.

There is no inheritance of claims from a submodule to a superior module or vice versa. There is no assumption, requirement or guarantee that the target of a superior module encompasses the targets of submodules. Thus, every submodule must explicitly describe its own debug state. The receiver of an EAT MUST NOT assume that debug is turned off in a submodule because there is a claim indicating it is turned off in a superior module.

An entity may have multiple debug facilities. The use of plural in the description of the states refers to that, not to any aggregation or inheritance.

The architecture of some chips or devices may be such that a debug facility operates for the whole chip or device. If the EAT for such a chip includes submodules, then each submodule should independently report the status of the whole-chip or whole-device debug facility. This is the only way the receiver can know the debug status of the submodules since there is no inheritance.

4.2.9.1. Enabled

If any debug facility, even manufacturer hardware diagnostics, is currently enabled, then this level must be indicated.

4.2.9.2. Disabled

This level indicates all debug facilities are currently disabled. It may be possible to enable them in the future. It may also be that they were enabled in the past, but they are currently disabled.

4.2.9.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the entity booted/started.

4.2.9.4. Disabled Permanently

This level indicates all non-manufacturer facilities are permanently disabled such that no end user or developer can enable them. Only the manufacturer indicated in the "oemid" claim can enable them. This also indicates that all debug facilities are currently disabled and have been so since boot/start. If this debug state is reported, the "oemid" claim MUST be present.

4.2.9.5. Disabled Fully and Permanently

This level indicates that all debug facilities for the entity are permanently disabled.

```
$$Claims-Set-Claims //= ( debug-status-label => debug-status-type )

debug-status-type = ds-enabled /
                    disabled /
                    disabled-since-boot /
                    disabled-permanently /
                    disabled-fully-and-permanently

ds-enabled                      = JC< "enabled", 0 >
disabled                        = JC< "disabled", 1 >
disabled-since-boot             = JC< "disabled-since-boot", 2 >
disabled-permanently            = JC< "disabled-permanently", 3 >
disabled-fully-and-permanently =
                       JC< "disabled-fully-and-permanently", 4 >
```

4.2.10. location (Location) Claim

The "location" claim gives the geographic position of the entity from which the attestation originates. Latitude, longitude, altitude, accuracy, altitude-accuracy, heading and speed MUST be as defined in the W3C Geolocation API [W3C.GeoLoc] (which, in turn, is based on [WGS84]). If the entity is stationary, the heading is NaN (floating-point not-a-number). Latitude and longitude MUST always be provided. If any other of these values are unknown, they are omitted.

The location may have been cached for a period of time before token creation. For example, it might have been minutes or hours or more since the last contact with a GNSS satellite. Either the timestamp or age data item can be used to quantify the cached period. The timestamp data item is preferred as it a non-relative time. If the entity has no clock or the clock is unset but has a means to measure the time interval between the acquisition of the location and the token creation the age may be reported instead. The age is in seconds.

See location-related privacy considerations in Section 8.2.

```
$$Claims-Set-Claims //= (location-label => location-type)

location-type = {
    latitude => number,
    longitude => number,
    ? altitude => number,
    ? accuracy => number,
    ? altitude-accuracy => number,
    ? heading => number,
    ? speed => number,
    ? timestamp => ~time-int,
    ? age => uint
}

latitude          = JC< "latitude",          1 >
longitude         = JC< "longitude",         2 >
altitude          = JC< "altitude",          3 >
accuracy          = JC< "accuracy",          4 >
altitude-accuracy = JC< "altitude-accuracy", 5 >
heading           = JC< "heading",           6 >
speed             = JC< "speed",             7 >
timestamp         = JC< "timestamp",         8 >
age               = JC< "age",               9 >
```

4.2.11. uptime (Uptime) Claim

The "uptime" claim contains the number of seconds that have elapsed since the entity or submodule was last booted.

```
$$Claims-Set-Claims //= (uptime-label => uint)
```

4.2.12. bootcount (Boot Count) Claim

The "bootcount" claim contains a count of the number times the entity or submodule has been booted. Support for this claim requires a persistent storage on the device.

```
$$Claims-Set-Claims //= (boot-count-label => uint)
```

4.2.13. bootseed (Boot Seed) Claim

The "bootseed" claim contains a value created at system boot time that allows differentiation of attestation reports from different boot sessions of a particular entity (e.g., a certain UEID).

This value is usually public. It is not a secret and MUST NOT be used for any purpose that a secret seed is needed, such as seeding a random number generator.

There are privacy considerations for this claim. See Section 8.3.

```
$$Claims-Set-Claims //=  (boot-seed-label => binary-data)
```

4.2.14. dloas (Digital Letters of Approval) Claim

The "dloas" claim conveys one or more Digital Letters of Approval (DLOAs). A DLOA [DLOA] is a document that describes a certification that an entity has received. Examples of certifications represented by a DLOA include those issued by Global Platform and those based on Common Criteria. The DLOA is unspecific to any particular certification type or those issued by any particular organization.

This claim is typically issued by a verifier, not an attester. Verifiers MUST NOT issue this claim unless the entity has received the certification indicated by the DLOA.

This claim MAY contain more than one DLOA. If multiple DLOAs are present, verifiers MUST NOT issue this claim unless the entity has received all of the certifications.

DLOA documents are always fetched from a registrar that stores them. This claim contains several data items used to construct a Uniform Resource Locator (URL) for fetching the DLOA from the particular registrar.

This claim MUST be encoded as an array with either two or three elements. The first element MUST be the URL for the registrar. The second element MUST be a platform label indicating which platform was certified. If the DLOA applies to an application, then the third element is added which MUST be an application label. The method of constructing the registrar URL, platform label and possibly application label is specified in [DLOA].

The retriever of a DLOA MUST follow the recommendation in [DLOA] and use TLS or some other means to be sure the DLOA registrar they are accessing is authentic. The platform and application labels in the claim indicate the correct DLOA for the entity.

```
$$Claims-Set-Claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: general-uri
    dloa_platform_label: text
    ? dloa_application_label: text
]
```

4.2.15. manifests (Software Manifests) Claim

The "manifests" claim contains descriptions of software present on the entity. These manifests are installed on the entity when the software is installed or are created as part of the installation process. Installation is anything that adds software to the entity, possibly factory installation, the user installing elective applications and so on. The defining characteristic of a manifest is that it is created by the software manufacturer. The purpose of this claim is to relay unmodified manifests to the verifier and possibly to the relying party.

Some manifests are signed by their software manufacturer independently, and some are not either because they do not support signing or the manufacturer chose not to sign them. For example, a CoSWID might be signed independently before it is included in an EAT. When signed manifests are put into an EAT, the manufacturer's signature SHOULD be included even though an EAT's signature will also cover the manifest. This allows the receiver to directly verify the manufacturer-originated manifest.

This claim allows multiple manifest formats. For example, the manifest may be a CBOR-encoded CoSWID, an XML-encoded SWID or other. Identification of the type of manifest is always by a Constrained Application Protocol (CoAP) Content-Format integer [RFC7252]. If there is no CoAP identifier registered for a manifest format, one MUST be registered.

This claim MUST be an array of one or more manifests. Each manifest in the claim MUST be an array of two. The first item in the array of two MUST be an integer CoAP Content-Format identifier. The second item is MUST be the actual manifest.

In JSON-encoded tokens the manifest, whatever encoding it is, MUST be placed in a text string. When a non-text encoded manifest like a CBOR-encoded CoSWID is put in a JSON-encoded token, the manifest MUST be base-64 encoded.

This claim allows for multiple manifests in one token since multiple software packages are likely to be present. The multiple manifests MAY be of different encodings. In some cases EAT submodules may be used instead of the array structure in this claim for multiple manifests.

A CoSWID manifest MUST be a payload CoSWID, not an evidence CoSWID. These are defined in [RFC9393].

A Software Updates for Internet of Things (SUIT) Manifest [SUIT.Manifest] may be used.

This claim is extensible for use of manifest formats beyond those mentioned in this document. No particular manifest format is preferred. For manifest interoperability, an EAT profile as defined in Section 6, should be used to specify which manifest format(s) are allowed.

```
$$Claims-Set-Claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ manifest-format]

manifest-format = [
    content-type:   coap-content-format,
    content-format: JC< $manifest-body-json,
                        $manifest-body-cbor >
]

$manifest-body-cbor /= bytes .cbor untagged-coswid
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= bytes .cbor SUIT_Envelope
$manifest-body-json /= base64-url-text
```

4.2.16. measurements (Measurements) Claim

The "measurements" claim contains descriptions, lists, evidence or measurements of the software that exists on the entity or any other measurable subsystem of the entity (e.g. hash of sections of a file system or non-volatile memory). The defining characteristic of this claim is that its contents are created by processes on the entity that inventory, measure or otherwise characterize the software on the entity. The contents of this claim do not originate from the manufacturer of the measurable subsystem (e.g. developer of a software library).

This claim can be a [RFC9393]. When the CoSWID format is used, it MUST be an evidence CoSWID, not a payload CoSWID.

Formats other than CoSWID MAY be used. The identification of format is by CoAP Content Format, the same as the "manifests" claim in Section 4.2.15.

```
$$Claims-Set-Claims //= (
    measurements-label => measurements-type
)

measurements-type = [+ measurements-format]

measurements-format = [
    content-type:   coap-content-format,
    content-format: JC< $measurements-body-json,
                        $measurements-body-cbor >
]

$measurements-body-cbor /= bytes .cbor untagged-coswid
$measurements-body-json /= base64-url-text
```

4.2.17. measres (Software Measurement Results) Claim

The "measres" claim is a general-purpose structure for reporting comparison of measurements to expected reference values. This claim provides a simple standard way to report the result of a comparison as success, failure, fail to run, and absence.

It is the nature of measurement systems that they are specific to the operating system, software and hardware of the entity that is being measured. It is not possible to standardize what is measured and how it is measured across platforms, OS's, software and hardware. The recipient must obtain the information about what was measured and what it indicates for the characterization of the security of the entity from the provider of the measurement system. What this claim provides is a standard way to report basic success or failure of the measurement. In some use cases it is valuable to know if measurements succeeded or failed in a general way even if the details of what was measured is not characterized.

This claim MAY be generated by the verifier and sent to the relying party. For example, it could be the results of the verifier comparing the contents of the "measurements" claim, Section 4.2.16, to reference values.

This claim MAY also be generated on the entity if the entity has the ability for one subsystem to measure and evaluate another subsystem. For example, a TEE might have the ability to measure the software of the rich OS and may have the reference values for the rich OS.

Within an entity, attestation target or submodule, multiple results can be reported. For example, it may be desirable to report the results for measurements of the file system, chip configuration, installed software, running software and so on.

Note that this claim is not for reporting the overall result of a verifier. It is solely for reporting the result of comparison to reference values.

An individual measurement result (individual-result) is an array consisting of two elements, an identifier of the measurement (result-id) and an enumerated type of the result (result). Different measurement systems will measure different things and perhaps measure the same thing in different ways. It is up to each measurement system to define identifiers (result-id) for the measurements it reports.

Each individual measurement result is part of a group that may contain many individual results. Each group has a text string that names it, typically the name of the measurement scheme or system.

The claim itself consists of one or more groups.

The values for the results enumerated type are as follows:

1 -- comparison successful: Indicates successful comparison to reference values.

2 -- comparison fail: The comparison was completed and did not compare correctly to the reference values.

3 -- comparison not run: The comparison was not run. This includes error conditions such as running out of memory.

4 -- measurement absent: The particular measurement was not available for comparison.

```
$$Claims-Set-Claims //= (
    measurement-results-label =>
        [ + measurement-results-group ] )

measurement-results-group = [
    measurement-system: tstr,
    measurement-results: [ + individual-result ]
]

individual-result = [
    result-id:  tstr / binary-data,
    result:     result-type,
]

result-type = comparison-successful /
              comparison-fail /
              comparison-not-run /
              measurement-absent

comparison-successful    = JC< "success",       1 >
comparison-fail          = JC< "fail",          2 >
comparison-not-run       = JC< "not-run",       3 >
measurement-absent       = JC< "absent",        4 >
```

4.2.18. submods (Submodules)

Some devices are complex and have many subsystems. A mobile phone is a good example. It may have subsystems for communications (e.g., Wi-Fi and cellular), low-power audio and video playback, multiple security-oriented subsystems like a TEE and a Secure Element, and etc. The claims for a subsystem can be grouped together in a submodule.

Submodules may be used in either evidence or attestation results.

Because system architecture will vary greatly from use case to use case, there are no set requirements for what a submodule represents either in evidence or in attestation results. Profiles, Section 6, may wish to impose requirements. An attester that outputs evidence with submodules should document the semantics it associates with particular submodules for the verifier. Likewise, a verifier that outputs attestation results with submodules should document the semantics it associates with the submodules for the relying party.

A submodule claim is a map that holds some number of submodules. Each submodule is named by its label in the submodule claim map. The value of each entry in a submodule may be a Claims-Set, nested token or Detached-Submodule-Digest. This allows for the submodule to serve as its own attester or not and allows for claims for each submodule to be represented directly or indirectly, i.e., detached.

A submodule may include a submodule, allowing for arbitrary levels of nesting. However, submodules do not inherit anything from the containing token and must explicitly include all claims. Submodules may contain claims that are present in any surrounding token or submodule. For example, the top-level of the token may have a UEID, a submodule may have a different UEID and a further subordinate submodule may also have a UEID.

The following sub-sections define the three types for representing submodules:

* A submodule Claims-Set

* The digest of a detached Claims-Set

* A nested token, which can be any EAT

The Submodule type definition and Nested-Token type definition vary with the type of encoding. The definitions for CBOR-encoded EATs are as follows:

```
Nested-Token = CBOR-Nested-Token

CBOR-Nested-Token =
    JSON-Token-Inside-CBOR-Token /
    CBOR-Token-Inside-CBOR-Token

CBOR-Token-Inside-CBOR-Token = bstr .cbor $EAT-CBOR-Tagged-Token

JSON-Token-Inside-CBOR-Token = tstr

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / CBOR-Nested-Token /
            Detached-Submodule-Digest
```

The Submodule and Nested-Token definitions for JSON-encoded EATs is as below. This difference in definitions vs. CBOR is necessary because JSON has no tag mechanism and no byte string type to help indicate the nested token is CBOR.

```
Nested-Token = JSON-Selector

JSON-Selector = $JSON-Selector

$JSON-Selector /= [type: "JWT", nested-token: JWT-Message]
$JSON-Selector /= [type: "CBOR", nested-token: CBOR-Token-Inside-JSON-Token]
$JSON-Selector /= [type: "BUNDLE", nested-token: Detached-EAT-Bundle]
$JSON-Selector /= [type: "DIGEST", nested-token: Detached-Submodule-Digest]

CBOR-Token-Inside-JSON-Token = base64-url-text

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / JSON-Selector
```

The Detached-Submodule-Digest type is defined as follows:

```
Detached-Submodule-Digest = [
   hash-algorithm : text / int,
   digest         : binary-data
]
```

Nested tokens can be one of three types as defined in this document or types standardized in follow-on documents (e.g., [UCCS]). Nested tokens are the only mechanism by which JSON can be embedded in CBOR and vice versa.

The addition of further types is accomplished by augmenting the $EAT-CBOR-Tagged-Token socket or the $JSON-Selector socket.

When decoding a JSON-encoded EAT, the type of submodule is determined as follows. A JSON object indicates the submodule is a Claims-Set. In all other cases, it is a JSON-Selector, which is an array of two elements that indicates whether the submodule is a nested token or a Detached-Submodule-Digest.The first element in the array indicates the type present in the second element. If the value is JWT, CBOR, BUNDLE or a future-standardized token types, e.g., [UCCS], the submodule is a nested token of the indicated type, i.e., JWT-Message, CBOR-Token-Inside-JSON-Token, Detached-EAT-Bundle, or a future type. If the value is "DIGEST", the submodule is a Detached-Submodule-Digest. Any other value indicates a standardized extension to this specification.

When decoding a CBOR-encoded EAT, the CBOR item type indicates the type of the submodule as follows. A map indicates a CBOR-encoded submodule Claims-Set. An array indicates a CBOR-encoded Detached-Submodule-Digest. A byte string indicates a CBOR-encoded CBOR-Nested-Token. A text string indicates a JSON-encoded JSON-Selector. Where JSON-Selector is used in a CBOR-encoded EAT, the "DIGEST" type and corresponding Detached-Submodule-Digest type MUST NOT be used.

The type of a CBOR-encoded nested token is always determined by the CBOR tag encountered after the byte string wrapping is removed in a CBOR-encoded enclosing token or after the base64 wrapping is removed in JSON-encoded enclosing token.

The type of a JSON-encoded nested token is always determined by the string name in JSON-Selector and is always JWT, BUNDLE or a new name standardized outside this document for a further type (e.g., UCCS). This string name may also be CBOR to indicate the nested token is CBOR-encoded.

"JWT": The second array item MUST be a JWT formatted according to [RFC7519]

"CBOR": The second array item MUST be some base64url-encoded CBOR that is a tag, typically a CWT or CBOR-encoded detached EAT bundle

"BUNDLE": The second array item MUST be a JSON-encoded Detached EAT Bundle as defined in this document.

"DIGEST": The second array item MUST be a JSON-encoded Detached-Submodule-Digest as defined in this document.

As noted elsewhere, additional EAT types may be defined by a standards action. New type specifications MUST address the integration of the new type into the Submodule claim type for submodules.

4.2.18.1. Submodule Claims-Set

The Claims-Set type provides a means of representing claims from a submodule that does not have its own attesting environment, i.e., it has no keys distinct from the attester producing the surrounding token. Claims are represented as a Claims-Set. Submodule claims represented in this way are secured by the same mechanism as the enclosing token (e.g., it is signed by the same attestation key).

The encoding of a submodule Claims-Set MUST be the same as the encoding as the surrounding EAT, e.g., all submodule Claims-Sets in a CBOR-encoded token must be CBOR-encoded.

4.2.18.2. Detached Submodule Digest

The Detached-Submodule-Digest type is similar to a submodule Claims-Set, except a digest of the Claims-Set is included in the claim with the Claims-Set contents conveyed separately. The separately-conveyed Claims-Set is called a detached claims set. The input to the digest algorithm is directly the CBOR or JSON-encoded Claims-Set for the submodule. There is no byte-string wrapping or base 64 encoding.

The data type for this type of submodule is an array consisting of two data items: an algorithm identifier and a byte string containing the digest. The hash algorithm identifier is always from the COSE Algorithm registry, [IANA.COSE.Algorithms]. Either the integer or string identifier may be used. The hash algorithm identifier is never from the JOSE Algorithm registry.

A detached EAT bundle, described in Section 5, may be used to convey detached claims sets and the EAT containing the corresponding detached digests. EAT, however, doesn’t require use of a detached EAT bundle. Any other protocols may be used to convey detached claims sets and the EAT containing the corresponding detached digests. Detached Claims-Sets must not be modified in transit, else validation will fail.

4.2.18.3. Nested Tokens

The CBOR-Nested-Token and JSON-Selector types provide a means of representing claims from a submodule that has its own attesting environment, i.e., it has keys distinct from the attester producing the surrounding token. Claims are represented in a signed EAT token.

Inclusion of a signed EAT as a claim cryptographically binds the EAT to the surrounding token. If it was conveyed in parallel with the surrounding token, there would be no such binding and attackers could substitute a good attestation from another device for the attestation of an errant subsystem.

A nested token need not use the same encoding as the enclosing token. This enables composite devices to be built without regards to the encoding used by components. Thus, a CBOR-encoded EAT can have a JSON-encoded EAT as a nested token and vice versa.

4.3. Claims Describing the Token

The claims in this section provide meta data about the token they occur in. They do not describe the entity. They may appear in evidence or attestation results.

4.3.1. iat (Timestamp) Claim

The "iat" claim defined in CWT and JWT is used to indicate the date-of-creation of the token, the time at which the claims are collected and the token is composed and signed.

The data for some claims may be held or cached for some period of time before the token is created. This period may be long, even days. Examples are measurements taken at boot or a geographic position fix taken the last time a satellite signal was received. There are individual timestamps associated with these claims to indicate their age is older than the "iat" timestamp.

CWT allows the use of floating-point for this claim. EAT disallows the use of floating-point. An EAT token MUST NOT contain an "iat" claim in floating-point format. Any recipient of a token with a floating-point format "iat" claim MUST consider it an error.

A 64-bit integer representation of the CBOR epoch-based time [RFC8949] used by this claim can represent a range of +/- 500 billion years, so the only point of a floating-point timestamp is to have precession greater than one second. This is not needed for EAT.

4.3.2. eat_profile (EAT Profile) Claim

See Section 6 for the detailed description of an EAT profile.

The "eat_profile" claim identifies an EAT profile by either a Uniform Resource Identifier (URI) or an Object Identifier (OID). Typically, the URI will reference a document describing the profile. An OID is just a unique identifier for the profile. It may exist anywhere in the OID tree. There is no requirement that the named document be publicly accessible. The primary purpose of the "eat_profile" claim is to uniquely identify the profile even if it is a private profile.

The OID is always absolute and never relative.

See Section 7.2.1 for OID and URI encoding.

```
$$Claims-Set-Claims //= (profile-label => general-uri / general-oid)
```

4.3.3. intuse (Intended Use) Claim

EATs may be employed in the context of several different applications. The "intuse" claim provides an indication to an EAT consumer about the intended usage of the token. This claim can be used as a way for an application using EAT to internally distinguish between different ways it utilizes EAT. 5 possible values for "intuse" are currently defined, but an IANA registry can be created in the future to extend these values based on new use cases of EAT.

1 -- Generic: Generic attestation describes an application where the EAT consumer requires the most up-to-date security assessment of the attesting entity. It is expected that this is the most commonly-used application of EAT.

2-- Registration: Entities that are registering for a new service may be expected to provide an attestation as part of the registration process. This "intuse" setting indicates that the attestation is not intended for any use but registration.

3 -- Provisioning: Entities may be provisioned with different values or settings by an EAT consumer. Examples include key material or device management trees. The consumer may require an EAT to assess entity security state of the entity prior to provisioning.

4 -- Certificate Issuance: Certification Authorities (CAs) may require attestation results (which in a background check model might require receiving evidence to be passed to a verifier) to make decisions about the issuance of certificates. An EAT may be used as part of the certificate signing request (CSR).

5 -- Proof-of-Possession: An EAT consumer may require an attestation as part of an accompanying proof-of-possession (PoP) application. More precisely, a PoP transaction is intended to provide to the recipient cryptographically-verifiable proof that the sender has possession of a key. This kind of attestation may be necessary to verify the security state of the entity storing the private key used in a PoP application.

```
$$Claims-Set-Claims //= ( intended-use-label => intended-use-type )

intended-use-type = generic /
                    registration /
                    provisioning /
                    csr /
                    pop

generic      = JC< "generic",      1 >
registration = JC< "registration", 2 >
provisioning = JC< "provisioning", 3 >
csr          = JC< "csr",          4 >
pop          = JC< "pop",          5 >
```

5. Detached EAT Bundles

A detached EAT bundle is a message to convey an EAT plus detached claims sets secured by that EAT. It is a top-level message like a CWT or JWT. It can occur in any place that a CWT or JWT occurs, for example as a submodule nested token as defined in Section 4.2.18.3.

A detached EAT bundle may be either CBOR or JSON-encoded.

A detached EAT bundle consists of two parts.

The first part is an encoded EAT as follows:

* MUST have at least one submodule that is a detached submodule digest as defined in Section 4.2.18.2

* MAY be either CBOR or JSON-encoded and doesn’t have to the the same as the encoding of the bundle

* MAY be a CWT, or JWT or some future-defined token type, but MUST NOT be a detached EAT bundle

* MUST be authenticity and integrity protected

The same mechanism for distinguishing the type for nested token submodules is employed here.

The second part is a map/object as follows:

* MUST be a Claims-Set

* MUST use the same encoding as the bundle

* MUST be wrapped in a byte string when the encoding is CBOR and be base64url-encoded when the encoding is JSON

For CBOR-encoded detached EAT bundles, tag TBD602 can be used to identify it. The standard rules apply for use or non-use of a tag. When it is sent as a submodule, it is always sent as a tag to distinguish it from the other types of nested tokens.

The digests of the detached claims sets are associated with detached Claims-Sets by label/name. It is up to the constructor of the detached EAT bundle to ensure the names uniquely identify the detached claims sets. Since the names are used only in the detached EAT bundle, they can be very short, perhaps one byte.

```
BUNDLE-Messages = BUNDLE-Tagged-Message / BUNDLE-Untagged-Message

BUNDLE-Tagged-Message   = #6.TBD602(BUNDLE-Untagged-Message)
BUNDLE-Untagged-Message = Detached-EAT-Bundle

Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => JC<json-wrapped-claims-set,
                     cbor-wrapped-claims-set>
    }
]

json-wrapped-claims-set = base64-url-text

cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

6. Profiles

EAT makes normative use of CBOR, JSON, COSE, JOSE, CWT and JWT. Most of these have implementation options to accommodate a range of use cases.

For example, COSE doesn’t require a particular set of cryptographic algorithms so as to accommodate different usage scenarios and evolution of algorithms over time. Section 10 of [RFC9052] describes the profiling considerations for COSE.

The use of encryption is optional for both CWT and JWT. Section 8 of [RFC7519] describes implementation requirement and recommendations for JWT.

Similarly, CBOR provides indefinite length encoding, which is not commonly used, but valuable for very constrained devices. For EAT itself, in a particular use case some claims will be used and others will not. Section 4 of [RFC8949] describes serialization considerations for CBOR.

For example a mobile phone use case may require the device make and model, and prohibit UEID and location for privacy reasons. The general EAT standard retains all this flexibility because it too is aimed to accommodate a broad range of use cases.

It is necessary to explicitly narrow these implementation options to guarantee interoperability. EAT chooses one general and explicit mechanism, the profile, to indicate the choices made for these implementation options for all aspects of the token.

Below is a list of the various issues that should be addressed by a profile.

The "eat_profile" claim in Section 4.3.2 provides a unique identifier for the profile a particular token uses.

A profile can apply to evidence or to attestation results or both.

6.1. Format of a Profile Document

A profile document doesn’t have to be in any particular format. It may be simple text, something more formal or a combination.

A profile may define, and possibly register, one or more new claims if needed. A profile may also reuse one or more already defined claims, either as-is or with values constrained to a subset or subrange.

6.2. Full and Partial Profiles

For a "full" profile, the receiver will be able to decode and verify every possible EAT sent when a sender and receiver both adhere to it. For a "partial" profile, there are still some protocol options left undecided.

For example, a profile that allows the use of signing algorithms by the sender that the receiver is not required to support is a partial profile. The sender might choose a signing algorithm that some receivers don’t support.

Full profiles MUST be complete such that a complying receiver can decode, verify and check for freshness every EAT created by a complying sender. A full profile MAY or MAY NOT require the receiver to fully handle every claim in an EAT from a complying sender. Profile specifications may assume the receiver has access to the necessary verification keys or may go into specific detail on the means to access verification keys.

The "eat_profile" claim MUST NOT be used to identify partial profiles.

While fewer profiles are preferrable, sometimes several may be needed for a use case. One approach to handling variation in devices might be to define several full profiles that are variants of each other. It is relatively easy and inexpensive to define profiles as they don't have to be standards track and don't have to be registered anywhere. For example, flexibility for post-quantum algorithms can be handled as follows. First, define a full profile for a set of non-post-quantum algorithms for current use. Then, when post-quantum algorithms are settled, define another full profile derived from the first.

6.3. List of Profile Issues

The following is a list of EAT, CWT, JWT, COSE, JOSE and CBOR options that a profile should address.

6.3.1. Use of JSON, CBOR or both

A profile should specify whether CBOR, JSON or both may be sent. A profile should specify that the receiver can accept all encodings that the sender is allowed to send.

This should be specified for the top-level and all nested tokens. For example, a profile might require all nested tokens to be of the same encoding of the top level token.

6.3.2. CBOR Map and Array Encoding

A profile should specify whether definite-length arrays/maps, indefinite-length arrays/maps or both may be sent. A profile should specify that the receiver be able to accept all length encodings that the sender is allowed to send.

This applies to individual EAT claims, CWT and COSE parts of the implementation.

For most use cases, specifying that only definite-length arrays/maps may be sent is suitable.

6.3.3. CBOR String Encoding

A profile should specify whether definite-length strings, indefinite-length strings or both may be sent. A profile should specify that the receiver be able to accept all types of string encodings that the sender is allowed to send.

For most use cases, specifying that only definite-length strings may be sent is suitable.

6.3.4. CBOR Preferred Serialization

A profile should specify whether or not CBOR preferred serialization must be sent or not. A profile should specify the receiver be able to accept preferred and/or non-preferred serialization so it will be able to accept anything sent by the sender.

6.3.5. CBOR Tags

The profile should specify whether the token should be a CWT Tag or not.

When COSE protection is used, the profile should specify whether COSE tags are used or not. Note that RFC 8392 requires COSE tags be used in a CWT tag.

Often a tag is unnecessary because the surrounding or carrying protocol identifies the object as an EAT.

6.3.6. COSE/JOSE Protection

COSE and JOSE have several options for signed, MACed and encrypted messages. JWT may use the JOSE NULL protection option. It is possible to implement no protection, sign only, MAC only, sign then encrypt and so on. All combinations allowed by COSE, JOSE, JWT, and CWT are allowed by EAT.

A profile should specify all signing, encryption and MAC message formats that may be sent. For example, a profile might allow only COSE_Sign1 to be sent. For another example, a profile might allow COSE_Sign and COSE_Encrypt to be sent to carry multiple signatures for post quantum cryptography and to use encryption to provide confidentiality.

A profile should specify the receiver accepts all message formats that are allowed to be sent.

When both signing and encryption are allowed, a profile should specify which is applied first.

6.3.7. COSE/JOSE Algorithms

See the section on "Application Profiling Considerations" in [RFC9052] for a discussion on selection of cryptographic algorithms and related issues.

The profile MAY require the protocol or system using EAT provide an algorithm negotiation mechanism.

If not, The profile document should list a set of algorithms for each COSE and JOSE message type allowed by the profile per Section 6.3.6. The verifier should implement all of them. The attester may implement any of them it wishes, possibly just one for each message type.

If detached submodule digests are used the profile should address the determination of the hash algorithm(s) for the digests.

6.3.8. Detached EAT Bundle Support

A profile should specify whether or not a detached EAT bundle (Section 5) can be sent. A profile should specify that a receiver be able to accept a detached EAT bundle if the sender is allowed to send it.

6.3.9. Key Identification

A profile should specify what must be sent to identify the verification, decryption or MAC key or keys. If multiple methods of key identification may be sent, a profile should require the receiver support them all.

Appendix F describes a number of methods for identifying verification keys. When encryption is used, there are further considerations. In some cases key identification may be very simple and in others involve multiple components. For example, it may be simple through use of COSE key ID or it may be complex through use of an X.509 certificate hierarchy.

While not always possible, a profile should specify or make reference to, a full end-end specification for key identification. For example, a profile should specify in full detail how COSE key IDs are

to be created, their lifecycle and such rather than just specifying that a COSE key ID be used. For example, a profile should specify the full details of an X.509 hierarchy including extension processing, algorithms allowed and so on rather than just saying X.509 certificates are used.

6.3.10. Endorsement Identification

Similar to, or perhaps the same as verification key identification, the profile may wish to specify how endorsements are to be identified. However note that endorsement identification is optional, whereas key identification is not.

6.3.11. Freshness

Security considerations, see Section 9.3, require a mechanism to provide freshness. This may be the EAT nonce claim in Section 4.1, or some claim or mechanism defined outside this document. The section on freshness in [RFC9334] describes several options. A profile should specify which freshness mechanism or mechanisms can be used.

If the EAT nonce claim is used, a profile should specify whether multiple nonces may be sent. If a profile allows multiple nonces to be sent, it should require the receiver to process multiple nonces.

6.3.12. Claims Requirements

A profile may define new claims that are not defined in this document.

This document requires an EAT receiver must accept tokens with claims it does not understand. A profile for a specific use case may reverse this and allow a receiver to reject tokens with claims it does not understand. A profile for a specific use case may specify that specific claims are prohibited.

A profile for a specific use case may modify this and specify that some claims are required.

A profile may constrain the definition of claims that are defined in this document or elsewhere. For example, a profile may require the EAT nonce be a certain length or the "location" claim always include the altitude.

Some claims are "pluggable" in that they allow different formats for their content. The "manifests" claim (Section 4.2.15) along with the measurement and "measurements" (Section 4.2.16) claims are examples

of this, allowing the use of CoSWID, SUIT Manifest and other formats. A profile should specify which formats are allowed to be sent, with the assumption that the corresponding CoAP content types have been registered. A profile should require the receiver to accept all formats that are allowed to be sent.

Further, if there is variation within a format that is allowed, the profile should specify which variations can be sent. For example, there are variations in the CoSWID format. A profile that require the receiver to accept all variations that are allowed to be sent.

6.4. The Constrained Device Standard Profile

It is anticipated that there will be many profiles defined for EAT for many different use cases. This section gives a normative definition of one profile that is good for many constrained device use cases.

The identifier for this profile is "urn:ietf:rfc:rfcTBD".

// RFC Editor: please replace rfcTBD with this RFC number and remove
// this note.

Issue	Profile Definition
CBOR/JSON	CBOR MUST be used
CBOR Encoding	Definite length maps and arrays MUST be used
CBOR Encoding	Definite length strings MUST be used
CBOR Serialization	Preferred serialization MUST be used
COSE Protection	COSE_Sign1 MUST be used
Algorithms	The receiver MUST accept ES256, ES384 and ES512; the sender MUST send one of these
Detached EAT Bundle Usage	Detached EAT bundles MUST not be sent with this profile
Verification Key Identification	Either the COSE kid or the UEID MUST be used to identify the verification key. If both are present, the kid takes precedence. (It is assumed the receiver has access to a database of trusted verification keys which allows lookup of the verification key ID; the key format and means of distribution are beyond the scope of this profile)
Endorsements	This profile contains no endorsement identifier
Freshness	A new single unique nonce MUST be used for every token request
Claims	No requirement is made on the presence or absence of claims other than requiring an EAT nonce. As per general EAT rules, the receiver MUST NOT error out on claims it doesn't understand.

Table 2: Constrained Device Profile Definition

Any profile with different requirements than those above MUST have a different profile identifier.

Note that many claims can be present for tokens conforming to this profile, even claims not defined in this document. Note also that even slight deviation from the above requirements is considered a different profile that MUST have a different identifier. For example, if a kid (key identifier) or UEID is not used for key identification, it is not in conformance with this profile. For another example, requiring the presence of some claim is also not in conformance and requires another profile.

Derivations of this profile are encouraged. For example another profile may be simply defined as The Constrained Device Standard Profile plus the requirement for the presence of claim xxxx and claim yyyy.

7. Encoding and Collected CDDL

An EAT is fundamentally defined using CDDL. This document specifies how to encode the CDDL in CBOR or JSON. Since CBOR can express some things that JSON can't (e.g., tags) or that are expressed differently (e.g., labels) there is some CDDL that is specific to the encoding.

7.1. Claims-Set and CDDL for CWT and JWT

CDDL was not used to define CWT or JWT. It was not available at the time.

This document defines CDDL for both CWT and JWT. This document does not change the encoding or semantics of anything in a CWT or JWT.

A Claims-Set is the central data structure for EAT, CWT and JWT. It holds all the claims and is the structure that is secured by signing or other means. It is not possible to define EAT, CWT, or JWT in CDDL without it. The CDDL definition of Claims-Set here is applicable to EAT, CWT and JWT.

This document specifies how to encode a Claims-Set in CBOR or JSON.

With the exception of nested tokens and some other externally defined structures (e.g., SWIDs) an entire Claims-Set must be in encoded in either CBOR or JSON, never a mixture.

CDDL for the seven claims defined by [RFC8392] and [RFC7519] is included here.

7.2. Encoding Data Types

This makes use of the types defined in [RFC8610] Appendix D, Standard Prelude.

7.2.1. Common Data Types

time-int is identical to the epoch-based time, but disallows floating-point representation.

For CBOR-encoded tokens, OIDs are specified using the CDDL type name "oid" from [RFC9090]. They are encoded without the tag number. For JSON-encoded tokens, OIDs are a text string in the common form of "nn.nn.nn...".

Unless expliclity indicated, URIs are not the URI tag defined in [RFC8949]. They are just text strings that contain a URI conforming to the format defined in [RFC3986].

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

binary-data = JC< base64-url-text, bstr>

base64-url-text = tstr .regexp "[A-Za-z0-9_-]+"

general-oid = JC< json-oid, ~oid >

json-oid = tstr .regexp "([0-2])((\\.0)|(\\.[1-9][0-9]*))*"

general-uri = JC< text, ~uri >

coap-content-format = uint .le 65535
```

7.2.2. JSON Interoperability

JSON should be encoded per [RFC8610], Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

* bstr -- MUST be base64url-encoded

* time -- MUST be encoded as NumericDate as described in Section 2 of [RFC7519].

* string-or-uri -- MUST be encoded as StringOrURI as described in Section 2 of [RFC7519].

* uri -- MUST be a URI [RFC3986].

* oid -- MUST be encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1") [RFC2252].

The CDDL generic "JC<>" is used in most places where there is a variance between CBOR and JSON. The first argument is the CDDL for JSON and the second is CDDL for CBOR.

7.2.3. Labels

Most map labels, Claims-Keys, Claim-Names and enumerated-type values are integers for CBOR-encoded tokens and strings for JSON-encoded tokens. When this is the case the "JC<>" CDDL construct is used to give both the integer and string values.

7.2.4. CBOR Interoperability

CBOR allows data items to be serialized in more than one form to accommodate a variety of use cases. This is addressed in Section 6.

7.3. Collected CDDL

7.3.1. Payload CDDL

This CDDL defines all the EAT Claims that are added to the main definition of a Claim-Set in Appendix D. Claims-Set is the payload for CWT, JWT and potentially other token types. This is for both CBOR and JSON. When there is variation between CBOR and JSON, the JC<> CDDL generic defined in Appendix D. Note that the JC<> generic uses the CDDL ".feature" control operator defined in [RFC9165].

This CDDL uses, but doesn’t define Submodule or nested tokens because the definition for these types varies between CBOR and JSON and the JC<> generic can’t be used to define it. The submodule claim is the one place where a CBOR token can be nested inside a JSON token and vice versa. Encoding-specific definitions are provided in the following sections.

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

binary-data = JC< base64-url-text, bstr>

base64-url-text = tstr .regexp "[A-Za-z0-9_-]+"

general-oid = JC< json-oid, ~oid >

json-oid = tstr .regexp "([0-2])((\\.0)|(\\.[1-9][0-9]*))*"

general-uri = JC< text, ~uri >

coap-content-format = uint .le 65535
```

```
$$Claims-Set-Claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

nonce-type = JC< tstr .size (8..88), bstr .size (8..64)>


$$Claims-Set-Claims //= (ueid-label => ueid-type)

ueid-type = JC<base64-url-text .size (10..44) , bstr .size (7..33)>

$$Claims-Set-Claims //= (sueids-label => sueids-type)

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

$$Claims-Set-Claims //= (
    oemid-label => oemid-pen / oemid-ieee / oemid-random
)

oemid-pen = int

oemid-ieee = JC<oemid-ieee-json, oemid-ieee-cbor>
oemid-ieee-cbor = bstr .size 3
oemid-ieee-json = base64-url-text .size 4

oemid-random = JC<oemid-random-json, oemid-random-cbor>
oemid-random-cbor = bstr .size 16
oemid-random-json = base64-url-text .size 24


$$Claims-Set-Claims //=  (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version:  tstr,
    ? scheme:  $version-scheme
]

$$Claims-Set-Claims //= (
    hardware-model-label => hardware-model-type
)

hardware-model-type = JC<base64-url-text .size (4..44),
                         bytes .size (1..32)>

$$Claims-Set-Claims //= ( sw-name-label => tstr )
```

```
$$Claims-Set-Claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version:  tstr
    ? scheme:  $version-scheme
]

$$Claims-Set-Claims //= (oem-boot-label => bool)

$$Claims-Set-Claims //= ( debug-status-label => debug-status-type )

debug-status-type = ds-enabled /
                    disabled /
                    disabled-since-boot /
                    disabled-permanently /
                    disabled-fully-and-permanently

ds-enabled                      = JC< "enabled", 0 >
disabled                        = JC< "disabled", 1 >
disabled-since-boot             = JC< "disabled-since-boot", 2 >
disabled-permanently            = JC< "disabled-permanently", 3 >
disabled-fully-and-permanently =
                       JC< "disabled-fully-and-permanently", 4 >

$$Claims-Set-Claims //= (location-label => location-type)

location-type = {
    latitude => number,
    longitude => number,
    ? altitude => number,
    ? accuracy => number,
    ? altitude-accuracy => number,
    ? heading => number,
    ? speed => number,
    ? timestamp => ~time-int,
    ? age => uint
}

latitude          = JC< "latitude",          1 >
longitude         = JC< "longitude",         2 >
altitude          = JC< "altitude",          3 >
accuracy          = JC< "accuracy",          4 >
altitude-accuracy = JC< "altitude-accuracy", 5 >
heading           = JC< "heading",           6 >
speed             = JC< "speed",             7 >
timestamp         = JC< "timestamp",         8 >
age               = JC< "age",               9 >
```

```
$$Claims-Set-Claims //= (uptime-label => uint)

$$Claims-Set-Claims //=  (boot-seed-label => binary-data)

$$Claims-Set-Claims //= (boot-count-label => uint)

$$Claims-Set-Claims //= ( intended-use-label => intended-use-type )

intended-use-type = generic /
                    registration /
                    provisioning /
                    csr /
                    pop

generic      = JC< "generic",      1 >
registration = JC< "registration", 2 >
provisioning = JC< "provisioning", 3 >
csr          = JC< "csr",          4 >
pop          = JC< "pop",          5 >

$$Claims-Set-Claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: general-uri
    dloa_platform_label: text
    ? dloa_application_label: text
]

$$Claims-Set-Claims //= (profile-label => general-uri / general-oid)

$$Claims-Set-Claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ manifest-format]

manifest-format = [
    content-type:   coap-content-format,
    content-format: JC< $manifest-body-json,
                        $manifest-body-cbor >
]

$manifest-body-cbor /= bytes .cbor untagged-coswid
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= bytes .cbor SUIT_Envelope
```

```
$manifest-body-json /= base64-url-text


$$Claims-Set-Claims //= (
    measurements-label => measurements-type
)

measurements-type = [+ measurements-format]

measurements-format = [
    content-type:   coap-content-format,
    content-format: JC< $measurements-body-json,
                        $measurements-body-cbor >
]

$measurements-body-cbor /= bytes .cbor untagged-coswid
$measurements-body-json /= base64-url-text


$$Claims-Set-Claims //= (
    measurement-results-label =>
        [ + measurement-results-group ] )

measurement-results-group = [
    measurement-system: tstr,
    measurement-results: [ + individual-result ]
]

individual-result = [
    result-id:  tstr / binary-data,
    result:     result-type,
]

result-type = comparison-successful /
              comparison-fail /
              comparison-not-run /
              measurement-absent

comparison-successful    = JC< "success",       1 >
comparison-fail          = JC< "fail",          2 >
comparison-not-run       = JC< "not-run",       3 >
measurement-absent       = JC< "absent",        4 >



Detached-Submodule-Digest = [
   hash-algorithm : text / int,
   digest         : binary-data
```

```
]


BUNDLE-Messages = BUNDLE-Tagged-Message / BUNDLE-Untagged-Message

BUNDLE-Tagged-Message   = #6.TBD602(BUNDLE-Untagged-Message)
BUNDLE-Untagged-Message = Detached-EAT-Bundle

Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => JC<json-wrapped-claims-set,
                     cbor-wrapped-claims-set>
    }
]

json-wrapped-claims-set = base64-url-text

cbor-wrapped-claims-set = bstr .cbor Claims-Set



nonce-label            = JC< "eat_nonce",  10 >
ueid-label             = JC< "ueid",       256 >
sueids-label           = JC< "sueids",     257 >
oemid-label            = JC< "oemid",      258 >
hardware-model-label   = JC< "hwmodel",    259 >
hardware-version-label = JC< "hwversion",  260 >
oem-boot-label         = JC< "oemboot",    262 >
debug-status-label     = JC< "dbgstat",    263 >
location-label         = JC< "location",   264 >
profile-label          = JC< "eat_profile",265 >
submods-label          = JC< "submods",    266 >

uptime-label           = JC< "uptime",     TBD >
boot-seed-label        = JC< "bootseed",   TBD >
intended-use-label     = JC< "intuse",     TBD >
dloas-label            = JC< "dloas",      TBD >
sw-name-label          = JC< "swname",     TBD >
sw-version-label       = JC< "swversion",  TBD >
manifests-label        = JC< "manifests",  TBD >
measurements-label     = JC< "measurements", TBD >
measurement-results-label = JC< "measres" , TBD >
boot-count-label       = JC< "bootcount",  TBD >
```

7.3.2. CBOR-Specific CDDL

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

$EAT-CBOR-Tagged-Token /= CWT-Tagged-Message
$EAT-CBOR-Tagged-Token /= BUNDLE-Tagged-Message

$EAT-CBOR-Untagged-Token /= CWT-Untagged-Message
$EAT-CBOR-Untagged-Token /= BUNDLE-Untagged-Message


Nested-Token = CBOR-Nested-Token

CBOR-Nested-Token =
    JSON-Token-Inside-CBOR-Token /
    CBOR-Token-Inside-CBOR-Token

CBOR-Token-Inside-CBOR-Token = bstr .cbor $EAT-CBOR-Tagged-Token

JSON-Token-Inside-CBOR-Token = tstr

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / CBOR-Nested-Token /
            Detached-Submodule-Digest
```

7.3.3. JSON-Specific CDDL

```
EAT-JSON-Token = $EAT-JSON-Token-Formats

$EAT-JSON-Token-Formats /= JWT-Message
$EAT-JSON-Token-Formats /= BUNDLE-Untagged-Message


Nested-Token = JSON-Selector

JSON-Selector = $JSON-Selector

$JSON-Selector /= [type: "JWT", nested-token: JWT-Message]
$JSON-Selector /= [type: "CBOR", nested-token: CBOR-Token-Inside-JSON-Token]
$JSON-Selector /= [type: "BUNDLE", nested-token: Detached-EAT-Bundle]
$JSON-Selector /= [type: "DIGEST", nested-token: Detached-Submodule-Digest]

CBOR-Token-Inside-JSON-Token = base64-url-text

$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / JSON-Selector
```

8. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity; therefore, implementations should consider privacy-preserving options dependent on the usage of the EAT. For example, the location claim might be suppressed in EATs sent to unauthenticated consumers.

8.1. UEID and SUEID Privacy Considerations

A UEID is usually not privacy-preserving. Relying parties receiving tokens that happen to be from a particular entity will be able to know the tokens are from the same entity and be able to identify the entity issuing those tokens.

Thus the use of the claim may violate privacy policies. In other usage situations a UEID will not be allowed for certain products like browsers that give privacy for the end user. It will often be the case that tokens will not have a UEID for these reasons.

An SUEID is also usually not privacy-preserving. In some cases it may have fewer privacy issues than a UEID depending on when and how and when it is generated.

There are several strategies that can be used to still be able to put UEIDs and SUEIDs in tokens:

* The entity obtains explicit permission from the user of the entity to use the UEID/SUEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID/SUEID.

* The UEID/SUEID is used only in a particular context or particular use case. It is used only by one relying party.

* The entity authenticates the relying party and generates a derived UEID/SUEID just for that particular relying party. For example, the relying party could prove their identity cryptographically to the entity, then the entity generates a UEID just for that relying party by hashing a proofed relying party ID with the main entity UEID/SUEID.

Note that some of these privacy preservation strategies result in multiple UEIDs and SUEIDs per entity. Each UEID/SUEID is used in a different context, use case or system on the entity. However, from the view of the relying party, there is just one UEID and it is still globally universal across manufacturers.

8.2. Location Privacy Considerations

Geographic location is most always considered personally identifiable information. Implementers should consider laws and regulations governing the transmission of location data from end user devices to servers and services. Implementers should consider using location management facilities offered by the operating system on the entity generating the attestation. For example, many mobile phones prompt the user for permission before sending location data.

8.3. Boot Seed Privacy Considerations

The "bootseed" claim is effectively a stable entity identifier within a given boot epoch. Therefore, it is not suitable for use in attestation schemes that are privacy-preserving.

8.4. Replay Protection and Privacy

EAT defines the EAT nonce claim for replay protection and token freshness. The nonce claim is based on a value usually derived remotely (outside of the entity). This claim might be used to extract and convey personally identifying information either inadvertently or by intention. For instance, an implementor may choose a nonce equivalent to a username associated with the device (e.g., account login). If the token is inspected by a 3rd-party then this information could be used to identify the source of the token or an account associated with the token. To avoid the conveyance of privacy-related information in the nonce claim, it should be derived using a salt that originates from a true and reliable random number generator or any other source of randomness that would still meet the target system requirements for replay protection and token freshness.

9. Security Considerations

The security considerations provided in Section 8 of [RFC8392] and Section 11 of [RFC7519] apply to EAT in its CWT and JWT form, respectively. Moreover, Chapter 12 of [RFC9334] is also applicable to implementations of EAT. In addition, implementors should consider the following.

9.1. Claim Trustworthiness

This specification defines semantics for each claim. It does not require any particular level of security in the implementation of the claims or even the attester itself. Such specification is far beyond the scope of this document which is about a message format not the security level of an implementation.

The receiver of an EAT comes to know the trustworthiness of the claims in it by understanding the implementation made by the attester vendor and/or understanding the checks and processing performed by the verifier.

For example, this document says that a UEID is permanent and that it must not change, but it doesn't say what degree of attack to change it must be defended.

The degree of security will vary from use case to use case. In some cases the receiver may only need to know something of the implementation such as that it was implemented in a TEE. In other cases the receiver may require the attester be certified by a particular certification program. Or perhaps the receiver is content with very little security.

9.2. Key Provisioning

Private key material can be used to sign and/or encrypt the EAT, or can be used to derive the keys used for signing and/or encryption. In some instances, the manufacturer of the entity may create the key material separately and provision the key material in the entity itself. The manufacturer of any entity that is capable of producing an EAT should take care to ensure that any private key material be suitably protected prior to provisioning the key material in the entity itself. This can require creation of key material in an enclave (see [RFC4949] for definition of "enclave"), secure transmission of the key material from the enclave to the entity using an appropriate protocol, and persistence of the private key material in some form of secure storage to which (preferably) only the entity has access.

9.2.1. Transmission of Key Material

Regarding transmission of key material from the enclave to the entity, the key material may pass through one or more intermediaries. Therefore some form of protection ("key wrapping") may be necessary. The transmission itself may be performed electronically, but can also be done by human courier. In the latter case, there should be minimal to no exposure of the key material to the human (e.g. encrypted portable memory). Moreover, the human should transport the key material directly from the secure enclave where it was created to a destination secure enclave where it can be provisioned.

9.3. Freshness

All EAT use MUST provide a freshness mechanism to prevent replay and related attacks. The extensive discussions on freshness in [RFC9334] including security considerations apply here. The EAT nonce claim, in Section 4.1, is one option to provide freshness.

9.4. Multiple EAT Consumers

In many cases, more than one EAT consumer may be required to fully verify the entity attestation. Examples include individual consumers for nested EATs, or consumers for individual claims with an EAT. When multiple consumers are required for verification of an EAT, it is important to minimize information exposure to each consumer. In addition, the communication between multiple consumers should be secure.

For instance, consider the example of an encrypted and signed EAT with multiple claims. A consumer may receive the EAT (denoted as the "receiving consumer"), decrypt its payload, verify its signature, but then pass specific subsets of claims to other consumers for evaluation ("downstream consumers"). Since any COSE encryption will be removed by the receiving consumer, the communication of claim subsets to any downstream consumer MUST leverage an equivalent communication security protocol (e.g. Transport Layer Security).

However, assume the EAT of the previous example is hierarchical and each claim subset for a downstream consumer is created in the form of a nested EAT. Then the nested EAT is itself encrypted and cryptographically verifiable (due to its COSE envelope) by a downstream consumer (unlike the previous example where a claims set without a COSE envelope is sent to a downstream consumer). Therefore, Transport Layer Security between the receiving and downstream consumers is not strictly required. Nevertheless, downstream consumers of a nested EAT should provide a nonce unique to the EAT they are consuming.

9.5. Detached EAT Bundle Digest Security Considerations

A detached EAT bundle is composed of a nested EAT and an claims set as per Section 5. Although the attached claims set is vulnerable to modification in transit, any modification can be detected by the receiver through the associated digest, which is a claim fully contained within an EAT. Moreover, the digest itself can only be derived using an appropriate COSE hash algorithm, implying that an attacker cannot induce false detection of modified detached claims because the algorithms in the COSE registry are assumed to be of sufficient cryptographic strength.

9.6. Verification Keys

In all cases there must be some way that the verification key is itself verified or determined to be trustworthy. The key identification itself is never enough. This will always be by some out-of-band mechanism that is not described here. For example, the verifier may be configured with a root certificate or a master key by the verifier system administrator.

Often an X.509 certificate or an endorsement carries more than just the verification key. For example, an X.509 certificate might have key usage constraints, and an endorsement might have reference values. When this is the case, the key identifier must be either a protected header or in the payload, such that it is cryptographically bound to the EAT. This is in line with the requirements in section 6 on Key Identification in JSON Web Signature [RFC7515].

10. IANA Considerations

10.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries

Claims defined for EAT are compatible with those of CWT and JWT so the CWT and JWT Claims Registries, [IANA.CWT.Claims] and [IANA.JWT.Claims], are re-used. No new IANA registry is created.

All EAT claims defined in this document are placed in both registries. All new EAT claims defined subsequently should be placed in both registries.

Appendix E describes some considerations when defining new claims.

10.2. CWT and JWT Claims Registered by This Document

This specification adds the following values to the "JSON Web Token Claims" registry established by [RFC7519] and the "CBOR Web Token Claims Registry" established by [RFC8392]. Each entry below is an addition to both registries.

The "Claim Description", "Change Controller" and "Specification Documents" are common and equivalent for the JWT and CWT registries. The "Claim Key" and "Claim Value Types(s)" are for the CWT registry only. The "Claim Name" is as defined for the CWT registry, not the JWT registry. The "JWT Claim Name" is equivalent to the "Claim Name" in the JWT registry.

IANA is requested to register the following claims. The "Claim Value Type(s)" here all name CDDL definitions and are only for the CWT registry.

// RFC editor: please see instructions in followg paragraph and
// remove for final publication

RFC Editor: Please make the following adjustments and remove this paragraph. Replace "*this document*" with this RFC number. In the following, the claims with "Claim Key: TBD" need to be assigned a value in the Specification Required Range, preferably starting around 267. Those below already with a Claim Key number were given early assignment. No change is requested for them except for Claim Key 262. Claim 262 should be renamed from "secboot" to "oemboot" in the JWT registry and its description changed in both the CWT and JWT registries.

* Claim Name: Nonce

* Claim Description: Nonce

* JWT Claim Name: "eat_nonce"

* Claim Key: 10

* Claim Value Type(s): bstr or array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: UEID

* Claim Description: The Universal Entity ID

* JWT Claim Name: "ueid"

* CWT Claim Key: 256

* Claim Value Type(s): bstr

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: SUEIDs

* Claim Description: Semi-permanent UEIDs

* JWT Claim Name: "sueids"

* CWT Claim Key: 257
* Claim Value Type(s): map
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Hardware OEM ID
* Claim Description: Hardware OEM ID
* JWT Claim Name: "oemid"
* Claim Key: 258
* Claim Value Type(s): bstr or int
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Hardware Model
* Claim Description: Model identifier for hardware
* JWT Claim Name: "hwmodel"
* Claim Key: 259
* Claim Value Type(s): bstr
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Hardware Version
* Claim Description: Hardware Version Identifier
* JWT Claim Name: "hwversion"
* Claim Key: TBD 260
* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: OEM Authorized Boot

* Claim Description: Indicates whether the software booted was OEM authorized

* JWT Claim Name: "oemboot"

* Claim Key: 262

* Claim Value Type(s): bool

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Debug Status

* Claim Description: Indicates status of debug facilities

* JWT Claim Name: "dbgstat"

* Claim Key: 263

* Claim Value Type(s): uint

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Location

* Claim Description: The geographic location

* JWT Claim Name: "location"

* Claim Key: 264

* Claim Value Type(s): map

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: EAT Profile
* Claim Description: Indicates the EAT profile followed
* JWT Claim Name: "eat_profile"
* Claim Key: 265
* Claim Value Type(s): uri or oid
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Submodules Section
* Claim Description: The section containing submodules
* JWT Claim Name: "submods"
* Claim Key: 266
* Claim Value Type(s): map
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Uptime
* Claim Description: Uptime
* JWT Claim Name: "uptime"
* Claim Key: TBD
* Claim Value Type(s): uint
* Change Controller: IETF
* Specification Document(s): *this document*

* Claim Name: Boot Count

* Claim Description: The number times the entity or submodule has been booted

* JWT Claim Name: "bootcount"

* Claim Key: TBD

* Claim Value Type(s): uint

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Boot Seed

* Claim Description: Identifies a boot cycle

* JWT Claim Name: "bootseed"

* Claim Key: TBD

* Claim Value Type(s): bstr

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: DLOAs

* Claim Description: Certifications received as Digital Letters of Approval

* JWT Claim Name: "dloas"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Name

* Claim Description: The name of the software running in the entity

* JWT Claim Name: "swname"

* Claim Key: TBD

* Claim Value Type(s): tstr

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Version

* Claim Description: The version of software running in the entity

* JWT Claim Name: "swversion"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Manifests

* Claim Description: Manifests describing the software installed on the entity

* JWT Claim Name: "manifests"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Measurements

* Claim Description: Measurements of the software, memory configuration and such on the entity

* JWT Claim Name: "measurements"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Software Measurement Results

* Claim Description: The results of comparing software measurements to reference values

* JWT Claim Name: "measres"

* Claim Key: TBD

* Claim Value Type(s): array

* Change Controller: IETF

* Specification Document(s): *this document*

* Claim Name: Intended Use

* Claim Description: Indicates intended use of the EAT

* JWT Claim Name: "intuse"

* Claim Key: TBD

* Claim Value Type(s): uint

* Change Controller: IETF

* Specification Document(s): *this document*

10.3. UEID URN Registered by this Document

IANA is requested to register the following new subtypes in the "DEV URN Subtypes" registry under "Device Identification". See [RFC9039].

Subtype	Description	Reference
ueid	Universal Entity Identifier	This document
sueid	Semi-permanent Universal Entity Identifier	This document

Table 3: UEID URN Registration

ABNF for these two URNs is as follows where b64ueid is the base64url-encoded binary byte-string for the UEID or SUEID:

```
body =/ ueidbody
ueidbody = %sueid: b64ueid
```

10.4. CBOR Tag for Detached EAT Bundle Registered by this Document

In the registry [IANA.cbor-tags], IANA is requested to allocate the following tag from the Specification Required space, with the present document as the specification reference.

Tag	Data Items	Semantics
TBD602	array	Detached EAT Bundle Section 5

Table 4: Detached EAT Bundle Tag Registration

11. References

11.1. Normative References

[DLOA] "Digital Letter of Approval", November 2015, <https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf>.

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

[IANA.COSE.Algorithms] IANA, "CBOR Object Signing and Encryption (COSE)", <https://www.iana.org/assignments/cose>.

[IANA.CWT.Claims]
IANA, "CBOR Web Token (CWT) Claims", <https://www.iana.org/assignments/cwt>.

[IANA.JWT.Claims]
IANA, "JSON Web Token (JWT)", <https://www.iana.org/assignments/jwt>.

[PEN] "Private Enterprise Number (PEN) Request", n.d., <https://pen.iana.org/pen/PenApplication.page>.

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

[RFC2252] Wahl, M., Coulbeck, A., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, DOI 10.17487/RFC2252, December 1997, <https://www.rfc-editor.org/info/rfc2252>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

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

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

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

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

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

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

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

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

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

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

[ThreeGPP.IMEI] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification", 2019, <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729>.

[WGS84] National Geospatial-Intelligence Agency (NGA), "WORLD GEODETIC SYSTEM 1984, NGA.STND.0036_1.0.0_WGS84", 8 July 2014, <https://earth-info.nga.mil/php/download.php?file=coord-wgs84>.

11.2. Informative References

[BirthdayAttack] "Birthday attack", <https://en.wikipedia.org/wiki/Birthday_attack.>.

[CBOR.Cert.Draft] Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-07, 20 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-07>.

[COSE.X509.Draft] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", Work in Progress, Internet-Draft, draft-ietf-cose-x509-09, 13 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-x509-09>.

[EAT.media-types] Lundblade, L., Birkholz, H., and T. Fossati, "EAT Media Types", Work in Progress, Internet-Draft, draft-ietf-rats-eat-media-type-05, 7 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-media-type-05>.

[IEEE-RA] "IEEE Registration Authority", <https://standards.ieee.org/products-services/regauth/index.html>.

[IEEE.802-2001] "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture", IEEE standard, DOI 10.1109/ieeestd.2014.6847097, July 2014, <https://doi.org/10.1109/ieeestd.2014.6847097>.

[IEEE.802.1AR] "IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity", IEEE standard, DOI 10.1109/ieeestd.2018.8423794, July 2018, <https://doi.org/10.1109/ieeestd.2018.8423794>.

[JTAG] "IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture", February 2010, <https://ieeexplore.ieee.org/document/5412866>.

[OUI.Guide] "Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", August 2017, <https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf>.

[OUI.Lookup] "IEEE Registration Authority Assignments", <https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries>.

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

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021, <https://www.rfc-editor.org/info/rfc9039>.

[SUIT.Manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. Rønningstad, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-24, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-24>.

[UCCS] Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-ietf-rats-uccs-07, 27 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-uccs-07>.

[W3C.GeoLoc]
Popescu, A., Ed., "Geolocation API Specification", W3C REC REC-geolocation-API-20131024, W3C REC-geolocation-API-20131024, 24 October 2013, <https://www.w3.org/TR/2013/REC-geolocation-API-20131024/>.

Appendix A. Examples

Most examples are shown as just a Claims-Set that would be a payload for a CWT, JWT, detached EAT bundle or future token types. The signing is left off so the Claims-Set is easier to see. Some examples of signed tokens are also given.

```
// RFC Editor: When the IANA values are permanently assigned, please
// contact the authors so the examples can be regenerated.
// Regeneration is required because IANA-assigned values are inside
// hex and based-64 encoded data and some of these are signed.
```

A.1. Claims Set Examples

A.1.1. Simple TEE Attestation

This is a simple attestation of a TEE that includes a manifest that is a payload CoSWID to describe the TEE's software.

```
/ This is an EAT payload that describes a simple TEE. /

{
    / eat_nonce /       10: h'48df7b172d70b5a18935d0460a73dd71',
    / oemboot /        262: true,
    / dbgstat /        263: 2, / disabled-since-boot /
    / manifests /      273: [
                              [
                               121, / CoAP Content ID. A     /
                                    / made up one until one  /
                                    / is assigned for CoSWID /

                               / This is byte-string wrapped      /
                               / payload CoSWID. It gives the TEE /
                               / software name, the version and   /
                               / the  name of the file it is in.  /
                               / {0: "3a24",                      /
                               /  12: 1,                          /
                               /   1: "Acme TEE OS",              /
                               /  13: "3.1.4",                    /
                               /   2: [{31: "Acme TEE OS", 33: 1}, /
                               /       {31: "Acme TEE OS", 33: 2}], /
                               /   6: {                           /
                               /       17: {                      /
                               /           24: "acme_tee_3.exe"   /
                               /       }                          /
                               /    }                             /
                               /  }                               /
                               h' a60064336132340c01016b
                                  41636d6520544545204f530d65332e31
                                  2e340282a2181f6b41636d6520544545
                                  204f53182101a2181f6b41636d652054
                                  4545204f5318210206a111a118186e61
                                  636d655f7465655f332e657865'
                              ]
                            ]
}
```

```
/ A payload CoSWID created by the SW vendor. All this really does /
/ is name the TEE SW, its version and lists the one file that     /
/ makes up the TEE. /

1398229316({
    / Unique CoSWID ID /    0: "3a24",
    / tag-version /        12: 1,
    / software-name /       1: "Acme TEE OS",
    / software-version /   13: "3.1.4",
    / entity /              2: [
                                   {
        / entity-name /                31: "Acme TEE OS",
        / role        /                33: 1 / tag-creator /
                                   },
                                   {
        / entity-name /                31: "Acme TEE OS",
        / role        /                33: 2 / software-creator /
                                   }
                               ],
    / payload /                6: {
        / ...file /                17: {
            / ...fs-name /             24: "acme_tee_3.exe"
                                   }
                               }
})
```

A.1.2. Submodules for Board and Device

```
/ This example shows use of submodules to give information  /
/ about the chip, board and overall device.                 /
/                                                           /
/ The main attestation is associated with the chip with the /
/ CPU and running the main OS. It is what has the keys and  /
/ produces the token.                                       /
/                                                           /
/ The board is made by a different vendor than the chip.    /
/ Perhaps it is some generic IoT board.                     /
/                                                           /
/ The device is some specific appliance that is made by a   /
/ different vendor than either the chip or the board.       /
/                                                           /
/ Here the board and device submodules aren't the typical   /
/ target environments as described by the RATS architecture /
/ document, but they are a valid use of submodules.         /

{
    / eat_nonce /       10: h'e253cabedc9eec24ac4e25bcbeaf7765'
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: h'894823', / IEEE OUI format OEM ID /
    / hwmodel /        259: h'549dcecc8b987c737b44e40f7c635ce8'
                              / Hash of chip model name /,
    / hwversion /      260: ["1.3.4", 1], / Multipartnumeric  /
    / swname /         271: "Acme OS",
    / swversion /      272: ["3.5.5", 1],
    / oemboot /        262: true,
    / dbgstat /        263: 3, / permanent-disable  /
    / timestamp (iat) /  6: 1526542894,
    / submods / 266: {
        / A submodule to hold some claims about the circuit board /
        "board" :  {
            / oemid /     258: h'9bef8787eba13e2c8f6e7cb4b1f4619a',
            / hwmodel /   259: h'ee80f5a66c1fb9742999a8fdab930893'
                                  / Hash of board module name /,
            / hwversion / 260: ["2.0a", 2] / multipartnumeric+sfx /
        },

        / A submodule to hold claims about the overall device /
        "device" :  {
            / oemid /     258: 61234, / PEN Format OEM ID /
            / hwversion / 260: ["4.0", 1] / Multipartnumeric /
        }
    }
}
```

A.1.3. EAT Produced by Attestation Hardware Block

```
/ This is an example of a token produced by a HW block            /
/ purpose-built for attestation.  Only the nonce claim changes    /
/ from one attestation to the next as the rest  either come       /
/ directly from the hardware or from one-time-programmable memory /
/ (e.g. a fuse). 47 bytes encoded in CBOR (8 byte nonce, 16 byte  /
/ UEID). /

{
    / eat_nonce /       10: h'd79b964ddd5471c1393c8888',
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: 64242, / Private Enterprise Number /
    / oemboot /        262: true,
    / dbgstat /        263: 3, / disabled-permanently /
    / hwversion /      260: [ "3.1", 1 ] / Type is multipartnumeric /
}
```

A.1.4. Key / Key Store Attestation

```
/ This is an attestation of a public key and the key store     /
/ implementation that protects and manages it. The key store   /
/ implementation is in a security-oriented execution           /
/ environment separate from the high-level OS (HLOS), for      /
/ example a Trusted Execution Environment (TEE). The key store /
/ is the Attester.                                             /
/                                                              /
/ There is some attestation of the high-level OS, just version /
/ and boot & debug status. It is a Claims-Set submodule because/
/ it has lower security level than the key store. The key      /
/ store's implementation has access to info about the HLOS, so /
/ it is able to include it.                                    /
/                                                              /
/ A key and an indication of the user authentication given to  /
/ allow access to the key is given. The labels for these are   /
/ in the private space since this is just a hypothetical       /
/ example, not part of a standard protocol.                    /


{
    / eat_nonce /       10: h'99b67438dba40743266f70bf75feb1026d5134
                              97a229bfe8'
    / oemboot /        262: true,
    / dbgstat /        263: 2, / disabled-since-boot /
    / manifests /      273: [
                                [ 121, / CoAP Content ID. A      /
                                       / made up one until one   /
                                       / is assigned for CoSWID  /
                                  h'a60068376262333438376 6
                                    0c000169436172626f6e6974650d6331
```

```
                                        2e320e0102a2181f75496e6475737472
                                        69616c204175746f6d6174696f6e1821
                                        02'
                                    ]
                                    / Above is an encoded CoSWID     /
                                    / with the following data        /
                                    /   SW Name: "Carbonite"         /
                                    /   SW Vers: "1.2"               /
                                    /   SW Creator:                  /
                                    /      "Industrial Automation"   /
                                ],
        / exp /              4: 1634324274, / 2021-10-15T18:57:54Z /
        / iat /              6: 1634317080, / 2021-10-15T16:58:00Z /
                       -80000 : "fingerprint",
                       -80001 : { / The key -- A COSE_Key  /
                    / kty /       1: 2, / EC2, eliptic curve with x & y /
                    / kid /       2: h'36675c206f96236c3f51f54637b94ced',
                    / curve /    -1: 2, / curve is P-256 /
                    / x-coord /  -2: h'65eda5a12577c2bae829437fe338701a
                                      10aaa375e1bb5b5de108de439c08551d',
                    / y-coord /  -3: h'1e52ed75701163f7f9e40ddf9f341b3d
                                      c9ba860af7e0ca7ca7e9eecd0084d19c'
                 },

        / submods /        266 : {
                                "HLOS" : { / submod for high-level OS /
             / eat_nonce /         10: h'8b0b28782a23d3f6',
               / oemboot /        262: true,
               / manifests /      273: [
                                       [ 121, / CoAP Content ID. A      /
                                              / made up one until one   /
                                              / is assigned for CoSWID  /
                                         h'a600687337
                                           6537346b78380c000168
                                           44726f6964204f530d65
                                           52322e44320e0302a218
                                           1F75496E647573747269
                                           616c204175746f6d6174
                                           696f6e182102'
                                         ]
                                         / Above is an encoded CoSWID /
                                         / with the following data:   /
                                         /   SW Name: "Droid OS"      /
                                         /   SW Vers: "R2.D2"         /
                                         /   SW Creator:              /
                                         /     "Industrial Automation"/
                                       ]
                                }
```

```
                        }
   }
```

A.1.5. Software Measurements of an IoT Device

This is a simple token that might be for an IoT device. It includes CoSWID format measurments of the SW. The CoSWID is in byte-string wrapped in the token and also shown in diagnostic form.

```
/ This EAT payload is for an IoT device with a TEE. The attestation /
/ is produced by the TEE. There is a submodule for the IoT OS (the  /
/ main OS of the IoT device that is not as secure as the TEE). The  /
/ submodule contains claims for the IoT OS. The TEE also measures   /
/ the IoT OS and puts the measurements in the submodule.            /

{
    / eat_nonce / 10: h'5e19fba4483c7896'
    / oemboot /  262: true,
    / dbgstat /  263: 2, / disabled-since-boot /
    / oemid /    258: h'8945ad', / IEEE CID based /
    / ueid /     256: h'0198f50a4ff6c05861c8860d13a638ea',
    / submods /  266: {
                        "OS" : {
        / oemboot /         262: true,
        / dbgstat /         263: 2, / disabled-since-boot /
        / measurements /    274: [
                                    [
                                      121, / CoAP Content ID. A     /
                                           / made up one until one  /
                                           / is assigned for CoSWID /

                                     / This is a byte-string wrapped /
                                     / evidence CoSWID. It has       /
                                     / hashes of the main files of   /
                                     / the IoT OS.  /
                                     h'a600663463613234350c
                                       17016d41636d6520522d496f542d4f
                                       530d65332e312e3402a2181f724163
                                       6d65204261736520417474657374 65
                                       7218210103a11183a318187161636d
                                       655f725f696f745f6f732e65786514
                                       1a0044b349078201582005f6b327c1
                                       73b4192bd2c3ec248a292215eab456
                                       611bf7a783e25c1782479905a31818
                                       6d7265736f75726365732e72736314
                                       1a000c38b10782015820c142b9aba4
                                       280c4bb8c75f716a43c99526694caa
```

```
                                        be529571f5569bb7dc542f98a31818
                                        6a636f6d6d6f6e2e6c6962141a0023
                                        3d3b0782015820a6a9dcdfb3884da5
                                        f884e4e1e8e8629958c2dbc7027414
                                        43a913e34de9333be6'
                                      ]
                                    ]
                                  }
                                }
   }

   / An evidence CoSWID created for the "Acme R-IoT-OS" created by /
   / the "Acme Base Attester" (both fictious names).  It provides  /
   / measurements of the SW (other than the attester SW) on the    /
   / device. /

   1398229316({
       / Unique CoSWID ID /    0: "4ca245",
       / tag-version /        12: 23, / Attester-maintained counter /
       / software-name /       1: "Acme R-IoT-OS",
       / software-version /   13: "3.1.4",
       / entity /              2: {
           / entity-name /        31: "Acme Base Attester",
           / role        /        33: 1 / tag-creator /
                                },
       / evidence /            3: {
           / ...file /             17: [
                                       {
               / ...fs-name /              24: "acme_r_iot_os.exe",
               / ...size    /              20: 4502345,
               / ...hash    /               7: [
                                                1, / SHA-256 /
                                                h'05f6b327c173b419
                                                  2bd2c3ec248a2922
                                                  15eab456611bf7a7
                                                  83e25c1782479905'
                                               ]
                                       },
                                       {
               / ...fs-name /              24: "resources.rsc",
               / ...size    /              20: 800945,
               / ...hash    /               7: [
                                                 1, / SHA-256 /
                                                h'c142b9aba4280c4b
                                                  b8c75f716a43c995
                                                  26694caabe529571
                                                  f5569bb7dc542f98'
                                               ]
```

```
                                        },
                                        {
                / ...fs-name /              24: "common.lib",
                / ...size    /              20: 2309435,
                / ...hash    /               7: [
                                                1, / SHA-256 /
                                                h'a6a9dcdfb3884da5
                                                  f884e4e1e8e86299
                                                  58c2dbc702741443
                                                  a913e34de9333be6'
                                             ]
                                        }
                                    ]
                                }
   })
```

A.1.6. Attestation Results in JSON

This is a JSON-encoded payload that might be the output of a verifier that evaluated the IoT Attestation example immediately above.

This particular verifier knows enough about the TEE attester to be able to pass claims like debug status directly through to the relying party. The verifier also knows the reference values for the measured software components and is able to check them. It informs the relying party that they were correct in the "measres" claim. "Trustus Verifications" is the name of the services that verifies the software component measurements.

```
{
   "eat_nonce": "jkd8KL-8=Qlzg4",
   "oemboot": true,
   "dbgstat": "disabled-since-boot",
   "oemid": "iUWt",
   "ueid": "AZj1Ck_2wFhhyIYNE6Y4",
   "swname": "Acme R-IoT-OS",
   "swversion": [
      "3.1.4"
   ],
   "measres": [
      [
         "Trustus Measurements",
         [
            [
               "all",
               "success"
            ]
         ]
      ]
   ]
}
```


A.1.7. JSON-encoded Token with Submodules

This example has its lines wrapped per [RFC8792].

```
{
   "eat_nonce": "lI-IYNE6Rj6O",
   "ueid": "AJj1Ck_2wFhhyIYNE6Y46g==",
   "secboot": true,
   "dbgstat": "disabled-permanently",
   "iat": 1526542894,
   "submods": {
      "Android App Foo": {
         "swname": "Foo.app"
      },
      "Secure Element Eat": [
         "CBOR",
         "2D3ShEOhASagWGaoCkiUj4hg0TpGPhkBAFABmPUKT_bAWGHIhg0TpjjqGQ\
ECGfryGQEFBBkBBvUZAQcDGQEEgmMzLjEBGQEKoWNURUWCL1gg5c-V_ST6txRGdC3VjU\
Pa4XjlX-K5QpGpKRCC_8JjWgtYQPaQywOIZ3-mJKN3X9fLxOhAnsmBa-MvpHRzOw-Ywn\
-67bvJljuctezAPD41s6_At7NbSV3qwJlxIuqGfwe41es="
      ],
      "Linux Android": {
         "swname": "Android"
      },
      "Subsystem J": [
         "JWT",
         "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJKLUF0dGVzd\
GVyIiwiaWF0IjoxNjUxNzc0ODY4LCJleHAiOm51bGwsImF1ZCI6IiIsInN1YiI6IiJ9.\
gjw4nFMhLpJUuPXvMPzK1GMjhyJq2vWXg1416XKszwQ"
      ]
   }
}
```


A.2. Signed Token Examples

A.2.1. Basic CWT Example

This is a simple CWT-format token signed with the ECDSA algorithm.

```
/ This is a full CWT-format token with a very simple payloal. /
/ The main structure visible here is that of the COSE_Sign1.  /

61( 18( [
    h'A10126',                          / protected headers  /
    {},                             / empty unprotected headers /
    h'A20B46024A6B0978DE0A49000102030405060708',    / payload /
    h'9B9B2F5E470000F6A20C8A4157B5763FC45BE759
      9A5334028517768C21AFFB845A56AB557E0C8973
      A07417391243A79C478562D285612E292C622162
      AB233787'                                     / signature /
] ) )
```

A.2.2. CBOR-encoded Detached EAT Bundle

In this detached EAT bundle, the main token is produced by a HW attestation block. The detached Claims-Set is produced by a TEE and is largely identical to the Simple TEE examples above. The TEE digests its Claims-Set and feeds that digest to the HW block.

In a better example the attestation produced by the HW block would be a CWT and thus signed and secured by the HW block. Since the signature covers the digest from the TEE that Claims-Set is also secured.

The detached EAT bundle itself can be assembled by untrusted software.

```
/ This is a detached EAT bundle tag.  Note that 602, the tag /
/ identifying a detached EAT bundle is not yet registered /
/ with IANA /

602([

    / First part is a full EAT token with claims like nonce and /
    / UEID. Most importantly, it includes a submodule that is a /
    / detached digest which is the hash of the "TEE" claims set /
    / in the next section. The COSE payload follows:            /
    / { /
    /      10: h'948F8860D13A463E', /
    /     256: h'0198F50A4FF6C05861C8860D13A638EA', /
    /     258: 64242, /
    /     262: true, /
    /     263: 3, /
    /     260: ["3.1", 1], /
    /     266: { /
    /         "TEE": [ /
    /             -16, /
    /              h'8DEF652F47000710D9F466A4C666E209  /
    /                DD74F927A1CEA352B03143E188838ABE' /
    /         ] /
    /     } /
    /   }  /
    h'D83DD28443A10126A05866A80A48948F8860D13A463E1901
      00500198F50A4FF6C05861C8860D13A638EA19010219FAF2
      19010504190106F519010703190104826333 2E310119010A
      A163544545822F58208DEF652F47000710D9F466A4C666E2
      09DD74F927A1CEA352B03143E188838ABE5840F690CB0388
      677FA624A3775FD7CBC4E8409EC9816BE32FA474733B0F98
      C27FBAEDBBC9963B9CB5ECC03C3E35B3AFC0B7B35B495DEA
      C0997122EA867F07B8D5EB',
    {
       / A CBOR-encoded byte-string wrapped EAT claims-set. It /
       / contains claims suitable for a TEE                    /
       "TEE" : h'a40a48948f8860d13a463e190106f519010702
                 190111818218795858a60064336132340c0101
                 6b41636d6520544545204f530d65332e312e34
                 0282a2181f6b41636d6520544545204f531821
                 01a2181f6b41636d6520544545204f53182102
                 06a111a118186e61636d655f7465655f332e65
                 7865'
    }
 ])
```

```
/ This example contains submodule that is a detached digest,   /
/ which is the hash of a Claims-Set convey outside this token. /
/ Other than that is is the other example of a token from an   /
/ attestation HW block                                         /

{
    / eat_nonce /       10: h'3515744961254b41a6cf9c02',
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: 64242, / Private Enterprise Number /
    / oemboot /        262: true,
    / dbgstat /        263: 3, / disabled-permanently /
    / hwversion /      260: [ "3.1", 1 ], / multipartnumeric /
    / submods/         266: {
                                "TEE": [ / detached digest submod /
                                           -16, / SHA-256 /
                                           h'e5cf95fd24fab7144674
                                             2dd58d43dae178e55fe2
                                             b94291a9291082ffc263
                                             5a0b'
                                         ]
                              }
}
```

A.2.3. JSON-encoded Detached EAT Bundle

In this bundle there are two detached Claims-Sets, "Audio Subsystem" and "Graphics Subsystem". The JWT at the start of the bundle has detached signature submodules with hashes that cover these two Claims-Sets. The JWT itself is protected using HMAC with a key of "xxxxxx".

This example has its lines wrapped per [RFC8792].

```
[
    [
        "JWT",
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlYXRfbm9uY2UiOiJ5dT\
c2Tk44SXVWNmUiLCJzdWJtb2RzIjp7IkF1ZGlvIFN1YnN5c3RlbSI6WyJESUdFU1QiLF\
siU0hBLTI1NiIsIkZSRW4yVlR3aTk5cWNNRVFzYmxtTVFnM2I1b2ZYUG5OM1BJYW5CME\
5RT3MiXV0sIkdyYXBoaWNzIFN1YnN5c3RlbSI6WyJESUdFU1QiLFsiU0hBLTI1NiIsIk\
52M3NqUVU3Q1Z0RFRka0RTUlhWcFZDNUNMVFBCWmVQWWhTLUhoVlZWMXMiXV19fQ.FYs\
7R-TKhgAk85NyCOPQlbtGGjFM_3chnhBEOuM6qCo"
    ],
    {
        "Audio Subsystem" : "ewogICAgImVhdF9ub25jZSI6ICJsSStJWU5FNlJ\
qNk8iLAogICAgInVlaWQiOiAiQWROSlU0b1lYdFVwQStIeDNqQTcvRFEiCiAgICAib2V\
taWQiOiAiaVVXdCIsCiAgICAib2VtYm9vdCI6IHRydWUsIAogICAgInN3bmFtZSI6ICJ\
BdWRpbyBQcm9jZXNzb3IgT1MiCn0K",
        "Graphics Subsystem" : "ewogICAgImVhdF9ub25jZSI6ICJZWStJWU5F\
NlJqNk8iLAogICAgInVlaWQiOiAiQWVUTUlRQ1NVMnhWQmtVdGlndHU3bGUiCiAgICAi\
b2VtaWQiOiA3NTAwMCwKICAgICJvZW1ib290IjogdHJ1ZSwgCiAgICAic3duYW1lIjog\
IkdyYXBoaWNzIE9TIgp9Cg"
    }
]
```


Appendix B. UEID Design Rationale

B.1. Collision Probability

This calculation is to determine the probability of a collision of type 0x01 UEIDs given the total possible entity population and the number of entities in a particular entity management database.

Three different sized databases are considered. The number of devices per person roughly models non-personal devices such as traffic lights, devices in stores they shop in, facilities they work in and so on, even considering individual light bulbs. A device may have individually attested subsystems, for example parts of a car or a mobile phone. It is assumed that the largest database will have at most 10% of the world’s population of devices. Note that databases that handle more than a trillion records exist today.

The trillion-record database size models an easy-to-imagine reality over the next decades. The quadrillion-record database is roughly at the limit of what is imaginable and should probably be accommodated. The 100 quadrillion database is highly speculative perhaps involving nanorobots for every person, livestock animal and domesticated bird. It is included to round out the analysis.

Note that the items counted here certainly do not have IP address and are not individually connected to the network. They may be connected to internal buses, via serial links, Bluetooth and so on. This is not the same problem as sizing IP addresses.

People	Devices / Person	Subsystems / Device	Database Portion	Database Size
10 billion	100	10	10%	trillion (10^12)
10 billion	100,000	10	10%	quadrillion (10^15)
100 billion	1,000,000	10	10%	100 quadrillion (10^17)

Table 5: Entity Database Size Examples

This is conceptually similar to the Birthday Problem where m is the number of possible birthdays, always 365, and k is the number of people. It is also conceptually similar to the Birthday Attack where collisions of the output of hash functions are considered.

The proper formula for the collision calculation is

$$p = 1 - e^{-k^2/(2n)}$$

p Collision Probability
n Total possible population
k Actual population

However, for the very large values involved here, this formula requires floating point precision higher than commonly available in calculators and software so this simple approximation is used. See [BirthdayAttack].

$$p = k^2 / 2n$$

For this calculation:

p Collision Probability
n Total population based on number of bits in UEID
k Population in a database

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	2 * 10^-15	8 * 10^-35	5 * 10^-55
quadrillion (10^15)	2 * 10^-09	8 * 10^-29	5 * 10^-49
100 quadrillion (10^17)	2 * 10^-05	8 * 10^-25	5 * 10^-45

Table 6: UEID Size Options

Next, to calculate the probability of a collision occurring in one year's operation of a database, it is assumed that the database size is in a steady state and that 10% of the database changes per year. For example, a trillion record database would have 100 billion states per year. Each of those states has the above calculated probability of a collision.

This assumption is a worst-case since it assumes that each state of the database is completely independent from the previous state. In reality this is unlikely as state changes will be the addition or deletion of a few records.

The following tables gives the time interval until there is a probability of a collision based on there being one tenth the number of states per year as the number of records in the database.

```
t = 1 / ((k / 10) * p)

t  Time until a collision
p  Collision probability for UEID size
k  Database size
```

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	60,000 years	10^24 years	10^44 years
quadrillion (10^15)	8 seconds	10^14 years	10^34 years
100 quadrillion (10^17)	8 microseconds	10^11 years	10^31 years

Table 7: UEID Collision Probability

Clearly, 128 bits is enough for the near future thus the requirement that type 0x01 UEIDs be a minimum of 128 bits.

There is no requirement for 256 bits today as quadrillion-record databases are not expected in the near future and because this time-to-collision calculation is a very worst case. A future update of the standard may increase the requirement to 256 bits, so there is a requirement that implementations be able to receive 256-bit UEIDs.

B.2. No Use of UUID

A UEID is not a Universally Unique Identifier (UUID) [RFC4122] by conscious choice for the following reasons.

UUIDs are limited to 128 bits which may not be enough for some future use cases.

Today, cryptographic-quality random numbers are available from common CPUs and hardware. This hardware was introduced between 2010 and 2015. Operating systems and cryptographic libraries give access to this hardware. Consequently, there is little need for implementations to construct such random values from multiple sources on their own.

Version 4 UUIDs do allow for use of such cryptographic-quality random numbers, but do so by mapping into the overall UUID structure of time and clock values. This structure is of no value here yet adds complexity. It also slightly reduces the number of actual bits with entropy.

The design of UUID accommodates the construction of a unique identifier by combination of several identifiers that separately do not provide sufficient uniqueness. UEID takes the view that this construction is no longer needed, in particular because cryptographic-quality random number generators are readily available. It takes the view that hardware, software and/or manufacturing process implement UEID in a simple and direct way.

Note also that that a type 2 UEID (EUI/MAC) is only 7 bytes compared to 16 for a UUID.

Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)

This section describes several distinct ways in which an IEEE IDevID [IEEE.802.1AR] relates to EAT, particularly to UEID and SUEID.

[IEEE.802.1AR] orients around the definition of an implementation called a "DevID Module." It describes how IDevIDs and LDevIDs are stored, protected and accessed using a DevID Module. A particular level of defense against attack that should be achieved to be a DevID is defined. The intent is that IDevIDs and LDevIDs can be used with any network protocol or message format. In these protocols and message formats the DevID secret is used to sign a nonce or similar to prove the association of the DevID certificates with the device.

By contrast, EAT standardizes a message format that is sent to a relying party, the very thing that is not defined in [IEEE.802.1AR]. Nor does EAT give details on how keys, data and such are stored protected and accessed. EAT is intended to work with a variety of different on-device implementations ranging from minimal protection of assets to the highest levels of asset protection. It does not define any particular level of defense against attack, instead providing a set of security considerations.

EAT and DevID can be viewed as complimentary when used together or as competing to provide a device identity service.

C.1. DevID Used With EAT

As just described, EAT standardizes a message format and [IEEE.802.1AR] doesn’t. Vice versa, EAT doesn’t define a an device implementation, but DevID does.

Hence, EAT can be the message format that a DevID is used with. The DevID secret becomes the attestation key used to sign EATs. The DevID and its certificate chain become the endorsement sent to the verifier.

In this case, the EAT and the DevID are likely to both provide a device identifier (e.g. a serial number). In the EAT it is the UEID (or SUEID). In the DevID (used as an endorsement), it is a device serial number included in the subject field of the DevID certificate. It is probably a good idea in this use for them to be the same serial number or for the UEID to be a hash of the DevID serial number.

C.2. How EAT Provides an Equivalent Secure Device Identity

The UEID, SUEID and other claims like OEM ID are equivalent to the secure device identity put into the subject field of a DevID certificate. These EAT claims can represent all the same fields and values that can be put in a DevID certificate subject. EAT explicitly and carefully defines a variety of useful claims.

EAT secures the conveyance of these claims by having them signed on the device by the attestation key when the EAT is generated. EAT also signs the nonce that gives freshness at this time. Since these claims are signed for every EAT generated, they can include things that vary over time like GPS location.

DevID secures the device identity fields by having them signed by the manufacturer of the device sign them into a certificate. That certificate is created once during the manufacturing of the device and never changes so the fields cannot change.

So in one case the signing of the identity happens on the device and the other in a manufacturing facility, but in both cases the signing of the nonce that proves the binding to the actual device happens on the device.

While EAT does not specify how the signing keys, signature process and storage of the identity values should be secured against attack, an EAT implementation may have equal defenses against attack. One reason EAT uses CBOR is because it is simple enough that a basic EAT implementation can be constructed entirely in hardware. This allows EAT to be implemented with the strongest defenses possible.

C.3. An X.509 Format EAT

It is possible to define a way to encode EAT claims in an X.509 certificate. For example, the EAT claims might be mapped to X.509 v3 extensions. It is even possible to stuff a whole CBOR-encoded unsigned EAT token into a X.509 certificate.

If that X.509 certificate is an IDevID or LDevID, this becomes another way to use EAT and DevID together.

Note that the DevID must still be used with an authentication protocol that has a nonce or equivalent. The EAT here is not being used as the protocol to interact with the rely party.

C.4. Device Identifier Permanence

In terms of permanence, an IDevID is similar to a UEID in that they do not change over the life of the device. They cease to exist only when the device is destroyed.

An SUEID is similar to an LDevID. They change on device life-cycle events.

[IEEE.802.1AR] describes much of this permanence as resistant to attacks that seek to change the ID. IDevID permanence can be described this way because [IEEE.802.1AR] is oriented around the definition of an implementation with a particular level of defense against attack.

EAT is not defined around a particular implementation and must work on a range of devices that have a range of defenses against attack. EAT thus can’t be defined permanence in terms of defense against attack. EAT’s definition of permanence is in terms of operations and device lifecycle.

Appendix D. CDDL for CWT and JWT

[RFC8392] was published before CDDL was available and thus is specified in prose, not CDDL. Following is CDDL specifying CWT as it is needed to complete this specification. This CDDL also covers the Claims-Set for JWT.

Note that Section 4.3.1 requires that the iat claim be the type ~time-int (Section 7.2.1), not the type ~time when it is used in an EAT as floating-point values are not allowed for the "iat" claim in EAT.

The COSE-related types in this CDDL are defined in [RFC9052].

This however is NOT a normative or standard definition of CWT or JWT in CDDL. The prose in CWT and JWT remain the normative definition.

```
; This is replicated from draft-ietf-rats-uccs

Claims-Set = {
    * $$Claims-Set-Claims
    * Claim-Label .feature "extended-claims-label" => any
}
Claim-Label = int / text
string-or-uri = text

$$Claims-Set-Claims //= ( iss-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( sub-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( aud-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( exp-claim-label => ~time )
$$Claims-Set-Claims //= ( nbf-claim-label => ~time )
$$Claims-Set-Claims //= ( iat-claim-label => ~time )
$$Claims-Set-Claims //= ( cti-claim-label => bytes )

iss-claim-label = JC<"iss", 1>
sub-claim-label = JC<"sub", 2>
aud-claim-label = JC<"aud", 3>
exp-claim-label = JC<"exp", 4>
nbf-claim-label = JC<"nbf", 5>
iat-claim-label = JC<"iat", 6>
cti-claim-label = CBOR-ONLY<7>  ; jti in JWT: different name and text

JSON-ONLY<J> = J .feature "json"
CBOR-ONLY<C> = C .feature "cbor"

JC<J,C> = JSON-ONLY<J> / CBOR-ONLY<C>

; A JWT message is either a JWS or JWE in compact serialization form
; with the payload a Claims-Set. Compact serialization is the
; protected headers, payload and signature, each b64url encoded and
; separated by a ".". This CDDL simply matches top-level syntax of of
; a JWS or JWE since it is not possible to do more in CDDL.

JWT-Message =
   text .regexp "[A-Za-z0-9_-]+\\.[A-Za-z0-9_-]+\\.[A-Za-z0-9_-]+"


; Note that the payload of a JWT is defined in claims-set.cddl. That
; definition is common to CBOR and JSON.
```

```
; This is some CDDL describing a CWT at the top level This is
; not normative. RFC 8392 is the normative definition of CWT.

CWT-Messages = CWT-Tagged-Message / CWT-Untagged-Message

; The payload of the COSE_Message is always a Claims-Set

; The contents of a CWT Tag must always be a COSE tag
CWT-Tagged-Message = #6.61(COSE_Tagged_Message)

; An untagged CWT may be a COSE tag or not
CWT-Untagged-Message = COSE_Messages
```

Appendix E. New Claim Design Considerations

The following are design considerations that may be helpful to take into account when creating new EAT claims. It is the product of discussion in the working group.

EAT reuses the CWT and JWT claims registries. There is no registriy exclusively for EAT claims. This is not an update to the expert review criteria for the JWT and CWT claims registries as that would be an overreach for this document.

E.1. Interoperability and Relying Party Orientation

It is a broad goal that EATs can be processed by relying parties in a general way regardless of the type, manufacturer or technology of the device from which they originate. It is a goal that there be general-purpose verification implementations that can verify tokens for large numbers of use cases with special cases and configurations for different device types. This is a goal of interoperability of the semantics of claims themselves, not just of the signing, encoding and serialization formats.

This is a lofty goal and difficult to achieve broadly requiring careful definition of claims in a technology neutral way. Sometimes it will be difficult to design a claim that can represent the semantics of data from very different device types. However, the goal remains even when difficult.

E.2. Operating System and Technology Neutral

Claims should be defined such that they are not specific to an operating system. They should be applicable to multiple large high-level operating systems from different vendors. They should also be applicable to multiple small embedded operating systems from multiple vendors and everything in between.

Claims should not be defined such that they are specific to a software environment or programming language.

Claims should not be defined such that they are specific to a chip or particular hardware. For example, they should not just be the contents of some HW status register as it is unlikely that the same HW status register with the same bits exists on a chip of a different manufacturer.

The boot and debug state claims in this document are an example of a claim that has been defined in this neutral way.

E.3. Security Level Neutral

Many use cases will have EATs generated by some of the most secure hardware and software that exists. Secure Elements and smart cards are examples of this. However, EAT is intended for use in low-security use cases the same as high-security use case. For example, an app on a mobile device may generate EATs on its own.

Claims should be defined and registered on the basis of whether they are useful and interoperable, not based on security level. In particular, there should be no exclusion of claims because they are just used only in low-security environments.

E.4. Reuse of Extant Data Formats

Where possible, claims should use already standardized data items, identifiers and formats. This takes advantage of the expertise put into creating those formats and improves interoperability.

Often extant claims will not be defined in an encoding or serialization format used by EAT. It is preferred to define a CBOR and JSON encoding for them so that EAT implementations do not require a plethora of encoders and decoders for serialization formats.

In some cases, it may be better to use the encoding and serialization as is. For example, signed X.509 certificates and CRLs can be carried as-is in a byte string. This retains interoperability with the extensive infrastructure for creating and processing X.509 certificates and CRLs.

E.5. Proprietary Claims

It is not always possible or convenient to achieve the above goals, so the definition and use of proprietary claims is an option.

For example, a device manufacturer may generate a token with proprietary claims intended only for verification by a service offered by that device manufacturer. This is a supported use case.

In many cases proprietary claims will be the easiest and most obvious way to proceed, however for better interoperability, use of general standardized claims is preferred.

Appendix F. Endorsements and Verification Keys

The verifier must possess the correct key when it performs the cryptographic part of an EAT verification (e.g., verifying the COSE/JOSE signature). This section describes several ways to identify the verification key. There is not one standard method.

The verification key itself may be a public key, a symmetric key or something complicated in the case of a scheme like Direct Anonymous Attestation (DAA).

RATS Architecture [RFC9334] describes what is called an endorsement. This is an input to the verifier that is usually the basis of the trust placed in an EAT and the attester that generated it. It may contain the public key for verification of the signature on the EAT. It may contain implied claims, those that are passed on to the relying party in attestation results.

There is not yet any standard format(s) for an endorsement. One format that may be used for an endorsement is an X.509 certificate. Endorsement data like reference values and implied claims can be carried in X.509 v3 extensions. In this use, the public key in the X.509 certificate becomes the verification key, so identification of the endorsement is also identification of the verification key.

The verification key identification and establishment of trust in the EAT and the attester may also be by some other means than an endorsement.

For the components (attester, verifier, relying party,...) of a particular end-end attestation system to reliably interoperate, its definition should specify how the verification key is identified. Usually, this will be in the profile document for a particular attestation system.

See also security consideration in Section 9.6.

F.1. Identification Methods

Following is a list of possible methods of key identification. A specific attestation system may employ any one of these or one not listed here.

The following assumes endorsements are X.509 certificates or equivalent and thus does not mention or define any identifier for endorsements in other formats. If such an endorsement format is created, new identifiers for them will also need to be created.

F.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC9052] and [RFC7515]

COSE leaves the semantics of the key ID open-ended. It could be a record locator in a database, a hash of a public key, an input to a Key Derivation Function (KDF), an Authority Key Identifier (AKI) for an X.509 certificate or other. The profile document should specify what the key ID's semantics are.

F.1.2. JWS and COSE X.509 Header Parameters

COSE X.509 [COSE.X509.Draft] and JSON Web Signature [RFC7515] define several header parameters (x5t, x5u,...) for referencing or carrying X.509 certificates any of which may be used.

The X.509 certificate may be an endorsement and thus carrying additional input to the verifier. It may be just an X.509 certificate, not an endorsement. The same header parameters are used in both cases. It is up to the attestation system design and the verifier to determine which.

F.1.3. CBOR Certificate COSE Header Parameters

Compressed X.509 and CBOR Native certificates are defined by CBOR Certificates [CBOR.Cert.Draft]. These are semantically compatible with X.509 and therefore can be used as an equivalent to X.509 as described above.

These are identified by their own header parameters (c5t, c5u,...).

F.1.4. Claim-Based Key Identification

For some attestation systems, a claim may be re-used as a key identifier. For example, the UEID uniquely identifies the entity and therefore can work well as a key identifier or endorsement identifier.

This has the advantage that key identification requires no additional bytes in the EAT and makes the EAT smaller.

This has the disadvantage that the unverified EAT must be substantially decoded to obtain the identifier since the identifier is in the COSE/JOSE payload, not in the headers.

Appendix G. Changes from Previous Drafts

// RFC editor: please remove this paragraph.

The following is a list of known changes since the immediately previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences. A comprehensive history is available via the IETF Datatracker's record for this document.

G.1. From draft-ietf-rats-eat-24

* Use only CDDL definition names for "Claim Value Type" column in CWT claim registry

* Correct the "Claim Value Type" for some claims

* Make SUIT reference informative (it use is optional in an optional claim)

Contributors

Many thanks to the following contributors to draft versions of this document:

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

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

Miguel Ballesteros

Michael Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca

Patrick Uiterwijk

Mathias Brossard

Hannes Tschofenig
Arm Limited
Email: hannes.tschofenig@arm.com

Paul Crowley

Authors' Addresses

Laurence Lundblade
Security Theory LLC
Email: lgl@securitytheory.com

Giridhar Mandyam
Email: giridhar.mandyam@gmail.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough
GU14 7LS
United Kingdom
Phone: +44 1252 363189
Email: jodonogh@qti.qualcomm.com

Carl Wallace
Red Hound Software, Inc.
Email: carl@redhoundsoftware.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 13 March 2024

H. Birkholz
Fraunhofer SIT
E. Voit
Cisco
W. Pan
Huawei

10 September 2023

Attestation Event Stream Subscription
draft-ietf-rats-network-device-subscription-04

Abstract

This memo defines how to subscribe to YANG Event Streams for Remote Attestation Procedures (RATS). In RATS, Conceptional Messages, are defined. Analogously, the YANG module defined in this memo augments the YANG module for TPM-based Challenge-Response based Remote Attestation (CHARRA) to allow for subscription to remote attestation Evidence. Additionally, this memo provides the methods and means to define additional Event Streams for other Conceptual Message as illustrated in the RATS Architecture, e.g. Attestation Results, Endorsements, or Event Logs.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-rats-tpm-based-network-device-attest] and [I-D.ietf-rats-yang-tpm-charra] define the operational prerequisites and a YANG Model for the acquisition of Evidence and other Conceptional Messages from a TPM-based network device. However, there are limitations inherent in the challenge-response based remote attestation (CHARRA [I-D.ietf-rats-reference-interaction-models]) upon which these documents are based. One of these limitation is that it is a RATS role's duty to request Conceptional Messages, such as Evidence as provided by [I-D.ietf-rats-yang-tpm-charra], from another RATS entity. The result is that the interval between the occurrence of a security-relevant change event, and the event's visibility within the interested RATS entity, such as a Verifier or a Relying Party, can be unacceptably long. It is common to convey Conceptual Messages ad-hoc or periodically via requests. As new technologies emerge, some of these solutions require Conceptual Messages to be conveyed from one RATS entity to another without the need of continuous polling. Subscription to YANG Notifications [RFC8639] provides a set of standardized tools to facilitate these emerging requirements. This memo specifies a YANG augment to subscribe to YANG modeled remote attestation Evidence as defined in [I-D.ietf-rats-yang-tpm-charra]. Additionally, this memo provides the means to define further Event Streams to convey Conceptional Messages other than Evidence, such as Attestation Results, Endorsements, or Event Logs.

In essence, the limitation of poll-based interactions results in two adverse effects:

1. Conceptual Messages are not streamed to an interested consumer of information, e.g., Verifiers or Relying Parties, as soon as they are generated.

2. If they were to be streamed, Conceptual Messages are not appraisable for their freshness in every scenario. This becomes more important with Conceptional Messages that have a strong dependency on freshness, such as Evidence and corresponding Attestation Results.

This specification addresses the first adverse effect by enabling a consumer of Conceptual Messages (the subscriber) to request a continuous stream of new or updated Conceptual Messages via an [RFC8639] subscription to an <attestation> Event Stream. This new Event Stream is defined in this document and exists upon the producer of Conceptual Messages (the publisher). In the case of a Verifier's subscription to an Attester's Evidence, the Attester will continuously stream a requested set of freshly generated Evidence to the subscribing Verifier.

The second adverse effect results from the use of nonces in the challenge-response interaction model [I-D.ietf-rats-reference-interaction-models] realized in [I-D.ietf-rats-yang-tpm-charra]. In [I-D.ietf-rats-yang-tpm-charra], an Attester must wait for a new nonce from a Verifier before it generates a new TPM Quote. To address delays resulting from such a wait, this specification enables freshness to be asserted asynchronously via the streaming attestation interaction model [I-D.ietf-rats-reference-interaction-models]. To convey a RATS Conceptual Message, an initial nonce is provided during the subscription to an Event Stream.

There are several options to refresh a nonce provided by the initial subscription or its freshness characteristics. All of these methods are out-of-band of an established subscription to YANG Notifications. Two complementary methods are taken into account by this memo:

1. a central provider supplies new fresh nonces, e.g. via a Handle Provider that distributes Epoch IDs to all entities in a domain as described in [I-D.ietf-rats-architecture] and as facilitated by the Uni-Directional Remote Attestation described in [I-D.ietf-rats-reference-interaction-models] or

2. the freshness characteristics of a received nonce are updated by -- potentially periodic or ad-hoc -- out-of-band TPM Quote requests as facilitated by [I-D.ietf-rats-yang-tpm-charra].

Both approaches to update the freshness characteristics of the Conceptual Messages conveyed via subscription to YANG Notification that are taken into account by this memo assume that clock drift between involved entities can occur. In consequence, in some usage scenarios the timing considerations for freshness [I-D.ietf-rats-architecture] might have to be updated in some regular interval. Analogously, there are can be additional methods that are not describe by but nevertheless supported by this memo.

This memo enables to remove the two adverse effects described by using the YANG augment specified. The YANG augment supports, for example, a RATS Verifier to maintain a continuous appraisal procedure of verifiably fresh Attester Evidence without relying on continuous polling.

2. Terminology

The following terms are imported from [I-D.ietf-rats-architecture]: Attester, Conceptual Message, Evidence, Relying Party, and Verifier. Also imported are the time definitions time(VG), time(NS), time(EG), time(RG), and time(RA) from that document's Appendix A. The following terms are imported from [RFC8639]: Event Stream, Subscription, Event Stream Filter, Dynamic Subscription.

2.1. Requirements Notation

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

3. Operational Model

[I-D.ietf-rats-tpm-based-network-device-attest] describes the conveyance of TPM-based Evidence from a Verifier to an Attester using the CHARRA interaction model [I-D.ietf-rats-reference-interaction-models]. The operational model and corresponding sequence diagram described in this section is based on [I-D.ietf-rats-yang-tpm-charra]. The basis for interoperability required for additional types of Event Streams is covered in Section 6. The following sub-section focuses on subscription to YANG Notifications to the <attestation> Event Stream.

3.1. Sequence Diagram

Figure 1 below is a sequence diagram which updates Figure 5 of [I-D.ietf-rats-tpm-based-network-device-attest]. This sequence diagram replaces the [I-D.ietf-rats-tpm-based-network-device-attest] TPM-specific challenge-response interaction model with a [RFC8639] Dynamic Subscription to an <attestation> Event Stream. The contents of the <attestation> Event Stream are defined below within Section 4.

```
   .----------.                          .--------------------------.
   | Attester |                          | Relying Party / Verifier |
   '----------'                          '--------------------------'
      time(VG)                                                      |
   generateClaims(targetEnvironment)                                |
         | => claims, eventLogs                                     |
         |                                                          |
         |<---------establish-subscription(<attestation>)------time(NS)
         |                                                          |
      time(EG)                                                      |
   generateEvidence(nonce, PcrSelection, collectedClaims)           |
         | => SignedPcrEvidence(nonce, PcrSelection)                |
         | => LogEvidence(collectedClaims)                          |
         |                                                          |
         |--filter(<pcr-extend>)---------------------------------->|
         |--<tpm12-attestation> or <tpm20-attestation>------------>|
         |                                                          |
         |                                                   time(RG,RA)
         |     appraiseEvidence(SignedPcrEvidence, eventLog, refClaims)
         |                                     attestationResult <= |
         |                                                          |
         ~                                                          ~
      time(VG')                                                     |
   generateClaimes(targetEnvironment)                               |
         | => claims                                                |
         |                                                          |
      time(EG')                                                     |
   generateEvidence(handle, PcrSelection, collectedClaims)          |
         | => SignedPcrEvidence(nonce, PcrSelection)                |
         | => LogEvidence(collectedClaims)                          |
         |                                                          |
         |--filter(<pcr-extend>)---------------------------------->|
         |--<tpm12-attestation> or <tpm20-attestation>------------>|
         |                                                          |
         |                                                  time(RG',RA')
         |    appraiseEvidence(SignedPcrEvidence, eventLog, refClaims)
         |                                     attestationResult <= |
         |                                                          |
```

Figure 1: YANG Subscription Model for Remote Attestation

* time(VG,RG,RA) are identical to the corresponding time definitions from [I-D.ietf-rats-tpm-based-network-device-attest].

* time(VG',RG',RA') are subsequent instances of the corresponding times from Figure 5 in [I-D.ietf-rats-tpm-based-network-device-attest].

* time(NS) - the subscriber generates a nonce and makes an [RFC8639] <establish-subscription> request based on a nonce. This request also includes the augmentations defined in this document's YANG model. Key subscription RPC parameters include:

  - the nonce,

  - a set of PCRs of interest which the wants to appraise, and

  - an optional filter which can reduce the logged events on the <attestation> stream pushed to the Verifier.

* time(EG) - an initial response of Evidence is returned to the Verifier. This includes:

  - a replay of filtered log entries which have extended into a PCR of interest since boot are sent in the <pcr-extend> notification, and

  - a signed TPM quote that contains at least the PCRs from the <establish-subscription> RPC are included in a <tpm12-attestation> or <tpm20-attestation>). This quote must have included the nonce provided at time(NS).

* time(VG',EG') - this occurs when a PCR is extended subsequent to time(EG). Immediately after the extension, the following information needs to be pushed to the Verifier:

  - any values extended into a PCR of interest,

  - a signed TPM Quote showing the result the PCR extension, and

  - and a handle (see Section 6. in [I-D.ietf-rats-reference-interaction-models], which is either the initially received nonce or a more recently received Epoch ID (see Section 10.3. in [I-D.ietf-rats-architecture] that contains a new nonce or equivalent qualified data.

One way to acquire a new time synchronisation that allows for the reuse of the initially received nonce as a fresh handle is elaborated on in the follow section Section 3.2.

3.2. Continuously Verifying Freshness

As there is no new Verifier nonce provided at time(EG'), it is important to validate the freshness of TPM Quotes which are delivered at that time. The method of doing this verification will vary based on the capabilities of the TPM cryptoprocessor used.

3.2.1. TPM 1.2 Quote

The [RFC8639] notification format includes the <eventTime> object. This can be used to determine the amount of time subsequent to the initial subscription each notification was sent. However this time is not part of the signed results which are returned from the Quote, and therefore is not trustworthy as objects returned in the Quote. Therefore a Verifier MUST periodically issue a new nonce, and receive this nonce within a TPM quote response in order to ensure the freshness of the results. This can be done using the <tpm12-challenge-response-attestation> RPC from [I-D.ietf-rats-yang-tpm-charra].

3.2.2. TPM 2 Quote

When the Attester includes a TPM2 compliant cryptoprocessor, internal time-related counters are included within the signed TPM Quote. By including a initial nonce in the [RFC8639] subscription request, fresh values for these counters are pushed as part of the first TPM Quote returned to the Verifier. And then as shown by [I-D.birkholz-rats-tuda], subsequent TPM Quotes delivered to the Verifier can the be appraised for freshness based on the predictable incrementing of these time-related countersr.

The relevant internal time-related counters defined within [TPM2.0] can be seen within <tpms-clock-info>. These counters include the <clock>, <reset-counter>, and <restart-counter> objects. The rules for appraising these objects are as follows:

* If the <clock> has incremented for no more than the same duration as both the <eventTime> and the Verifier's internal time since the initial time(EG) and any previous time(EG'), then the TPM Quote may be considered fresh. Note that [TPM2.0] allows for +/- 15% clock drift. However many chips significantly improve on this maximum drift. If available, chip specific maximum drifts SHOULD be considered during the appraisal process.

* If the <reset-counter>, <restart-counter> has incremented. The existing subscription MUST be terminated, and a new <establish-subscription> SHOULD be generated.

* If a TPM Quote on any subscribed PCR has not been pushed to the Verifier for a duration of an Attester defined heartbeat interval, then a new TPM Quote notification should be sent to the Verifier. This may often be the case, as certain PCRs might be infrequently updated.

```
.----------.                        .--------------------------.
| Attester |                        | Relying Party / Verifier |
'----------'                        '--------------------------'
   time(VG',EG')                                               |
     |-<tpm20-attestation>------------------------------>|
     |                                   :                |
     ~                            Heartbeat interval      ~
     |                                   :                |
   time(EG')                             :                |
     |-<tpm20-attestation>------------------------------>|
     |                                                    |
```

4. Remote Attestation Event Stream

The <attestation> Event Stream is an [RFC8639] compliant Event Stream which is defined within this section and within the YANG Module of [I-D.ietf-rats-yang-tpm-charra]. This Event Stream contains YANG notifications which carry Evidence to assists a Verifier in appraising the Trustworthiness Level of an Attester. Data Nodes within Section 4.6 allow the configuration of this Event Stream's contents on an Attester.

This <attestation> Event Stream may only be exposed on Attesters supporting [I-D.ietf-rats-tpm-based-network-device-attest]. As with [I-D.ietf-rats-tpm-based-network-device-attest], it is up to the Verifier to understand which types of cryptoprocessors and keys are acceptable.

4.1. Subscription to the <attestation> Event Stream

To establish a subscription to an Attester in a way which provides provably fresh Evidence, initial randomness must be provided to the Attester. This is done via the augmentation of a <nonce-value> into [RFC8639] the <establish-subscription> RPC. Additionally, a Verifier must ask for PCRs of interest from a platform.

```
augment /sn:establish-subscription/sn:input:
  +---w nonce-value    binary
  +---w pcr-index*     tpm:pcr
```

The result of the subscription will be that passing of the following information:

1. <tpm12-attestation> and <tpm20-attestation> notifications which include the provided <nonce-value>. These attestation notifications MUST at least include all the <pcr-indicies> requested in the RPC.

2. a series of <pcr-extend> notifications which reference the requested PCRs on all TPM based cryptoprocessors on the Attester.

3. <tpm12-attestation> and <tpm20-attestation> notifications generated within a few seconds of the <pcr-extend> notifications. These attestation notifications MUST at least include any PCRs extended.

If the Verifier does not want to see the logged extend operations for all PCRs available from an Attester, an Event Stream Filter should be applied. This filter will remove Evidence from any PCRs which are not interesting to the Verifier.

4.2. Replaying a history of previous TPM extend operations

Unless it is relying on Known Good Values, a Verifier will need to acquire a history of PCR extensions since the Attester has been booted. This history may be requested from the Attester as part of the <establish-subscription> RPC. This request is accomplished by placing a very old <replay-start-time> within the original RPC request. As the very old <replay-start-time> will pre-date the time of Attester boot, a <replay-start-time-revision> will be returned in the <establish-subscription> RPC response, indicating when the Attester booted. Immediately following the response (and before the notifications above) one or more <pcr-extend> notifications which document all extend operations which have occurred for the requested PCRs since boot will be sent. Many extend operations to a single PCR index on a single TPM SHOULD be included within a single notification.

Note that if a Verifier has a partial history of extensions, the <replay-start-time> can be adjusted so that known extensions are not forwarded.

The end of this history replay will be indicated with the [RFC8639] <replay-completed> notification. For more on this sequence, see Section 2.4.2.1 of [RFC8639].

After the <replay-complete> notification is provided, a TPM Quote will be requested and the result passed to the Verifier via a <tpm12-attestation> and <tpm20-attestation> notification. If there have been any additional extend operations which have changed a subscribed PCR value in this quote, these MUST be pushed to the Verifier before the <tpm12-attestation> and <tpm20-attestation> notification.

At this point the Verifier has sufficient Evidence appraise the reported extend operations for each PCR, as well compare the expected value of the PCR value against that signed by the TPM.

4.2.1. TPM2 Heartbeat

For TPM2, make sure that every requested PCR is sent within an <tpm20-attestation> no less frequently than once per heartbeat interval. This MAY be done with a single <tpm20-attestation> notification that includes all requested PCRs every heartbeat interval. This MAY be done with several <tpm20-attestation> notifications at different times during that heartbeat interval.

4.3. YANG notifications placed on the <attestation> Event Stream

4.3.1. pcr-extend

This notification documents when a subscribed PCR is extended within a single TPM cryptoprocessor. It SHOULD be emmitted no less than the <marshalling-period> after an the PCR is first extended. (The reason for the marshalling is that it is quite possible that multiple extensions to the same PCR have been made in quick succession, and these should be reflected in the same notification.) This notification MUST be emmitted prior to a <tpm12-attestation> or <tpm20-attestation> notification which has included and signed the results of any specific PCR extension. If pcr extending events occur during the generation of the <tpm12-attestation> or <tpm20-attestation> notification, the marshalling period MUST be extended so that a new <pcr-extend> is not sent until the corresponding notifications have been sent.

```
+---n pcr-extend
   +--ro certificate-name     certificate-name-ref
   +--ro pcr-index-changed*   tpm:pcr
   +--ro attested-event* []
      +--ro attested-event
         +--ro extended-with             binary
         +--ro (event-details)?
            +--:(bios-event-log)
            |  +--ro bios-event-entry* [event-number]
            |     +--ro event-number    uint32
            |     +--ro event-type?     uint32
            |     +--ro pcr-index?      pcr
            |     +--ro digest-list* []
            |     |  +--ro hash-algo?   identityref
            |     |  +--ro digest*      binary
            |     +--ro event-size?     uint32
            |     +--ro event-data*     uint8
            +--:(ima-event-log)
            |  +--ro ima-event-entry* [event-number]
            |     +--ro event-number               uint64
            |     +--ro ima-template?              string
            |     +--ro filename-hint?             string
            |     +--ro filedata-hash?             binary
            |     +--ro filedata-hash-algorithm?   string
            |     +--ro template-hash-algorithm?   string
            |     +--ro template-hash?             binary
            |     +--ro pcr-index?                 pcr
            |     +--ro signature?                 binary
            +--:(netequip-boot-event-log)
               +--ro boot-event-entry* [event-number]
                  +--ro event-number               uint64
                  +--ro filename-hint?             string
                  +--ro filedata-hash?             binary
                  +--ro filedata-hash-algorithm?   string
                  +--ro file-version?              string
                  +--ro file-type?                 string
                  +--ro pcr-index?                 pcr
```

Each <pcr-extend> MUST include one or more values being extended into the PCR. These are passed within the <extended-with> object. For each extension, details of the event SHOULD be provided within the <event-details> object. The format of any included <event-details> is identified by the <event-type>. This document includes two YANG structures which may be inserted into the <event-details>. These two structures are: <ima-event-log> and <bios-event-log>. Implementations wanting to provide additional documentation of a type of PCR extension may choose to define additional YANG structures which can be placed into <event-details>.

4.3.2. tpm12-attestation

This notification contains an instance of a TPM1.2 style signed cryptoprocessor measurement. It is supplemented by Attester information which is not signed. This notification is generated and emitted from an Attester when at least one PCR identified within the subscribed <pcr-indices> has changed from the previous <tpm12-attestation> notification. This notification MUST NOT include the results of any PCR extensions not previously reported by a <pcr-extend>. This notification SHOULD be emitted as soon as a TPM Quote can extract the latest PCR hashed values. This notification MUST be emitted prior to a subsequent <pcr-extend>.

```
+---n tpm12-attestation {taa:TPM12}?
   +--ro certificate-name       tpm:certificate-name-ref
   +--ro up-time?               uint32
   +--ro TPM_QUOTE2?            binary
   +--ro TPM12-hash-algo?       identityref
   +--ro unsigned-pcr-values* []
      +--ro pcr-index*   tpm:pcr
      +--ro pcr-value*   binary
```

All YANG objects above are defined within [I-D.ietf-rats-yang-tpm-charra]. The <tpm12-attestation> is not replayable.

4.3.3. tpm20-attestation

This notification contains an instance of TPM2 style signed cryptoprocessor measurements. It is supplemented by Attester information which is not signed. This notification is generated at two points in time:

* every time at least one PCR has changed from a previous tpm20-attestation. In this case, the notification SHOULD be emitted within 10 seconds of the corresponding <pcr-extend> being sent:

* after a locally configurable minimum heartbeat period since a previous tpm20-attestation was sent.

```
+---n tpm20-attestation {taa:TPM20}?
   +--ro certificate-name       tpm:certificate-name-ref
   +--ro TPMS_QUOTE_INFO        binary
   +--ro quote-signature?       binary
   +--ro up-time?               uint32
   +--ro unsigned-pcr-values* []
      +--ro TPM20-hash-algo?   identityref
      +--ro pcr-values* [pcr-index]
         +--ro pcr-index    pcr
         +--ro pcr-value?   binary
```

All YANG objects above are defined within [I-D.ietf-rats-yang-tpm-charra]. The <tpm20-attestation> is not replayable.

4.4. Filtering Evidence at the Attester

It can be useful _not_ to receive all Evidence related to a PCR. An example of this is would be a when a Verifier maintains known good values of a PCR. In this case, it is not necessary to send each extend operation.

To accomplish this reduction, when an RFC8639 <establish-subscription> RPC is sent, a <stream-filter> as per RFC8639, Section 2.2 can be set to discard a <pcr-extend> notification when the <pcr-index-changed> is uninteresting to the verifier.

4.5. Replaying previous PCR Extend events

To verify the value of a PCR, a Verifier must either know that the value is a known good value [KGV] or be able to reconstruct the hash value by viewing all the PCR-Extends since the Attester rebooted. Wherever a hash reconstruction might be needed, the <attestation> Event Stream MUST support the RFC8639 <replay> feature. Through the <replay> feature, it is possible for a Verifier to retrieve and sequentially hash all of the PCR extending events since an Attester booted. And thus, the Verifier has access to all the evidence needed to verify a PCR’s current value.

4.6. Configuring the <attestation> Event Stream

Figure 2 is tree diagram which exposes the operator configurable elements of the <attestation> Event Stream. This allows an Attester to select what information should be available on the stream. A fetch operation also allows an external device such as a Verifier to understand the current configuration of stream.

Almost all YANG objects below are defined via reference from [I-D.ietf-rats-yang-tpm-charra]. There is one object which is new with this model however. <tpm2-heartbeat> defines the maximum amount of time which should pass before a subscriber to the Event Stream should get a <tpm20-attestation> notification from devices which contain a TPM2.

```
  augment /tpm:rats-support-structures:
    +--rw marshalling-period?                  uint8
    +--rw tpm12-subscribed-signature-scheme?
    |   -> ../tpm:attester-supported-algos/tpm12-asymmetric-signing
    |      {taa:tpm12}?
    +--rw tpm20-subscribed-signature-scheme?
    |   -> ../tpm:attester-supported-algos/tpm20-asymmetric-signing
    |      {taa:tpm20}?
    +--rw tpm20-subscription-heartbeat?        uint16
  augment /tpm:rats-support-structures/tpm:tpms:
    +--rw subscription-aik?        tpm:certificate-name-ref
    +--rw (subscribable)?
       +--:(tpm12-stream) {taa:tpm12}?
       |  +--rw TPM12-hash-algo?   identityref
       |  +--rw tpm12-pcr-index*   tpm:pcr
       +--:(tpm20-stream) {taa:tpm20}?
          +--rw TPM20-hash-algo?   identityref
          +--rw tpm20-pcr-index*   tpm:pcr
```

Figure 2: Configuring the \<attestation\> Event Stream

5. YANG Module

This YANG module imports modules from [I-D.ietf-rats-yang-tpm-charra] and [RFC8639]. It is also work-in-progress.

```
<CODE BEGINS> ietf-rats-attestation-stream@2023-09-10.yang
module ietf-tpm-remote-attestation-stream {
  yang-version 1.1;
  namespace
     "urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation-stream";
  prefix tras;

  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-tpm-remote-attestation {
    prefix tpm;
    reference
```

```
         "draft-ietf-rats-yang-tpm-charra";
     }
     import ietf-tcg-algs {
       prefix taa;
     }

     organization "IETF";
     contact
       "WG Web:   <http://tools.ietf.org/wg/rats/>
        WG List:  <mailto:rats@ietf.org>

        Editor:   Eric Voit
                  <mailto:evoit@cisco.com>";

     description
       "This module contains conceptual YANG specifications for
        subscribing to attestation streams being generated from TPM chips.

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

        Redistribution and use in source and binary forms, with
        or without modification, is permitted pursuant to, and
        subject to the license terms contained in, the Simplified
        BSD License set forth in Section 4.c of the IETF Trust's
        Legal Provisions Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
        itself for full legal notices.";

     revision 2023-09-10 {
       description
         "Initial version.";
       reference
         "draft-birkholz-rats-network-device-subscription";
     }

     /*
      * IDENTITIES
      */

     identity pcr-unsubscribable {
       base sn:establish-subscription-error;
       description
         "Requested PCR is subscribable by the Attester.";
     }
```

```
/*
 * Groupings
 */

grouping heartbeat {
  description
    "Allows an Attester to push verifiable, current TPM PCR values
    even when there have been no recent changes to PCRs.";
  leaf tpm20-subscription-heartbeat {
    type uint16;
    description
      "Number of seconds before the Attestation stream should send a
      new notification with a fresh quote.  This allows confirmation
      that the PCR values haven't changed since the last
      tpm20-attestation.";
  }
}


/*
 * RPCs
 */

augment "/sn:establish-subscription/sn:input" {
  when 'derived-from-or-self(sn:stream, "attestation")';
  description
    "This augmentation adds a nonce to as a subscription parameters
     that apply specifically to datastore updates to RPC input.";
  uses tpm:nonce;
  leaf-list pcr-index {
    type tpm:pcr;
    min-elements 1;
    description
      "The numbers/indexes of the PCRs. This will act as a filter for
      the subscription so that 'tpm-extend' notifications related to
      non-requested PCRs will not be sent to a subscriber.";
  }
}

/*
 * NOTIFICATIONS
 */

notification pcr-extend {
  description
    "This notification indicates that one or more PCRs have been
    extended within a TPM based cryptoprocessor.  In less than the
    'marshalling-period', it MUST be followed with either a
```

```
        corresponding tpm12-attestation or tpm20-attestation notification
        which exposes the result of the PCRs updated.";
      uses tpm:certificate-name-ref;
      leaf-list pcr-index-changed {
        type tpm:pcr;
        min-elements 1;
        description
          "The number of each PCR extended.  This list MUST contain the
          set of PCRs descibed within the event log details.  This leaf
          can be derived from the list of attested events, but exposing
          it here allows for easy filtering of the notifications of
          interest to a verifier.";
      }
      list attested-event {
        description
          "A set of events which extended an Attester PCR.  The sequence
          of elements represented in list must match the sequence of
          events placed into the TPM's PCR.";
        container attested-event {
          description
            "An instance of an event which extended an Attester PCR";
          leaf extended-with {
            type binary;
            mandatory true;
            description
              "Information extending the PCR.";
          }
          choice event-details {
            description
              "Contains the event happened the Attester thought
              was worthy of recording in a PCR.

              choices are of types defined by the identityref
              base tpm:attested_event_log_type";
            case bios-event-log {
              if-feature "tpm:bios";
              description
                "BIOS/UEFI event log format";
              uses tpm:bios-event-log;
            }
            case ima-event-log {
              if-feature "tpm:ima";
              description
                "IMA event log format";
              uses tpm:ima-event-log;
            }
            case netequip-boot-event-log {
              if-feature "tpm:netequip_boot";
```

```
              description
                "IMA event log format";
              uses tpm:network-equipment-boot-event-log;
            }
          }
        }
      }
    }

    notification tpm12-attestation {
      if-feature "taa:tpm12";
      description
        "Contains an instance of TPM1.2 style signed cryptoprocessor
        measurements.  It is supplemented by unsigned Attester
        information.";
      leaf certificate-name {
        type tpm:certificate-name-ref;
        mandatory true;
        description
          "Allows a TPM quote to be associated with a certificate.";
      }
      uses tpm:tpm12-attestation;
      uses tpm:tpm12-hash-algo;
      list unsigned-pcr-values {
        description
          "Allows notifications to be filtered by PCR number or
          PCR value based on via YANG related mechanisms such as PATH.
          This is done without requiring the filtering structure to be
          applied against TCG structured data.";
        leaf-list pcr-index {
          type tpm:pcr;
          min-elements 1;
          description
            "PCR index number.";
        }
        leaf-list pcr-value {
          type binary;
          description
            "PCR value in a sequence which matches to the 'pcr-index'.";
        }
      }
    }

    notification tpm20-attestation {
      if-feature "taa:tpm20";
      description
        "Contains an instance of TPM2 style signed cryptoprocessor
        measurements.  It is supplemented by unsigned Attester
```

```
        information.";
      leaf certificate-name {
        type tpm:certificate-name-ref;
        mandatory true;
        description
          "Allows a TPM quote to be associated with a certificate.";
      }
      uses tpm:tpm20-attestation {
        description
          "Provides the attestation info.  Also ensures PCRs can be XPATH
          filtered by refining the unsigned data so that it appears.";
        refine unsigned-pcr-values {
          min-elements 1;
        }
        refine unsigned-pcr-values/pcr-values {
          min-elements 1;
        }
      }
    }


    /*
     * DATA NODES
     */

    augment "/tpm:rats-support-structures" {
      description
        "Defines platform wide 'attestation' stream subscription
        parameters.";
      leaf marshalling-period {
        type uint8;
        default 5;
        description
          "The maximum number of seconds between the time an event
          extends a PCR, and the 'tpm-extend' notification which reports
          it to a subscribed Verifier.  This period allows multiple
          extend operations bundled together and handled as a group.";
      }
      leaf tpm12-subscribed-signature-scheme {
        if-feature "taa:tpm12";
        type leafref {
          path "../tpm:attester-supported-algos" +
                 "/tpm:tpm12-asymmetric-signing";
        }
        description
          "A single signature-scheme which will be used to sign the
          evidence from a TPM 1.2. which is then placed onto the
          'attestation' event stream.";
```

```
      }
      leaf tpm20-subscribed-signature-scheme {
        if-feature "taa:tpm20";
        type leafref {
          path "../tpm:attester-supported-algos" +
                 "/tpm:tpm20-asymmetric-signing";
        }
        description
          "A single signature-scheme which will be used to sign the
          evidence from a TPM 2.0. which is then placed onto the
          'attestation' event stream.";
      }
      uses heartbeat{
        if-feature "taa:tpm20";
      }
    }

    augment "/tpm:rats-support-structures/tpm:tpms" {
      description
        "Allows the configuration 'attestation' stream parameters for a
        TPM.";
      leaf subscription-aik {
        type tpm:certificate-name-ref;
        description
          "Identifies the certificate-name associated with the
          notifications in the 'attestation' stream.";
      }
      choice subscribable {
        config true;
        description
          "Indicates that the set of notifications which comprise the
          'attestation' event stream can be modified or tuned by a
          network administrator.";
        case tpm12-stream {
          if-feature "taa:tpm12";
          description
            "Configuration elements for a TPM1.2 event stream.";
          uses tpm:tpm12-hash-algo;
          leaf-list tpm12-pcr-index {
            type tpm:pcr;
            description
              "The numbers/indexes of the PCRs which can be subscribed.";
          }
        }
        case tpm20-stream {
          if-feature "taa:tpm20";
          description
            "Configuration elements for a TPM2.0 event stream.";
```

```
          uses tpm:tpm20-hash-algo;
          leaf-list tpm20-pcr-index {
            type tpm:pcr;
            description
              "The numbers/indexes of the PCRs which can be subscribed.";
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Event Streams for Conceptual Messages

Analogous to the [RFC8639] compliant <attestation> Event Stream for the conveyance of remote attestation Evidence as defined in Section Section 4, additional Event Streams can be defined for this YANG augment. Additional Event Streams require separate YANG augment specifications that provide the Event Stream definition and optionally a content format definition either via subscriptions to YANG datastores or dedicated YANG Notifications. It is possible to use either YANG subscription methods to other YANG modules for RATS Conceptual Messages or to define Event Streams for other none-YANG-modeled data. In the context of RATS Conceptual Messages, both options MUST be a specified via YANG augments to this specification.

7. Security Considerations

To be written.

8. IANA Considerations

To be written.

9. References

9.1. Normative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-22, 28 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-22>.

[I-D.ietf-rats-reference-interaction-models]
Birkholz, H., Eckel, M., Pan, W., and E. Voit, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-ietf-rats-reference-interaction-models-08, 10 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-reference-interaction-models-08>.

[I-D.ietf-rats-tpm-based-network-device-attest]
Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", Work in Progress, Internet-Draft, draft-ietf-rats-tpm-based-network-device-attest-14, 22 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-tpm-based-network-device-attest-14>.

[I-D.ietf-rats-yang-tpm-charra]
Birkholz, H., Eckel, M., Bhandari, S., Voit, E., Sulzen, B., Xia, L., Laffey, T., and G. Fedorkow, "A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs", Work in Progress, Internet-Draft, draft-ietf-rats-yang-tpm-charra-21, 18 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-yang-tpm-charra-21>.

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

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

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://doi.org/10.17487/RFC8639>.

[TPM2.0] TCG, "TPM 2.0 Library Specification", n.d., <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

9.2. Informative References

[I-D.birkholz-rats-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in

Progress, Internet-Draft, draft-birkholz-rats-tuda-07, 10 July 2022, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-tuda-07>.

[KGV] TCG, "KGV", October 2003, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-NetEq-Attestation-Workflow-Outline_v1r9b_pubrev.pdf>.

Appendix A. Change Log

v01-v02

* Match YANG changes/simplifications made to charra

v00-v01

* rename notification: pcr-extended, which supports multiple PCRs

* netequip boot added

* YANG structure extension removed

* Matched to structural changes made within charra

Acknowledgements

Thanks to ...

Authors' Addresses

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

Eric Voit
Cisco Systems, Inc.
Email: evoit@cisco.com

Wei Pan
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu
210012
China
Email: william.panwei@huawei.com

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 5 September 2024

H. Birkholz
M. Eckel
Fraunhofer SIT
W. Pan
Huawei Technologies
E. Voit
Cisco

4 March 2024

Reference Interaction Models for Remote Attestation Procedures
draft-ietf-rats-reference-interaction-models-09

Abstract

This document describes interaction models for remote attestation procedures (RATS). Three conveying mechanisms -- Challenge/Response, Uni-Directional, and Streaming Remote Attestation -- are illustrated and defined. Analogously, a general overview about the information elements typically used by corresponding conveyance protocols are highlighted.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Remote ATtestation procedureS (RATS, [RFC9334]) are workflows composed of roles and interactions, in which Verifiers create Attestation Results about the trustworthiness of an Attester's system component characteristics. The Verifier's assessment in the form of Attestation Results is created based on Attestation Policies and Evidence -- trustable and tamper-evident Claims Sets about an Attester's system component characteristics -- generated by an Attester. The roles _Attester_ and _Verifier_, as well as the Conceptual Messages _Evidence_ and _Attestation Results_ are concepts defined by the RATS Architecture [RFC9334]. This document defines interaction models that can be used in specific RATS-related solution documents. The primary focus of this document is the conveyance of attestation Evidence. The reference models defined can also be applied to the conveyance of other Conceptual Messages in RATS. Specific goals of this document are to:

1. prevent inconsistencies in descriptions of interaction models in other documents (due to text cloning and evolution over time), and to

2. enable to highlight an exact delta/divergence between the core set of characteristics captured here in this document and variants of these interaction models used in other specifications or solutions.

In summary, this document enables the specification and design of trustworthy and privacy preserving conveyance methods for attestation Evidence from an Attester to a Verifier. While the conveyance of other Conceptual Messages is out-of-scope the methods described can also be applied to the conveyance of, for example, Endorsements or Attestation Results.

2. Terminology

This document uses the following set of terms, roles, and concepts as defined in [RFC9334]: Attester, Verifier, Relying Party, Conceptual Message, Evidence, Endorsement, Attestation Result, Appraisal Policy, Attesting Environment, Target Environment

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.

2.1. Disambiguation

The term "Remote Attestation" is a common expression and often associated or connoted with certain properties. The term "Remote" in this context does not necessarily refer to a remote entity in the scope of network topologies or the Internet. It rather refers to decoupled systems or entities that exchange the payload of the Conceptual Message type called Evidence [RFC9334]. This conveyance can also be "Local", if the Verifier role is part of the same entity as the Attester role, e.g., separate system components of the same Composite Device (a single RATS entity). Even if an entity takes on two or more different roles, the functions they provide typically reside in isolated environments that are components of the same entity. Examples of such isolated environments include: a Trusted Execution Environment (TEE), Baseboard Management Controllers (BMCs), as well as other physical or logical protected/isolated/shielded Computing Environments (e.g. embedded Secure Elements (eSE) or Trusted Platform Modules (TPM)). Readers of this document should be familiar with the concept of Layered Attestation as described in Section 3.1 Two Types of Environments of an Attester in [RFC9334] and the definition of Attestation as described in [I-D.ietf-rats-tpm-based-network-device-attest].

3. Scope and Intent

This document focuses on generic interaction models between Attesters and Verifiers in order to convey Evidence. Complementary procedures, functions, or services that are required for a complete semantic binding of the concepts defined in [RFC9334] are out-of-scope of this document. Examples include: identity establishment, key distribution and enrollment, time synchronization, as well as certificate revocation.

Furthermore, any processes and duties that go beyond carrying out remote attestation procedures are out-of-scope.

For instance, using the results of a remote attestation procedure that are created by the Verifier, e.g., how to triggering remediation actions or recovery processes, as well as such remediation actions and recovery processes themselves, are also out-of-scope.

The interaction models illustrated in this document are intended to provide a stable basis and reference for other solutions documents inside or outside the IETF. Solution documents of any kind can reference the interaction models in order to avoid text clones and to avoid the danger of subtle discrepancies. Analogously, deviations from the generic model descriptions in this document can be illustrated in solutions documents to highlight distinct contributions.

4. Essential Requirements

In order to ensure appropriate conveyance of Evidence, there exist essential requirements which MUST be fulfilled:

Integrity: Information provided by an Attester MUST be integral. This may be achieved by means of a digital signature over Attestation Evidence. The signature may be symmetric, such as an HMAC, or asymmetric, such as ECDSA.

Authentication: The information provided by the Attester MUST be authentic. For that purpose, the Attester should authenticate itself to the Verifier. This may be an implicit authentication by means of a digital signature over the Attestation Evidence, which does not require additional protocol steps, or may be achieved by using a confidential channel by means of encryption.

4.1. Endorsement of Attesting Environments

Via its Attesting Environments, an Attester only generates Evidence about its Target Environments. After being appraised to be trustworthy, a Target Environment may become a new Attesting Environment in charge of generating Evidence for further Target Environments. [RFC9334] explains this as Layered Attestation. Layered Attestation has to start with an initial Attesting Environment. In essence, there cannot be turtles all the way down [turtles]. At this rock bottom of Layered Attestation, the Attesting Environments are always called Roots of Trust (RoT). An Attester cannot generate Evidence about its own RoTs by design. As a consequence, a Verifier requires trustable statements about this subset of Attesting Environments from a different source than the Attester itself. The corresponding trustable statements are called Endorsements and originate from external, trustable entities that take on the role of an Endorser (e.g., supply chain entities).

5. Normative Prerequisites

In order to ensure an appropriate conveyance of Evidence via interaction models in general, the following set of prerequisites MUST be in place to support the implementation of interaction models:

Authentication Secret: An Authentication Secret MUST be available exclusively to an Attesting Environment of an Attester.

The Attester MUST protect Claims with that Authentication Secret, thereby proving the authenticity of the Claims included in Evidence. The Authentication Secret MUST be established before RATS can take place.

Attester Identity: A statement about a distinguishable Attester made by an Endorser.

The provenance of Evidence with respect to a distinguishable Attesting Environment MUST be correct and unambiguous.

An Attester Identity MAY be an Authentication Secret which is available exclusively to one of the Attesting Environments of an Attester. It MAY be a unique identity, MAY be included in a zero-knowledge proof (ZKP), MAY be part of a group signature, or it MAY be a randomized DAA credential [DAA].

Attestation Evidence Authenticity: Attestation Evidence MUST be authentic.

In order to provide proofs of authenticity, Attestation Evidence SHOULD be cryptographically associated with an identity document (e.g., a PKIX certificate or trusted key material, or a randomized DAA credential [DAA]), or SHOULD include a correct, unambiguous and stable reference to an accessible identity document.

Evidence Freshness: Evidence MUST include an indicator about its freshness that can be understood by a Verifier. Analogously, interaction models MUST support the conveyance of proofs of freshness in a way that is useful to Verifiers and their appraisal procedures.

Evidence Protection: Evidence MUST be a set of well-formatted and well-protected Claims that an Attester can create and convey to a Verifier in a tamper-evident manner.

6. Generic Information Elements

This section defines the information elements that are vital to all kinds interaction models. Varying from solution to solution, generic information elements can be either included in the scope of protocol messages (instantiating Conceptual Messages) or can be included in additional protocol parameters or payload. Ultimately, the following information elements are required by any kind of scalable remote attestation procedure using one or more of the interaction models provided.

Authentication Secret IDs ('authSecIDs'): _mandatory_

A statement representing an identifier list that MUST be associated with corresponding Authentication Secrets used to protect Claims included in Evidence.

Each distinguishable Attesting Environment has access to a protected capability that provides an Authentication Secret associated with that Attesting Environment. Consequently, an Authentication Secret ID can also identify an Attesting Environment.

Handle ('handle'): _mandatory_

A statement that is intended to uniquely distinguish received Evidence and/or determine the freshness of Evidence.

A Verifier can also use a Handle as an indicator for authenticity or attestation provenance, as only Attesters and Verifiers that are intended to exchange Evidence should have knowledge of the corresponding Handles. Examples include Nonces or signed timestamps.

Claims ('claims'): _mandatory_

Claims are assertions that represent characteristics of an Attester's Target Environment.

Claims are part of a Conceptual Message and are, for example, used to appraise the integrity of Attesters via Verifiers. The other information elements in this section can be expressed as Claims in any type of Conceptional Messages.

Event Logs ('eventLogs'): _optional_

Event Logs accompany Claims by providing event trails of security-

critical events in a system. The primary purpose of Event Logs is to support Claim reproducibility by providing information on how Claims originated.

Reference Values ('refValues') _mandatory_

Reference Values as defined in [RFC9334]. This specific type of Claims is used to appraise Claims incorporated in Evidence. For example, Reference Values MAY be Reference Integrity Measurements (RIM) or assertions that are implicitly trusted because they are signed by a trusted authority (see Endorsements in [RFC9334]). Reference Values typically represent (trusted) Claim sets about an Attester's intended platform operational state.

Claim Selection ('claimSelection'): _optional_

A (sub-)set of Claims which can be created by an Attester.

Claim Selections act as filters to specify the exact set of Claims to be included in Evidence. In a remote attestation process, a Verifier sends a Claim Selection, among other elements, to an Attester. An Attester MAY decide whether or not to provide all requested Claims from a Claim Selection to the Verifier.

Collected Claims ('collectedClaims'): _mandatory_

Collected Claims represent a (sub-)set of Claims created by an Attester.

Collected Claims are gathered based on the Claims selected in the Claim Selection. If a Verifier does not provide a Claim Selection, then all available Claims on the Attester are part of the Collected Claims.

Evidence ('evidence'): _mandatory_

A set of Claims that consists of a list of Authentication Secret IDs that each identifies an Authentication Secret in a single Attesting Environment, the Attester Identity, Claims, and a Handle. Attestation Evidence MUST cryptographically bind all of these information elements. Evidence MUST be protected via an Authentication Secret. The Authentication Secret MUST be trusted by the Verifier as authoritative.

Attestation Result ('attestationResult'): _mandatory_

An Attestation Result is produced by the Verifier as the output of

the appraisal of Evidence. Attestation Results include condensed assertions about integrity or other characteristics of the corresponding Attester that are processible by Relying Parties.

7. Interaction Models

The following subsections introduce and illustrate the interaction models:

1. Challenge/Response Remote Attestation

2. Uni-Directional Remote Attestation

3. Streaming Remote Attestation

Each section starts with a sequence diagram illustrating the interactions between Attester and Verifier. While the presented interaction models focus on the conveyance of Evidence, the intention of this document is in support of future work that applies the presented models to the conveyance of other Conceptual Messages, namely Attestation Results, Endorsements, Reference Values, or Appraisal Policies.

All interaction models have a strong focus on the use of a handle to incorporate a type of proof of freshness and to prevent replay attacks. The way these handles are processed is the most prominent difference between the three interaction models.

7.1. Challenge/Response Remote Attestation

```
.----------.                                                .----------.
| Attester |                                                | Verifier |
'----+-----'                                                '-----+----'
     |                                                            |
=================[Evidence Generation and Conveyance]===================
     |                                                            |
  generateClaims(attestingEnvironment)                            |
     | => claims, eventLogs                                       |
     |                                                            |
     |<--- requestAttestation(handle, authSecIDs, claimSelection) |
     |                                                            |
  collectClaims(claims, claimSelection)                           |
     | => collectedClaims                                         |
     |                                                            |
  generateEvidence(handle, authSecIDs, collectedClaims)           |
     | => evidence                                                |
     |                                                            |
     | evidence, eventLogs ------------------------------------->|
     |                                                            |
==========================[Evidence Appraisal]==========================
     |                                                            |
     |                appraiseEvidence(evidence, eventLogs, refValues)
     |                                          attestationResult <= |
     |                                                            |
```

The Attester boots up and thereby produces claims about its boot state and its operational state. Event Logs accompany the produced claims by providing an event trail of security-critical events in a system. Claims are produced by all attesting Environments of an Attester system.

The Challenge/Response remote attestation procedure is initiated by the Verifier by sending a remote attestation request to the Attester. A request includes a Handle, a list of Authentication Secret IDs, and a Claim Selection.

In the Challenge/Response model, the handle is composed of qualifying data in the form of a practically infeasible to guess nonce, such as a cryptographically strong random number. The Verifier-generated nonce is intended to guarantee Evidence freshness and to prevent replay attacks.

The list of Authentication Secret IDs selects the attestation keys with which the Attester is requested to sign the Attestation Evidence. Each selected key is uniquely associated with an Attesting Environment of the Attester. As a result, a single Authentication Secret ID identifies a single Attesting Environment. Correspondingly, a particular set of Evidence originating from a

particular Attesting Environment in a composite device can be requested via multiple Authentication Secret IDs. Methods to acquire Authentication Secret IDs or mappings between Attesting Environments to Authentication Secret IDs are out-of-scope of this document.

The Attester collects Claims based on the Claim Selection. With the Claim Selection the Verifier defines the set of Claims it requires. Correspondingly, collected Claims can be a subset of the produced Claims. This could be all available Claims, depending on the Claim Selection. If the Claim Selection is omitted, then by default all Claims that are known and available on the Attester MUST be used to create corresponding Evidence. For example, when performing a boot integrity evaluation, a Verifier may only be requesting a particular subset of claims about the Attester, such as Evidence about BIOS/UEFI and firmware that the Attester booted up, and not include information about all currently running software.

With the Handle, the Authentication Secret IDs, and the collected Claims, the Attester produces signed Evidence. That is, it digitally signs the Handle and the collected Claims with a cryptographic secret identified by the Authentication Secret ID. This is done once per Attesting Environment which is identified by the particular Authentication Secret ID. The Attester communicates the signed Evidence as well as all accompanying Event Logs back to the Verifier.

While it is crucial that Claims, the Handle, and the Attester Identity information (i.e., the Authentication Secret) MUST be cryptographically bound to the signature of Evidence, they MAY be presented obfuscated, encrypted, or cryptographically blinded. For further reference see section Section 10.

As soon as the Verifier receives the Evidence and the Event Logs, it appraises the Evidence. For this purpose, it validates the signature, the Attester Identity, and the Handle, and then appraises the Claims. Appraisal procedures are application-specific and can be conducted via comparison of the Claims with corresponding Reference Values, such as Reference Integrity Measurements. The final output of the Verifier are Attestation Results. Attestation Results constitute new Claim Sets about the properties and characteristics of an Attester, which enables Relying Parties, for example, to assess an Attester’s trustworthiness.

7.1.1. Models and Example Sequences of Challenge/Response Remote Attestation

According to the RATS Architecture, two reference models for Challenge/Response Attestation have been proposed. This section highlights the information flows between the Attester, Verifier, and Relying Party undergoing Remote Attestation Procedure, using these models.

7.1.1.1. Passport Model

The passport model is so named because of its resemblance to how nations issue passports to their citizens. In this model, the attestation sequence is a two-step procedure. In the first step, an Attester conveys Evidence to a Verifier, which compares the Evidence against its appraisal policy. The Verifier then gives back an Attestation Result to the Attester, which simply caches it. In the second step, the Attester presents the Attestation Result (and possibly additional Claims/Evidence) to a Relying Party, which then compares this information against its own appraisal policy to establish the trustworthiness of the Attester.

```
.----------.                          .----------.    .---------------.
| Attester |                          | Verifier |    | Relying Party |
'----+-----'                          '-----+----'    '-------+-------'
     |                                      |                 |
=================[Evidence Generation and Conveyance]===================
     |                                      |                 |
  generateClaims(attestingEnvironment)      |                 |
     | => claims, eventLogs                 |                 |
     |                                      |                 |
     |<--------------------- requestAttestation(handle,       |
     |                           authSecIDs, claimSelection)  |
     |                                      |                 |
  collectClaims(claims, claimSelection)     |                 |
     | => collectedClaims                   |                 |
     |                                      |                 |
  generateEvidence(handle,                  |                 |
     authSecIDs, collectedClaims)           |                 |
     | => evidence                          |                 |
     |                                      |                 |
     | {evidence, eventLogs} -------------->|                 |
     |                                      |                 |
=========================[Evidence Appraisal]=========================
     |                                      |                 |
     |                         appraiseEvidence(evidence,     |
     |                             eventLogs, refValues)      |
     |                 attestationResult <= |                 |
     |                                      |                 |
     |<------------------ attestationResult |                 |
     |                                      |                 |
     | {evidence, attestationResult} ------------------------>|
     |                                      |                 |
====================[Attestation Result Generation]=====================
     |                                      |                 |
     |                                      |    appraiseResult(policy,
     |                                      |       attestationResult)
     |                                      |                 |
```

7.1.1.2. Background-Check Model

The background-check model is so named because of the resemblance of how employers and volunteer organizations perform background checks. In this model, the attestation sequence is initiated by a Relying Party. The Attester conveys Evidence to the Relying Party, which does not process its payload, but relays the message and optionally checks its signature against a policed trust anchor store. Upon receiving the Evidence, the Relying Party initiates a session with the Verifier. Once the session is established, it forwards the received Evidence to the Verifier. The Verifier appraises the

received Evidence according to its appraisal policy for Evidence and returns a corresponding Attestation Result to the Relying Party. The Relying Party then checks the Attestation Result against its own appraisal policy to conclude attestation.

```
.----------.                    .---------------.         .----------.
| Attester |                    | Relying Party |         | Verifier |
'----+-----'                    '-------+-------'         '-----+----'
     |                                  |                       |
================[Evidence Generation and Conveyance]==================
     |                                  |                       |
     |<--------------------- requestAttestation(handle,          |
     |                           authSecIDs, claimSelection)     |
     |                                  |                       |
  generateClaims(attestingEnvironment)  |                       |
     | => {claims, eventLogs}           |                       |
     |                                  |                       |
  collectClaims(claims,                 |                       |
     claimSelection)                    |                       |
     | => collectedClaims               |                       |
     |                                  |                       |
  generateEvidence(handle,              |                       |
     authSecIDs, collectedClaims)       |                       |
     | => evidence                      |                       |
     |                                  |                       |
     | {evidence, eventLogs} ----------->|                      |
     |                                  |                       |
=========================[Evidence Appraisal]=========================
     |                                  |                       |
     |                                  | {handle, evidence,    |
     |                                  |  eventLogs} --------->|
     |                                  |                       |
     |                                  |  appraiseEvidence(evidence,
     |                                  |       eventLogs, refValues)
     |                                  | attestationResult <=  |
     |                                  |                       |
     |                                  |<---------- {evidence, |
     |                                  |    attestationResult} |
     |                                  |                       |
===================[Attestation Result Generation]===================
     |                                  |                       |
     |                        appraiseResult(policy,            |
     |                          attestationResult)              |
     |                                  |                       |
```

7.2. Uni-Directional Remote Attestation

```
.----------.                        .--------------------.   .----------.
| Attester |                        | Handle Distributor |   | Verifier |
'----+-----'                        '---------+----------'   '-----+----'
     |                                        |                     |
==========================[Handle Generation]==========================
     |                                   generateHandle()           |
     |                                       | => handle              |
     |                                       |                        |
     |<---------------------------- {handle} | {handle} --------->|
     |                                       |                        |
     |                                       x                        |
     |                                                                |
=================[Evidence Generation and Conveyance]===================
     |                                                                |
  generateClaims(attestingEnvironment)                                |
     | => claims, eventLogs                                           |
     |                                                                |
  collectClaims(claims, claimSelection)                               |
     | => collectedClaims                                             |
     |                                                                |
  generateEvidence(handle, authSecIDs, collectedClaims)               |
     | => evidence                                                    |
     |                                                                |
     | {evidence, eventLogs} ----------------------------------->|
     |                                                                |
==========================[Evidence Appraisal]==========================
     |                                                                |
     |                                     appraiseEvidence(evidence,
     |                                                     eventLogs,
     |                                                     refValues)
     |                                      attestationResult <= |
     ~                                                                ~
     |                                                                |
 .--------[loop]-------------------------------------------------------.
|    |                                                                |   |
| =============[Delta Evidence Generation and Conveyance]============ |
|    |                                                                |   |
| generateClaims(attestingEnvironment)                                |   |
|    | => claimsDelta, eventLogsDelta                                 |   |
|    |                                                                |   |
| collectClaims(claimsDelta, claimSelection)                          |   |
|    | => collectedClaimsDelta                                        |   |
|    |                                                                |   |
| generateEvidence(handle, authSecIDs, collectedClaimsDelta)          |   |
|    | => evidence                                                    |   |
|    |                                                                |   |
|    | {evidence, eventLogsDelta} ------------------------------->|   |
|    |                                                                |   |
```

```
| ====================[Delta Evidence Appraisal]==================== |
|    |                                                              |   |
|    |                                     appraiseEvidence(evidence, |
|    |                                                eventLogsDelta, |
|    |                                                     refValues) |
|    |                                      attestationResult <= |    |
|    |                                                              |   |
 '-------------------------------------------------------------------'
     |                                                              |
```

Uni-Directional Remote Attestation procedures can be initiated both by the Attester and by the Verifier. Initiation by the Attester can result in unsolicited pushes of Evidence to the Verifier. Initiation by the Verifier always results in solicited pushes to the Verifier.

The Uni-Directional model uses the same information elements as the Challenge/Response model. In the sequence diagram above, the Attester initiates the conveyance of Evidence (comparable with a RESTful POST operation or the emission of a beacon). While a request of Evidence from the Verifier would result in a sequence diagram more similar to the Challenge/Response model (comparable with a RESTful GET operation). The specific manner how Handles are created and used always remains as the distinguishing quality of this model.

In the Uni-Directional model, handles are composed of cryptographically signed trusted timestamps as shown in [I-D.birkholz-rats-tuda], potentially including other qualifying data. The Handles are created by an external 3rd entity -- the Handle Distributor -- which includes a trustworthy source of time, and takes on the role of a Time Stamping Authority (TSA, as initially defined in [RFC3161]). Timestamps created from local clocks (absolute clocks using a global timescale, as well as relative clocks, such as tick-counters) of Attesters and Verifiers MUST be cryptographically bound to fresh Handles received from the Handle Distributor. This binding provides a proof of synchronization that MUST be included in all produced Evidence. Correspondingly, conveyed Evidence in this model provides a proof that it was fresh at a certain point in time.

While periodically pushing Evidence to the Verifier, the Attester only needs to generate and convey evidence generated from Claim values that have changed and new Event Log entries since the previous conveyance. These updates reflecting the differences are called "delta" in the sequence diagram above.

Effectively, the Uni-Directional model allows for a series of Evidence to be pushed to multiple Verifiers simultaneously. Methods to detect excessive time drift that would mandate a fresh Handle to be received by the Handle Distributor as well as timing of Handle distribution are out-of-scope of this document.

7.3. Streaming Remote Attestation

Streaming Remote Attestation serves as the foundational concept for both the observer pattern ([ISIS]) and the publish-subscribe pattern ([DesignPatterns]). It entails establishing subscription states to enable continuous remote attestation. The observer pattern directly connects observers to subjects without a broker, while the publish-subscribe pattern involves a central broker for message distribution. In the following Subsections, streaming remote attestation without a broker (observer pattern) as well as with a broker (publish-subscribe pattern) are illustrated.

7.3.1. Streaming Remote Attestation without a Broker

```
.----------.                                                .----------.
| Attester |                                                | Verifier |
'----+-----'                                                '-----+----'
     |                                                            |
==========================[Handle Generation]==========================
     |                                                            |
     |                                                 generateHandle()
     |                                                    handle<= |
     |                                                            |
     |<------------ subscribe(handle, authSecIDs, claimSelection) |
     | {handle} ------------------------------------------------->|
     |                                                            |
=================[Evidence Generation and Conveyance]===================
     |                                                            |
  generateClaims(attestingEnvironment)                            |
     | => claims, eventLogs                                       |
     |                                                            |
  collectClaims(claims, claimSelection)                           |
     | => collectedClaims                                         |
     |                                                            |
  generateEvidence(handle, authSecIDs, collectedClaims)           |
     | => evidence                                                |
     |                                                            |
==========================[Evidence Appraisal]==========================
     |                                                            |
     | {handle, evidence, eventLogs} ---------------------------->|
     |                                                            |
     |                                        appraiseEvidence(evidence,
```

```
     |                                                    eventLogs,
     |                                                    refValues)
     |                                     attestationResult <= |
     ~                                                          ~
     |                                                          |
 .--------[loop]-------------------------------------------------.
|    |                                                          |    |
| =============[Delta Evidence Generation and Conveyance]============ |
|    |                                                          |    |
| generateClaims(attestingEnvironment)                          |    |
|    | => claimsDelta, eventLogsDelta                           |    |
|    |                                                          |    |
| collectClaims(claimsDelta, claimSelection)                    |    |
|    | => collectedClaimsDelta                                  |    |
|    |                                                          |    |
| generateEvidence(handle, authSecIDs, collectedClaimsDelta)    |    |
|    | => evidence                                              |    |
|    |                                                          |    |
| =====================[Delta Evidence Appraisal]===================== |
|    |                                                          |    |
|    | {evidence, eventLogsDelta} ------------------------------>|    |
|    |                                                          |    |
|    |                                       appraiseEvidence(evidence, |
|    |                                                 eventLogsDelta, |
|    |                                                      refValues) |
|    |                                     attestationResult <= |    |
|    |                                                          |    |
 '-----------------------------------------------------------------'
     |                                                          |
```

The observer pattern is employed in scenarios where message delivery does not involve a central broker. Instead, an observer directly subscribes to observed resources via a dedicated mechanism. Consequently, these dedicated mechanisms contain information about the observer and are responsible for maintaining subscription state. Setting up subscription state between a Verifier and an Attester is conducted via a subscribe operation. The subscribe operation is used to convey Handles required for Evidence generation. Effectively, this allows for a series of Evidence to be pushed to a Verifier, similar to the Uni-Directional model. While a Handle Distributor is not mandatory in this model, the model is also limited to bi-lateral subscription relationships, in which each Verifier has to create and provide Handles individually. Handles provided by a specific subscribing Verifier MUST be used in Evidence generation for that specific Verifier. The streaming model without a broker uses the same information elements as the Challenge/Response and the Uni-Directional model. Methods to detect excessive time drift that would render Handles stale and mandate a fresh Handles to be conveyed via another subscribe operation are out-of-scope of this document.

7.3.2. Streaming Remote Attestation with a Broker

The publish-subscribe messaging pattern is widely used for communication in different areas. Unlike the _Streaming Remote Attestation without a Broker_ interaction model, Attesters do not (need to) be aware of corresponding Verifiers. In scenarios with large numbers of Attesters and Verifiers, the publish-subscribe pattern may reduce interdependencies and improve scalability.

With publish-subscribe, clients typically _connect_ to (or _register_ with) a publish-subscribe server (PubSub server or Broker). Clients may _publish_ data in the form of a _message_ under a certain _topic_. _Subscribers_ to that topic get _notified_ whenever a message arrives under a topic, and the appropriate message is forwarded to them. Depending on the particular publish-subscribe model and implementation, clients can be either publishers or subscribers or both.

In the following sections, the interaction models _Challenge/Response Remote Attestation over Publish-Subscribe_ and _Uni-Directional Remote Attestation over Publish-Subscribe_ are described. There are different phases that both models go through:

1. Handle Generation

2. Evidence Generation and Conveyance

3. Evidence Appraisal

4. Attestation Result Generation

The models only differ in the handle generation phase. From a remote attestations procedure's point of view Evidence Generation, Conveyance, and Appraisal, as well as Attestation Result Generation are identical in both models.

7.3.2.1. Handle Generation for Challenge/Response Remote Attestation over Publish-Subscribe

```
.----------.                  .---------------.             .----------.
| Attester |                  | PubSub Server |             | Verifier |
'----+-----'                  '-------+-------'             '-----+----'
     |                                |                           |
==========================[Handle Generation]==========================
     |                                |                           |
   sub(topic=AttReq) ---------------->|                           |
     |                                |<------------ pub(topic=AttReq,
     |                                |                        handle)
     |<------------------- notify(topic=AttReq, handle)           |
     |                                |                           |
     ~                                ~                           ~
```

The _Challenge/Response Remote Attestation over Publish-Subscribe_ interaction model uses the same information elements as the _Challenge/Response Remote Attestation_ interaction model. Handles are provided by a Verifier on a per-request basis. In the sequence diagram above, an Attester subscribes to the "AttReq" (= Attestation Request) topic on the PubSub server. The Verifier publishes a Handle to the "AttReq" topic, which the PubSub server forwards to the Attester by notifying it.

7.3.2.2. Handle Generation for Uni-Directional Remote Attestation over Publish-Subscribe

```
.----------.  .-------------.    .---------------.          .----------.
| Attester |  |   Handle    |    | PubSub Server |          | Verifier |
'----+-----'  | Distributor |    '-------+-------'          '-----+----'
     |        '------+------'            |                        |
     |               |                   |                        |
=========================[Handle Generation]==========================
     |               |                   |                        |
     |               |                   |                        |
   sub(topic=Handle) ------------------->|                        |
     |               |                   |                        |
     |               |                   |<--------- sub(topic=Handle)
     |               |                   |                        |
     |               |                   |                        |
     |           generateHandle()        |                        |
     |               | => handle         |                        |
     |               |                   |                        |
     |             pub(topic=Handle,     |                        |
     |               | handle) --------->|                        |
     |               x                   |                        |
     |                                   |                        |
     |<--------------------- notify(topic=Handle, handle)         |
     |                                   |                        |
     |                       notify(topic=Handle, handle) ------->|
     |                                   |                        |
     ~                                   ~                        ~
```

The _Uni-Directional Remote Attestation over Publish-Subscribe_ model uses the same information elements as the Uni-Directional Remote Attestation model. Accordingly, Handles are created by a 3rd party, the Handle Distributor. In the sequence diagram above, both an Attester and a Verifier subscribe to the topic "Handle" on the PubSub server. When the Handle Distributor generates and publishes a Handle to the "Handle" topic on the PubSub server, the PubSub server notifies the subscribers, Attester and Verifier, and forwards ("notify") the Handle to them during Handle Generation.

7.3.2.3. Evidence Generation and Appraisal

```
     ~                                  ~                       ~
     |                                  |                       |
.----+-----.                     .-------+-------.          .-----+----.
| Attester |                     | PubSub Server |          | Verifier |
'----+-----'                     '-------+-------'          '-----+----'
     |                                  |                       |
     |                                  |<---------- sub(topic=AttEv)
     |                                  |                       |
 .--------[loop]-----------------------------------------------------.
|    |                                  |                       |    |
| ===============[Evidence Generation and Conveyance]================ |
|    |                                  |                       |    |
| generateClaims(attestingEnvironment)  |                       |    |
|    | => claims, eventLogs             |                       |    |
|    |                                  |                       |    |
| collectClaims(claims, claimSelection) |                       |    |
|    | => collectedClaims               |                       |    |
|    |                                  |                       |    |
| generateEvidence(handle, authSecIDs,  |                       |    |
|    |             collectedClaims)     |                       |    |
|    | => evidence                      |                       |    |
|    |                                  |                       |    |
|  pub(topic=AttEv,                     |                       |    |
|    | evidence, eventLogs) ------------>|                       |    |
|    |                                  |                       |    |
|    |                                  |notify(topic=AttEv,    |    |
|    |                                  |  evidence,            |    |
|    |                                  |  eventLogs) ---------->|    |
|    |                                  |                       |    |
| ========================[Evidence Appraisal]======================== |
|    |                                  |                       |    |
|    |                                  |           appraiseEvidence( |
|    |                                  |                   evidence, |
|    |                                  |                  eventLogs, |
|    |                                  |                  refValues) |
|    |                                  |   attestationResult <= |    |
|    |                                  |                       |    |
| ==================[Attestation Result Generation]=================== |
|    |                                  |                       |    |
|    |                                  |<--------- pub(topic=AttRes, |
|    |                                  |          attestationResult) |
|    |                                  |                       |    |
 '-------------------------------------------------------------------'
     |                                  |                       |
     ~                                  ~                       ~
```

Exactly as in the Challenge/Response and Uni-Directional interaction models, there is an Evidence Generation-Appraisal loop, in which the Attester generates Evidence and the Verifier appraises it. In the Publish-Subscribe model above, the Attester publishes Evidence to the topic "AttEv" (= Attestation Evidence) on the PubSub server, to which a Verifier subscribed before. The PubSub server notifies Verifiers, accordingly, by forwarding the attestation Evidence. Although the above diagram depicts only full attestation Evidence and Event Logs, later attestations may use "deltas' for Evidence and Event Logs. Verifiers appraise the Evidence and publish the Attestation Result to topic "AttRes" (= Attestation Result) on the PubSub server.

7.3.2.4. Attestation Result Generation

```
     ~          ~                       ~                       ~
     |          |                       |                       |
.----+-----. .--+------------.   .-------+-------.          .-----+----.
| Attester | | Relying Party |   | PubSub Server |          | Verifier |
'----+-----' '--+------------'   '-------+-------'          '-----+----'
     |          |                       |                       |
====================[Attestation Result Generation]====================
     |          |                       |                       |
     |     sub(topic=AttRes)            |                       |
     |         handle) ----------------->|                       |
     |          |                       |                       |
 .--------[loop]-------------------------------------------------------.
|    |          |                       |                       |    |
|    |          |                       |<--------- pub(topic=AttRes, |
|    |          |                       |          attestationResult) |
|    |          |                       |                       |    |
|    |          |<----------------- notify(topic=AttRes         |    |
|    |          |                       | attestationResult)    |    |
|    |          |                       |                       |    |
 '-----------------------------------------------------------------'
     |          |                       |                       |
     ~          ~                       ~                       ~
```

Attestation Result Generation is the same for both publish-subscribe models,_Challenge/Response Remote Attestation over Publish-Subscribe_ and _Uni-Directional Remote Attestation over Publish-Subscribe_. Relying Parties subscribe to topic AttRes (= Attestation Result) on the PubSub server. The PubSub server forwards Attestation Results to the Relying Parties as soon as they are published to topic AttRes.

7.3.2.5. Publish/Subscribe Topics

Many publish-subscribe models provide hierarchical organization of topics. This way, subscribers can subscribe to either all attestations (topic AttRes), or, for example, to topic AttRes/DbServers/Germany to receive only attestations from database servers in Germany. Further, it may be required to distinguish between uni-directional and challenge-response attestation evidence.

8. Additional Application-Specific Requirements

Depending on the use cases covered, there can be additional requirements. An exemplary subset is illustrated in this section.

8.1. Confidentiality

Confidentiality of exchanged attestation information may be desirable. This requirement usually is present when communication takes place over insecure channels, such as the public Internet. In such cases, TLS may be used as a suitable communication protocol which provides confidentiality protection. In private networks, such as carrier management networks, it must be evaluated whether or not the transport medium is considered confidential.

8.2. Mutual Authentication

In particular use cases, mutual authentication may be desirable in such a way that a Verifier also needs to prove its identity to the Attester, instead of only the Attester proving its identity to the Verifier.

8.3. Hardware-Enforcement/Support

Depending on given usage scenarios, hardware support for secure storage of cryptographic keys, crypto accelerators, as well as protected or isolated execution environments can be mandatory requirements. Well-known technologies in support of these requirements are roots of trusts, such as Hardware Security Modules (HSM), Physically Unclonable Functions (PUFs), Shielded Secrets, or Trusted Executions Environments (TEEs).

9. Implementation Status

Note to RFC Editor: Please remove this section as well as references to [BCP205] before AUTH48.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [BCP205]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [BCP205], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. Implementer

The open-source implementation was initiated and is maintained by the Fraunhofer Institute for Secure Information Technology SIT.

9.2. Implementation Name

The open-source implementation is named "CHAllenge-Response based Remote Attestation" or in short: CHARRA.

9.3. Implementation URL

The open-source implementation project resource can be located via: https://github.com/fraunhofer-sit/charra (https://github.com/fraunhofer-sit/charra)

9.4. Maturity

The code’s level of maturity is considered to be "prototype".

9.5. Coverage and Version Compatibility

The current version (’6194b3b’) implements a challenge/response interaction model and is aligned with the exemplary specification of the CoAP FETCH bodies defined in Section Appendix A of this document.

9.6. License

The CHARRA project and all corresponding code and data maintained on GitHub are provided under the BSD 3-Clause "New" or "Revised" license.

9.7. Implementation Dependencies

The implementation requires the use of the official Trusted Computing Group (TCG) open-source Trusted Software Stack (TSS) for the Trusted Platform Module (TPM) 2.0. The corresponding project resources (code and data) for Linux-based operating systems are maintained on GitHub at https://github.com/tpm2-software/tpm2-tss/ (https://github.com/tpm2-software/tpm2-tss/).

The implementation uses the Constrained Application Protocol [RFC7252] (http://coap.technology/) and the Concise Binary Object Representation [RFC7049] (https://cbor.io/).

9.8. Contact

Michael Eckel (michael.eckel@sit.fraunhofer.de)

10. Security and Privacy Considerations

In a remote attestation procedure the Verifier or the Attester MAY want to cryptographically blind several attributes. For instance, information can be part of the signature after applying a one-way function (e. g., a hash function).

There is also a possibility to scramble the Nonce or Attester Identity with other information that is known to both the Verifier and Attester. A prominent example is the IP address of the Attester that usually is known by the Attester itself as well as the Verifier. This extra information can be used to scramble the Nonce in order to counter certain types of relay attacks.

11. Acknowledgments

Olaf Bergmann, Michael Richardson, and Ned Smith

12. References

12.1. Normative References

[BCP205] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://doi.org/10.17487/RFC7942>.

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

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://doi.org/10.17487/RFC3161>.

[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://doi.org/10.17487/RFC5280>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://doi.org/10.17487/RFC7049>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://doi.org/10.17487/RFC7252>.

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

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

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

12.2. Informative References

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

[DesignPatterns] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software", Publisher Addison-Wesley, 1994.

[I-D.birkholz-rats-tuda] Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-07, 10 July 2022, <https://datatracker.ietf.org/doc/html/draft-birkholz-rats-tuda-07>.

[I-D.ietf-rats-tpm-based-network-device-attest] Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", Work in Progress, Internet-Draft, draft-ietf-rats-tpm-based-network-device-attest-14, 22 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-tpm-based-network-device-attest-14>.

[ISIS] Birman, K. and T. Joseph, "Exploiting Virtual Synchrony in Distributed Systems", DOI 10.1145/41457.37515, 1987, <https://doi.org/10.1145/41457.37515>.

[MQTT] OASIS, "Message Queuing Telemetry Transport (MQTT) Version 5.0 Committee Specification 02", Specification Version 5.0, 2018.

[TNC] TCG, "TCG Trusted Network Communications TNC Architecture for Interoperability", Specification Version 2.0 Revision 13, 2017.

[turtles] Rudnicki, R., "Turtles All the Way Down: Foundation, Edifice, and Ruin in Faulkner and McCarthy", DOI 10.1353/fau.2010.0002, The Faulkner Journal 25.2, 2010, <https://doi.org/10.1353/fau.2010.0002>.

Appendix A. CDDL Specification for a simple CoAP Challenge/Response Interaction

The following CDDL specification is an exemplary proof-of-concept to illustrate a potential implementation of the Challenge/Response Interaction Model. The communication protocol used is CoAP. Both the request message and the response message are exchanged via the FETCH operation and corresponding FETCH request and FETCH response body.

In this example, Evidence is created via the root-of-trust for reporting primitive operation "quote" that is provided by a TPM 2.0.

```
charra-bodies = charra-attestation-request / charra-attestation-response

charra-attestation-request = [
    hello: bool,    ; if true, the TPM 2.0 AK Cert shall be conveyed
    key-id: bytes,  ; the key ID to use for signing
    nonce: bytes,   ; a (random) nonce, providing freshness and/or recentness
    pcr-selections: [ * pcr-selection ]
]

pcr-selection = [
    tcg-hash-alg-id: uint .size 2,  ; TPM2_ALG_ID
    pcrs: [
        pcr: uint .size 2
    ]
]

charra-attestation-response = [
    attestation-data: bytes,  ; TPMS_ATTEST.quoted
    tpm2-signature: bytes,
    ? ak-cert: bytes,         ; TPM2 attestation key certificate (AK Cert)
]
```

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@ietf.contact

Michael Eckel
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: michael.eckel@sit.fraunhofer.de

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

Eric Voit
Cisco Systems
Email: evoit@cisco.com

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 23 September 2022

G. C. Fedorkow, Ed.
Juniper Networks, Inc.
E. Voit
Cisco
J. Fitzgerald-McKay
National Security Agency
22 March 2022

TPM-based Network Device Remote Integrity Verification
draft-ietf-rats-tpm-based-network-device-attest-14

Abstract

This document describes a workflow for remote attestation of the integrity of firmware and software installed on network devices that contain Trusted Platform Modules [TPM1.2], [TPM2.0], as defined by the Trusted Computing Group (TCG)), or equivalent hardware implementations that include the protected capabilities, as provided by TPMs.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

There are many aspects to consider in fielding a trusted computing device, from operating systems to applications. Mechanisms to prove that a device installed at a customer's site is authentic (i.e., not counterfeit) and has been configured with authorized software, all as part of a trusted supply chain, are just a few of the many aspects which need to be considered concurrently to have confidence that a device is truly trustworthy.

A generic architecture for remote attestation has been defined in [I-D.ietf-rats-architecture]. Additionally, use cases for remotely attesting networking devices are discussed within Section 6 of [I-D.richardson-rats-usecases]. However, these documents do not provide sufficient guidance for network equipment vendors and operators to design, build, and deploy interoperable devices.

The intent of this document is to provide such guidance. It does this by outlining the Remote Integrity Verification (RIV) problem, and then identifies elements that are necessary to get the complete, scalable attestation procedure working with commercial networking products such as routers, switches and firewalls. An underlying assumption will be the availability within the device of a Trusted Platform Module [TPM1.2], [TPM2.0] compatible cryptoprocessor to enable the trustworthy remote assessment of the device's software and hardware.

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

A number of terms are reused from [I-D.ietf-rats-architecture]. These include: Appraisal Policy for Evidence, Attestation Result, Attester, Evidence, Reference Value, Relying Party, Verifier, and Verifier Owner.

Additionally, this document defines the following term:

Attestation: the process of generating, conveying and appraising claims, backed by evidence, about device trustworthiness characteristics, including supply chain trust, identity, device provenance, software configuration, device composition, compliance to test suites, functional and assurance evaluations, etc.

The goal of attestation is simply to assure an administrator or auditor that the device configuration and software that was launched when the device was last started is authentic and untampered-with. The determination of software authenticity is not prescribed in this document, but it's typically taken to mean a software image generated by an authority trusted by the administrator, such as the device manufacturer.

Within the Trusted Computing Group (TCG) context, the scope of attestation is typically narrowed to describe the process by which an independent Verifier can obtain cryptographic proof as to the identity of the device in question, and evidence of the integrity of software loaded on that device when it started up, and then verify that what's there matches the intended configuration. For network equipment, a Verifier capability can be embedded in a Network Management Station (NMS), a posture collection server, or other network analytics tool (such as a software asset management solution, or a threat detection and mitigation tool, etc.). While informally referred to as attestation, this document focuses on a specific subset of attestation tasks, defined here as Remote Integrity Verification (RIV). RIV in this document takes a network-equipment-centric perspective that includes a set of protocols and procedures for determining whether a particular device was launched with authentic software, starting from Roots of Trust. While there are many ways to accomplish attestation, RIV sets out a specific set of protocols and tools that work in environments commonly found in network equipment. RIV does not cover other device characteristics that could be attested (e.g., geographic location, connectivity; see [I-D.richardson-rats-usecases]), although it does provide evidence of a secure infrastructure to increase the level of trust in other device characteristics attested by other means (e.g., by Entity Attestation Tokens [I-D.ietf-rats-eat]).

In line with [I-D.ietf-rats-architecture] definitions, this document uses the term Endorser to refer to the role that signs identity and attestation certificates used by the Attester, while Reference Values are signed by a Reference Value Provider. Typically, the manufacturer of a network device would be accepted as both the Endorser and Reference Value Provider, although the choice is ultimately up to the Verifier Owner.

1.3. Document Organization

The remainder of this document is organized into several sections:

* The remainder of this section covers goals and requirements, plus a top-level description of RIV.

* The Solution Overview section outlines how Remote Integrity Verification works.

* The Standards Components section links components of RIV to normative standards.

* Privacy and Security shows how specific features of RIV contribute to the trustworthiness of the Attestation Result.

* Supporting material is in an appendix at the end.

1.4. Goals

Network operators benefit from a trustworthy attestation mechanism that provides assurance that their network comprises authentic equipment, and has loaded software free of known vulnerabilities and unauthorized tampering. In line with the overall goal of assuring integrity, attestation can be used to assist in asset management, vulnerability and compliance assessment, plus configuration management.

The RIV attestation workflow outlined in this document is intended to meet the following high-level goals:

* Provable Device Identity - This specification requires that an Attester (i.e., the attesting device) includes a cryptographic identifier unique to each device. Effectively this means that the device’s TPM must be so provisioned during the manufacturing cycle.

* Software Inventory - A key goal is to identify the software release(s) installed on the Attester, and to provide evidence that the software stored within hasn’t been altered without authorization.

* Verifiability - Verification of software and configuration of the device shows that the software that the administrator authorized for use was actually launched.

In addition, RIV is designed to operate either in a centralized environment, such as with a central authority that manages and configures a number of network devices, or 'peer-to-peer', where network devices independently verify one another to establish a trust relationship. (See Section 3.3 below)

1.5. Description of Remote Integrity Verification (RIV)

Attestation requires two interlocking mechanisms between the Attester network device and the Verifier:

* Device Identity, the mechanism providing trusted identity, can reassure network managers that the specific devices they ordered from authorized manufacturers for attachment to their network are those that were installed, and that they continue to be present in their network. As part of the mechanism for Device Identity, cryptographic proof of the identity of the manufacturer is also provided.

* Software Measurement is the mechanism that reports the state of mutable software components on the device, and can assure administrators that they have known, authentic software configured to run in their network.

Using these two interlocking mechanisms, RIV is a component in a chain of procedures that can assure a network operator that the equipment in their network can be reliably identified, and that authentic software of a known version is installed on each device. Equipment in the network includes devices that make up the network itself, such as routers, switches and firewalls.

Software used to boot a device can be identified by a chain of measurements, anchored at the start by a Root of Trust for Measurement (see Section 9.2), each measuring the next stage and recording the result in tamper-resistant storage, normally ending when the system software is fully loaded. A measurement signifies the identity, integrity and version of each software component registered with an Attester's TPM [TPM1.2], [TPM2.0], so that a subsequent verification stage can determine if the software installed is authentic, up-to-date, and free of tampering.

RIV includes several major processes, split between the Attester and Verifier:

1. Generation of Evidence is the process whereby an Attester generates cryptographic proof (Evidence) of claims about device properties. In particular, the device identity and its software configuration are both of critical importance.

2. Device Identification refers to the mechanism assuring the Relying Party (ultimately, a network administrator) of the identity of devices that make up their network, and that their manufacturers are known.

3. Conveyance of Evidence reliably transports the collected Evidence from Attester to a Verifier to allow a management station to perform a meaningful appraisal in Step 4. The transport is typically carried out via a management network. While not required for reliable attestation, an encrypted channel may be used to provide integrity, authenticity, or confidentiality once attestation is complete. It should be noted that critical attestation evidence from the TPM is signed by a key known only to TPM, and is not dependent on encyption carried out as part of a reliable transport.

4. Finally, Appraisal of Evidence occurs. This is the process of verifying the Evidence received by a Verifier from the Attester, and using an Appraisal Policy to develop an Attestation Result, used to inform decision-making. In practice, this means comparing the Attester's measurements reported as Evidence with the device configuration expected by the Verifier. Subsequently, the Appraisal Policy for Evidence might match Evidence found against Reference Values (aka Golden Measurements), which represent the intended configured state of the connected device.

All implementations supporting this RIV specification require the support of the following three technologies:

1. Identity: Device identity in RIV is based on IEEE 802.1AR Device Identity (DevID) [IEEE-802-1AR], coupled with careful supply-chain management by the manufacturer. The Initial DevID (IDevID) certificate contains a statement by the manufacturer that establishes the identity of the device as it left the factory. Some applications with a more-complex post-manufacture supply chain (e.g., Value Added Resellers), or with different privacy concerns, may want to use alternative mechanisms for platform authentication (for example, TCG Platform Certificates [Platform-Certificates], or post-manufacture installation of Local Device ID (LDevID)).

2. Platform Attestation provides evidence of configuration of software elements present in the device. This form of attestation can be implemented with TPM Platform Configuration Registers (PCRs), Quote and Log mechanisms, which provide cryptographically authenticated evidence to report what software was started on the device through the boot cycle. Successful attestation requires an unbroken chain from a boot-time root of

trust through all layers of software needed to bring the device to an operational state, in which each stage computes the hash of components of the next stage, then updates the attestation log and the TPM. The TPM can then report the hashes of all the measured hashes as signed evidence called a Quote (see Section 9.1 for an overview of TPM operation, or [TPM1.2] and [TPM2.0] for many more details).

3. Signed Reference Values (aka Reference Integrity Measurements) must be conveyed from the Reference Value Provider (the entity accepted as the software authority, often the manufacturer of the network device) to the Verifier.

1.6. Solution Requirements

Remote Integrity Verification must address the "Lying Endpoint" problem, in which malicious software on an endpoint may subvert the intended function, and also prevent the endpoint from reporting its compromised status. (See Section 5 for further Security Considerations.)

RIV attestation is designed to be simple to deploy at scale. RIV should work "out of the box" as far as possible, that is, with the fewest possible provisioning steps or configuration databases needed at the end-user's site. Network equipment is often required to "self-configure", to reliably reach out without manual intervention to prove its identity and operating posture, then download its own configuration, a process which precludes pre-installation configuration. See [RFC8572] for an example of Secure Zero Touch Provisioning.

1.7. Scope

The need for assurance of software integrity, addressed by Remote Attestation, is a very general problem that could apply to most network-connected computing devices. However, this document includes several assumptions that limit the scope to network equipment (e.g., routers, switches and firewalls):

* This solution is for use in non-privacy-preserving applications (for example, networking, Industrial IoT), avoiding the need for a Privacy Certificate Authority (also called an Attestation CA) for attestation keys [AK-Enrollment] or TCG Platform Certificates [Platform-Certificates].

* This document assumes network protocols that are common in network equipment such as YANG [RFC7950] and NETCONF [RFC6241], but not generally used in other applications.

* The approach outlined in this document mandates the use of a TPM [TPM1.2], [TPM2.0], or a compatible cryptoprocessor.

1.7.1. Out of Scope

* Run-Time Attestation: The Linux Integrity Measurement Architecture [IMA] attests each process launched after a device is started (and is in scope for RIV in general), but continuous run-time attestation of Linux or other multi-threaded operating system processes after the OS has started considerably expands the scope of the problem. Many researchers are working on that problem, but this document defers the problem of continuous, in-memory run-time attestation.

* Multi-Vendor Embedded Systems: Additional coordination would be needed for devices that themselves comprise hardware and software from multiple vendors, integrated by the end user. Although out of scope for this document, these issues are accommodated in [I-D.ietf-rats-architecture].

* Processor Sleep Modes: Network equipment typically does not "sleep", so sleep and hibernate modes are not considered. Although out of scope for RIV in this document, Trusted Computing Group specifications do encompass sleep and hibernate states, which could be incorporated into remote attestation for network equipment in the future, given a compelling need.

* Virtualization and Containerization: In a non-virtualized system, the host OS is responsible for measuring each User Space file or process throughout the operational lifetime of the system. For virtualized systems, the host OS must verify the hypervisor, but then the hypervisor must manage its own chain of trust through the virtual machine. Virtualization and containerization technologies are increasingly used in network equipment, but are not considered in this document.

2. Solution Overview

2.1. RIV Software Configuration Attestation using TPM

RIV Attestation is a process which can be used to determine the identity of software running on a specifically-identified device. The Remote Attestation steps of Section 1.5 are broken into two phases, shown in Figure 1:

* During system startup, or boot phase, each distinct software object is "measured" by the Attester. The object’s identity, hash (i.e., cryptographic digest) and version information are recorded

in a log. Hashes are also extended into the TPM (see Section 9.1 for more on 'extending hashes'), in a way that can be used to validate the log entries. The measurement process generally follows the layered chain-of-trust model used in Measured Boot, where each stage of the system measures the next one, and extends its measurement into the TPM, before launching it. See [I-D.ietf-rats-architecture], section "Layered Attestation Environments," for an architectural definition of this model.

* Once the device is running and has operational network connectivity, verification can take place. A separate Verifier, running in its own trusted environment, will interrogate the network device to retrieve the logs and a copy of the digests collected by hashing each software object, signed by an attestation private key secured by, but never released by, the TPM. The YANG model described in [I-D.ietf-rats-yang-tpm-charra] facilitates this operation.

The result is that the Verifier can verify the device's identity by checking the subject[RFC5280] and signature of the certificate containing the TPM's attestation public key, and can validate the software that was launched by verifying the correctness of the logs by comparing with the signed digests from the TPM, and comparing digests in the log with Reference Values.

It should be noted that attestation and identity are inextricably linked; signed Evidence that a particular version of software was loaded is of little value without cryptographic proof of the identity of the Attester producing the Evidence.

```
       +-------------------------------------------------------+
       | +---------+    +--------+   +--------+    +---------+ |
       | |UEFI BIOS|--->| Loader |-->| Kernel |--->|Userland | |
       | +---------+    +--------+   +--------+    +---------+ |
       |     |            |           |                        |
       |     |            |           |                        |
       |     +------------+-----------+-+                      |
       |                    Boot Phase  |                      |
       |                                V                      |
       |                            +--------+                 |
       |                            |  TPM   |                 |
       |                            +--------+                 |
       |   Router                       |                      |
       +--------------------------------|----------------------+
                                        |
                                        |  Verification Phase
                                        |    +-----------+
                                        +--->| Verifier  |
                                             +-----------+

       Reset---------------flow-of-time-during-boot...--------->
```

Figure 1: Layered RIV Attestation Model

In the Boot phase, measurements are "extended", or hashed, into the TPM as processes start, with the result that the TPM ends up containing hashes of all the measured hashes. Later, once the system is operational, during the Verification phase, signed digests are retrieved from the TPM for off-box analysis.

2.1.1. What Does RIV Attest?

TPM attestation is focused on Platform Configuration Registers (PCRs), but those registers are only vehicles for certifying accompanying Evidence, conveyed in log entries. It is the hashes in log entries that are extended into PCRs, where the final PCR values can be retrieved in the form of a structure called a Quote, signed by an Attestation key known only to the TPM. The use of multiple PCRs serves only to provide some independence between different classes of object, so that one class of objects can be updated without changing the extended hash for other classes. Although PCRs can be used for any purpose, this section outlines the objects within the scope of this document which may be extended into the TPM.

In general, assignment of measurements to PCRs is a policy choice made by the device manufacturer, selected to independently attest three classes of object:

* Code, (i.e., instructions) to be executed by a CPU.

* Configuration - Many devices offer numerous options controlled by non-volatile configuration variables which can impact the device's security posture. These settings may have vendor defaults, but often can be changed by administrators, who may want to verify via attestation that the operational state of the settings match their intended state.

* Credentials - Administrators may wish to verify via attestation that public keys and credentials outside the Root of Trust have not been subject to unauthorized tampering. (By definition, keys protecting the root of trust can't be verified independently.)

The TCG PC Client Platform Firmware Profile Specification [PC-Client-BIOS-TPM-2.0] gives considerable detail on what is to be measured during the boot phase of platform startup using a UEFI BIOS (www.uefi.org), but the goal is simply to measure every bit of code executed in the process of starting the device, along with any configuration information related to security posture, leaving no gap for unmeasured code to remain undetected, potentially subverting the chain.

For devices using a UEFI BIOS, [PC-Client-BIOS-TPM-2.0] and [PC-Client-EFI-TPM-1.2] give detailed normative requirements for PCR usage. For other platform architectures, where TCG normative requirements currently do not exist, the table in Figure 2 gives non-normative guidance for PCR assignment that generalizes the specific details of [PC-Client-BIOS-TPM-2.0].

By convention, most PCRs are assigned in pairs, which the even-numbered PCR used to measure executable code, and the odd-numbered PCR used to measure whatever data and configuration are associated with that code. It is important to note that each PCR may contain results from dozens (or even thousands) of individual measurements.

Function	Assigned PCR # Code	Assigned PCR # Configuration
Firmware Static Root of Trust, (i.e., initial boot firmware and drivers)	0	1
Drivers and initialization for optional or add-in devices	2	3
OS Loader code and configuration, (i.e., the code launched by firmware) to load an operating system kernel. These PCRs record each boot attempt, and an identifier for where the loader was found	4	5
Vendor Specific Measurements during boot	6	6
Secure Boot Policy. This PCR records keys and configuration used to validate the OS loader		7
Measurements made by the OS Loader (e.g. GRUB2 for Linux)	8	9
Measurements made by OS (e.g., Linux IMA)	10	10

Figure 2: Attested Objects

2.1.2. Notes on PCR Allocations

It is important to recognize that PCR[0] is critical. The first measurement into PCR[0] is taken by the Root of Trust for Measurement, code which, by definition, cannot be verified by measurement. This measurement establishes the chain of trust for all subsequent measurements. If the PCR[0] measurement cannot be trusted, the validity of the entire chain is put into question.

Distinctions Between PCR[0], PCR[2], PCR[4] and PCR[8] are summarized below:

* PCR[0] typically represents a consistent view of rarely-changed Host Platform boot components, allowing Attestation policies to be defined using the less changeable components of the transitive trust chain. This PCR typically provides a consistent view of the platform regardless of user selected options.

* PCR[2] is intended to represent a "user configurable" environment where the user has the ability to alter the components that are measured into PCR[2]. This is typically done by adding adapter cards, etc., into user-accessible PCI or other slots. In UEFI systems these devices may be configured by Option ROMs measured into PCR[2] and executed by the UEFI BIOS.

* PCR[4] is intended to represent the software that manages the transition between the platform's Pre-Operating System start and the state of a system with the Operating System present. This PCR, along with PCR[5], identifies the initial operating system loader (e.g., GRUB for Linux).

* PCR[8] is used by the OS loader (e.g. GRUB) to record measurements of the various components of the operating system.

Although the TCG PC Client document specifies the use of the first eight PCRs very carefully to ensure interoperability among multiple UEFI BIOS vendors, it should be noted that embedded software vendors may have considerably more flexibility. Verifiers typically need to know which log entries are consequential and which are not (possibly controlled by local policies) but the Verifier may not need to know what each log entry means or why it was assigned to a particular PCR. Designers must recognize that some PCRs may cover log entries that a particular Verifier considers critical and other log entries that are not considered important, so differing PCR values may not on their own constitute a check for authenticity. For example, in a UEFI system, some administrators may consider booting an image from a removable drive, something recorded in a PCR, to be a security violation, while others might consider that operation an authorized recovery procedure.

Designers may allocate particular events to specific PCRs in order to achieve a particular objective with local attestation, (e.g., allowing a procedure to execute, or releasing a particular decryption key, only if a given PCR is in a given state). It may also be important to designers to consider whether streaming notification of PCR updates is required (see [I-D.birkholz-rats-network-device-subscription]). Specific log entries can only be validated if the Verifier receives every log entry affecting the relevant PCR, so (for example) a designer might want to separate rare, high-value events such as configuration changes, from high-volume, routine measurements such as IMA [IMA] logs.

2.2. RIV Keying

RIV attestation relies on two credentials:

* An identity key pair and matching certificate is required to certify the identity of the Attester itself. RIV specifies the use of an IEEE 802.1AR Device Identity (DevID) [IEEE-802-1AR], signed by the device manufacturer, containing the device serial number. This requirement goes slightly beyond 802.1AR; see Section 2.4 for notes.

* An Attestation key pair and matching certificate is required to sign the Quote generated by the TPM to report evidence of software configuration.

In a TPM application, both the Attestation private key and the DevID private key MUST be protected by the TPM. Depending on other TPM configuration procedures, the two keys are likely to be different; some of the considerations are outlined in TCG "TPM 2.0 Keys for Device Identity and Attestation" [Platform-DevID-TPM-2.0].

The TCG TPM 2.0 Keys document [Platform-DevID-TPM-2.0] specifies further conventions for these keys:

* When separate Identity and Attestation keys are used, the Attestation Key (AK) and its X.509 certificate should parallel the DevID, with the same unique device identification as the DevID certificate (that is, the same subject and subjectAltName (if present), even though the key pairs are different). This allows a quote from the device, signed by an AK, to be linked directly to the device that provided it, by examining the corresponding AK certificate. If the subject in the AK certificate doesn’t match the corresponding DevID certificate, or they’re signed by differing authorities the Verifier may signal the detection of an Asokan-style person-in-the-middle attack (see Section 5.2).

* Network devices that are expected to use secure zero touch provisioning as specified in [RFC8572] MUST be shipped by the manufacturer with pre-provisioned keys (Initial DevID and Initial AK, called IDevID and IAK). IDevID and IAK certificates MUST both be signed by the Endorser (typically the device manufacturer). Inclusion of an IDevID and IAK by a vendor does not preclude a mechanism whereby an administrator can define Local Identity and Attestation Keys (LDevID and LAK) if desired.

2.3. RIV Information Flow

RIV workflow for network equipment is organized around a simple use case where a network operator wishes to verify the integrity of software installed in specific, fielded devices. A normative taxonomy of terms is given in [I-D.ietf-rats-architecture], but as a reminder, this use case implies several roles and objects:

1. The Attester, the device which the network operator wants to examine.

2. A Verifier (which might be a network management station) somewhere separate from the Device that will retrieve the signed evidence and measurement logs, and analyze them to pass judgment on the security posture of the device.

3. A Relying Party, which can act on Attestation Results. Interaction between the Relying Party and the Verifier is considered out of scope for RIV.

4. Signed Reference Integrity Manifests (RIMs), containing Reference Values, can either be created by the device manufacturer and shipped along with the device as part of its software image, or alternatively, could be obtained several other ways (direct to the Verifier from the manufacturer, from a third party, from the owner's observation of what's thought to be a "known good system", etc.). Retrieving RIMs from the device itself allows attestation to be done in systems that may not have access to the public internet, or by other devices that are not management stations per se (e.g., a peer device; see Section 3.1.3). If Reference Values are obtained from multiple sources, the Verifier may need to evaluate the relative level of trust to be placed in each source in case of a discrepancy.

These components are illustrated in Figure 3.

```
+----------------+        +-------------+        +---------+--------+
|Reference Value |        | Attester    | Step 1 | Verifier|        |
|Provider        |        | (Device     |<-------| (Network| Relying|
|(Device         |        | under       |------->| Mngmt   | Party  |
|Manufacturer    |        | attestation)| Step 2 | Station)|        |
|or other        |        |             |        |         |        |
|authority)      |        |             |        |         |        |
+----------------+        +-------------+        +---------+--------+
       |                                              /\
       |                  Step 0                       |
       ------------------------------------------------
```

Figure 3: RIV Reference Configuration for Network Equipment

* In Step 0, The Reference Value Provider (the device manufacturer or other authority) makes one or more Reference Integrity Manifests (RIMs), corresponding to the software image expected to be found on the device, signed by the Reference Value Provider, available to the Verifier (see Section 3.1.3 for "in-band" and "out of band" ways to make this happen).

* In Step 1, the Verifier (Network Management Station), on behalf of a Relying Party, requests Identity, Measurement Values, and possibly RIMs, from the Attester.

* In Step 2, the Attester responds to the request by providing a DevID, quotes (measured values, signed by the Attester), and optionally RIMs.

Use of the following standards components allows for interoperability:

1. TPM Keys MUST be configured according to [Platform-DevID-TPM-2.0], or [Platform-ID-TPM-1.2].

2. For devices using UEFI and Linux, measurements of firmware and bootable modules MUST be taken according to TCG PC Client [PC-Client-EFI-TPM-1.2] or [PC-Client-BIOS-TPM-2.0], and Linux IMA [IMA].

3. Device Identity MUST be managed as specified in IEEE 802.1AR Device Identity certificates [IEEE-802-1AR], with keys protected by TPMs.

4. Attestation logs from Linux-based systems MUST be formatted according to the Canonical Event Log format [Canonical-Event-Log]. UEFI-based systems MUST use the TCG UEFI BIOS event log [PC-Client-EFI-TPM-1.2] for TPM1.2 systems, and TCG PC Client Platform Firmware Profile [PC-Client-BIOS-TPM-2.0] for TPM2.0.

5. Quotes MUST be retrieved from the TPM according to TCG TAP Information Model [TAP] and the CHARRA YANG model [I-D.ietf-rats-yang-tpm-charra]. While the TAP IM gives a protocol-independent description of the data elements involved, it’s important to note that quotes from the TPM are signed inside the TPM, and MUST be retrieved in a way that does not invalidate the signature, to preserve the trust model. The [I-D.ietf-rats-yang-tpm-charra] is used for this purpose. (See Section 5 Security Considerations).

6. Reference Values MUST be encoded as defined in the TCG RIM document [RIM], typically using SWID [SWID], [NIST-IR-8060] or CoSWID tags [I-D.ietf-sacm-coswid].

2.4. RIV Simplifying Assumptions

This document makes the following simplifying assumptions to reduce complexity:

* The product to be attested MUST be shipped by the equipment vendor with both an IEEE 802.1AR Device Identity and an Initial Attestation Key (IAK), with certificates in place. The IAK certificate must contain the same identity information as the DevID (specifically, the same subject and subjectAltName (if used), signed by the manufacturer). The IAK is a type of key that can be used to sign a TPM Quote, but not other objects (i.e., it's marked as a TCG "Restricted" key; this convention is described in "TPM 2.0 Keys for Device Identity and Attestation" [Platform-DevID-TPM-2.0]). For network equipment, which is generally non-privacy-sensitive, shipping a device with both an IDevID and an IAK already provisioned substantially simplifies initial startup.

* IEEE 802.1AR does not require a product serial number as part of the subject, but RIV-compliant devices MUST include their serial numbers in the DevID/IAK certificates to simplify tracking logistics for network equipment users. All other optional 802.1AR fields remain optional in RIV.

  It should be noted that 802.1AR use of X.509 certificate fields is not identical to those descsribed in [RFC6125] for representation of application service identity.

* The product MUST be equipped with a Root of Trust for Measurement (RTM), Root of Trust for Storage and Root of Trust for Reporting (as defined in [SP800-155]) which together are capable of conforming to TCG Trusted Attestation Protocol Information Model [TAP].

* The authorized software supplier MUST make available Reference Values in the form of signed SWID or CoSWID tags.

2.4.1. Reference Integrity Manifests (RIMs)

[I-D.ietf-rats-yang-tpm-charra] focuses on collecting and transmitting evidence in the form of PCR measurements and attestation logs. But the critical part of the process is enabling the Verifier to decide whether the measurements are "the right ones" or not.

While it must be up to network administrators to decide what they want on their networks, the software supplier should supply the Reference Values, in signed Reference Integrity Manifests, that may be used by a Verifier to determine if evidence shows known good, known bad or unknown software configurations.

In general, there are two kinds of reference measurements:

1. Measurements of early system startup (e.g., BIOS, boot loader, OS kernel) are essentially single-threaded, and executed exactly once, in a known sequence, before any results could be reported. In this case, while the method for computing the hash and extending relevant PCRs may be complicated, the net result is that the software (more likely, firmware) vendor will have one known good PCR value that "should" be present in the relevant PCRs after the box has booted. In this case, the signed reference measurement could simply list the expected hashes for the given version. However, a RIM that contains the intermediate hashes can be useful in debugging cases where the expected final hash is not the one reported.

2. Measurements taken later in operation of the system, once an OS has started (for example, Linux IMA [IMA]), may be more complex, with unpredictable "final" PCR values. In this case, the Verifier must have enough information to reconstruct the expected PCR values from logs and signed reference measurements from a trusted authority.

In both cases, the expected values can be expressed as signed SWID or CoSWID tags, but the SWID structure in the second case is somewhat more complex, as reconstruction of the extended hash in a PCR may involve thousands of files and other objects.

TCG has published an information model defining elements of Reference Integrity Manifests under the title TCG Reference Integrity Manifest Information Model [RIM]. This information model outlines how SWID tags should be structured to allow attestation, and defines "bundles" of SWID tags that may be needed to describe a complete software release. The RIM contains metadata relating to the software release it belongs to, plus hashes for each individual file or other object that could be attested.

Many network equipment vendors use a UEFI BIOS to launch their network operating system. These vendors may want to also use the TCG PC Client Reference Integrity Measurement specification [PC-Client-RIM], which focuses specifically on a SWID-compatible format suitable for expressing measurement values expected from a UEFI BIOS.

2.4.2. Attestation Logs

Quotes from a TPM can provide evidence of the state of a device up to the time the evidence was recorded, but to make sense of the quote in cases where several events are extended into one PCR an event log that identifies which software modules contributed which values to the quote during startup must also be provided. When required, the log MUST contain enough information to demonstrate its integrity by allowing exact reconstruction of the digest conveyed in the signed quote (that is, calculating the hash of all the hashes in the log should produce the same values as contained in the PCRs; if they don't match, the log may have been tampered with. See Section 9.1).

There are multiple event log formats which may be supported as viable formats of Evidence between the Attester and Verifier, but to simplify interoperability, RIV focuses on just three:

* TCG UEFI BIOS event log for TPM 2.0 (TCG PC Client Platform Firmware Profile) [PC-Client-BIOS-TPM-2.0]

* TCG UEFI BIOS event log for TPM 1.2 (TCG EFI Platform Specification for TPM Family 1.1 or 1.2, Section 7) [PC-Client-EFI-TPM-1.2]

* TCG Canonical Event Log [Canonical-Event-Log]

3. Standards Components

3.1. Prerequisites for RIV

The Reference Interaction Model for Challenge-Response-based Remote Attestation ([I-D.birkholz-rats-reference-interaction-model]) is based on the standard roles defined in [I-D.ietf-rats-architecture]. However, additional prerequisites have been established to allow for interoperable RIV use case implementations. These prerequisites are intended to provide sufficient context information so that the Verifier can acquire and evaluate measurements collected by the Attester.

3.1.1. Unique Device Identity

A secure Device Identity (DevID) in the form of an IEEE 802.1AR DevID certificate [IEEE-802-1AR] must be provisioned in the Attester's TPMs.

3.1.2. Keys

The Attestation Key (AK) and certificate must also be provisioned on the Attester according to [Platform-DevID-TPM-2.0], or [Platform-ID-TPM-1.2].

It MUST be possible for the Verifier to determine that the Attester's Attestation keys are resident in the same TPM as its DevID keys (see Section 2.2 and Section 5 Security Considerations).

3.1.3. Appraisal Policy for Evidence

As noted in Section 2.3, the Verifier may obtain Reference Values from several sources. In addition, administrators may make authorized, site-specific changes (e.g. keys in key databases) that could impact attestation results. As such, there could be conflicts, omissions or ambiguities between some Reference Values and collected Evidence.

The Verifier MUST have an Appraisal Policy for Evidence to evaluate the significance of any discrepancies between different reference sources, or between reference values and evidence from logs and quotes. While there must be an Appraisal Policy, this document does not specify the format or mechanism to convey the intended policy, nor does RIV specify mechanisms by which the results of applying the policy are communicated to the Relying Party.

3.2. Reference Model for Challenge-Response

Once the prerequisites for RIV are met, a Verifier is able to acquire Evidence from an Attester. The following diagram illustrates a RIV information flow between a Verifier and an Attester, derived from Section 7.1 of [I-D.birkholz-rats-reference-interaction-model]. In this diagram, each event with its input and output parameters is shown as "Event(input-params)=>(outputs)". Event times shown correspond to the time types described within Appendix A of [I-D.ietf-rats-architecture]:

```
   .----------.                              .-----------------------.
   | Attester |                              | Relying Party/Verifier |
   '----------'                              '-----------------------'
     time(VG)                                                         |
   generateClaims(attestingEnvironment)                               |
       | => claims, eventLogs                                         |
       |                                                              |
       |                                                              |
       |                                                         time(NS)
       | <-- requestAttestation(handle, authSecIDs, claimSelection)   |
       |                                                              |
    time(EG)                                                          |
   collectClaims(claims, claimSelection)                              |
       | => collectedClaims                                           |
       |                                                              |
   generateEvidence(handle, authSecIDs, collectedClaims)              |
       | => evidence                                                  |
       |                                                       time(RG,RA)
       | evidence, eventLogs ------------------------------------> |
       |                                                              |
       |                appraiseEvidence(evidence, eventLogs, refValues)
       |                                         attestationResult <= |
       |                                                              |
       ~                                                              ~
       |                                                         time(RX)
```

Figure 4: IETF Attestation Information Flow

* Step 1 (time(VG)): One or more Attesting Network Device PCRs are extended with measurements. RIV provides no direct link between the time at which the event takes place and the time that it's attested, although streaming attestation as in [I-D.birkholz-rats-network-device-subscription] could.

* Step 2 (time(NS)): The Verifier generates a unique random nonce ("number used once"), and makes a request for one or more PCRs from an Attester. For interoperability, this must be accomplished as specified in the YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs [I-D.ietf-rats-yang-tpm-charra]. TPM1.2 and TPM2.0 both allow nonces as large as the operative digest size (i.e., 20 or 32 bytes; see [TPM1.2] Part 2, Section 5.5 and [TPM2.0] Part 2, Section 10.4.4).

* Step 3 (time(EG)): On the Attester, measured values are retrieved from the Attester's TPM. This requested PCR evidence, along with the Verifier's nonce, called a Quote, is signed by the Attestation Key (AK) associated with the DevID. Quotes are retrieved according to CHARRA YANG model [I-D.ietf-rats-yang-tpm-charra].

At the same time, the Attester collects log evidence showing the values have been extended into that PCR. Section 9.1 gives more detail on how this works, including references to the structure and contents of quotes in TPM documents.

* Step 4: Collected Evidence is passed from the Attester to the Verifier

* Step 5 (time(RG,RA)): The Verifier reviews the Evidence and takes action as needed. As the interaction between Relying Party and Verifier is out of scope for RIV, this can be described as one step.

  - If the signature covering TPM Evidence is not correct, the device SHOULD NOT be trusted.

  - If the nonce in the response doesn't match the Verifier's nonce, the response may be a replay, and device SHOULD NOT be trusted.

  - If the signed PCR values do not match the set of log entries which have extended a particular PCR, the device SHOULD NOT be trusted.

  - If the log entries that the Verifier considers important do not match known good values, the device SHOULD NOT be trusted. We note that the process of collecting and analyzing the log can be omitted if the value in the relevant PCR is already a known-good value.

  - If the set of log entries are not seen as acceptable by the Appraisal Policy for Evidence, the device SHOULD NOT be trusted.

  - If time(RG)-time(NS) is greater than the Appraisal Policy for Evidence's threshold for assessing freshness, the Evidence is considered stale and SHOULD NOT be trusted.

3.2.1. Transport and Encoding

Network Management systems may retrieve signed PCR based Evidence using NETCONF or RESTCONF with [I-D.ietf-rats-yang-tpm-charra]. In either case, implementatations must do so using a secure tunnel.

Log Evidence MUST be retrieved via log interfaces specified in [I-D.ietf-rats-yang-tpm-charra].

3.3. Centralized vs Peer-to-Peer

Figure 4 above assumes that the Verifier is trusted, while the Attester is not. In a Peer-to-Peer application such as two routers negotiating a trust relationship, the two peers can each ask the other to prove software integrity. In this application, the information flow is the same, but each side plays a role both as an Attester and a Verifier. Each device issues a challenge, and each device responds to the other's challenge, as shown in Figure 5. Peer-to-peer challenges, particularly if used to establish a trust relationship between routers, require devices to carry their own signed reference measurements (RIMs). Devices may also have to carry Appraisal Policy for Evidence for each possible peer device so that each device has everything needed for remote attestation, without having to resort to a central authority.

```
+---------------+                            +---------------+
| RefVal        |                            | RefVal        |
| Provider A    |                            | Provider B    |
| Firmware      |                            | Firmware      |
| Configuration |                            | Configuration |
| Authority     |                            | Authority     |
|               |                            |               |
+---------------+                            +---------------+
       |                                            |
       |                                            |Step 0B
       |        +------------+        +------------+  |
       |        |            | Step 1 |            |  |    \
       |        | Attester   |<------>| Verifier   |  |    |
       |        |            |<------>|            |  |    |  Router B
       +------>|            | Step 2 |            |  |    |- Challenges
        Step 0A|            |        |            |  |    |  Router A
               |            |------->|            |  |    |
               |- Router A -| Step 3 |- Router B -|  |    /
               |            |        |            |  |
               |            | Step 1 |            |  |    \
               | Verifier   |<------>| Attester   |<-+    |  Router A
               |            |<------>|            |       |- Challenges
               |            | Step 2 |            |       |  Router B
               |            |        |            |       |
               |            |<-------|            |       |
               +------------+ Step 3 +------------+       /
```

Figure 5: Peer-to-Peer Attestation Information Flow

In this application, each device may need to be equipped with signed RIMs to act as an Attester, and also an Appraisal Policy for Evidence and a selection of trusted X.509 root certificates, to allow the device to act as a Verifier. An existing link layer protocol such as 802.1X [IEEE-802.1X] or 802.1AE [IEEE-802.1AE], with Evidence being enclosed over a variant of EAP [RFC3748] or LLDP [LLDP] are suitable methods for such an exchange. Details of peer-to-peer operation are out of scope for this document.

4. Privacy Considerations

Network equipment, such as routers, switches and firewalls, has a key role to play in guarding the privacy of individuals using the network. Network equipment generally adheres to several rules to protect privacy:

* Packets passing through the device must not be sent to unauthorized destinations. For example:

  - Routers often act as Policy Enforcement Points, where individual subscribers may be checked for authorization to access a network. Subscriber login information must not be released to unauthorized parties.

  - Network equipment is often called upon to block access to protected resources from unauthorized users.

* Routing information, such as the identity of a router’s peers, must not be leaked to unauthorized neighbors.

* If configured, encryption and decryption of traffic must be carried out reliably, while protecting keys and credentials.

Functions that protect privacy are implemented as part of each layer of hardware and software that makes up the networking device. In light of these requirements for protecting the privacy of users of the network, the network equipment must identify itself, and its boot configuration and measured device state (for example, PCR values), to the equipment’s administrator, so there’s no uncertainty as to what function each device and configuration is configured to carry out. Attestation is a component that allows the administrator to ensure that the network provides individual and peer privacy guarantees, even though the device itself may not have a right to keep its identity secret.

See [NetEq] for more context on privacy in networking devices.

While attestation information from network devices is not likely to contain privacy-sensitive content regarding network users, administrators may want to keep attestation records confidential to avoid disclosing versions of software loaded on the device, information which could facilitate attacks against known vulnerabilities.

5. Security Considerations

Specifications such as [RFC8446] (TLS) and [RFC7950] (YANG) contain considerable advice on keeping network-connected systems secure. This section outlines specific risks and mitigations related to attestation.

Attestation Evidence obtained by the RIV procedure is subject to a number of attacks:

* Keys may be compromised.

* A counterfeit device may attempt to impersonate (spoof) a known authentic device.

* Person-in-the-middle attacks may be used by a compromised device to attempt to deliver responses that originate in an authentic device.

* Replay attacks may be attempted by a compromised device.

5.1. Keys Used in RIV

Trustworthiness of RIV attestation depends strongly on the validity of keys used for identity and attestation reports. RIV takes full advantage of TPM capabilities to ensure that evidence can be trusted.

Two sets of key-pairs are relevant to RIV attestation:

* A DevID key-pair is used to certify the identity of the device in which the TPM is installed.

* An Attestation Key-pair (AK) key is used to certify attestation Evidence (called 'quotes' in TCG documents), used to provide evidence for integrity of the software on the device

TPM practices usually require that these keys be different, as a way of ensuring that a general-purpose signing key cannot be used to spoof an attestation quote.

In each case, the private half of the key is known only to the TPM, and cannot be retrieved externally, even by a trusted party. To ensure that's the case, specification-compliant private/public key-pairs are generated inside the TPM, where they are never exposed, and cannot be extracted (See [Platform-DevID-TPM-2.0]).

Keeping keys safe is a critical enabler of trustworthiness, but it's just part of attestation security; knowing which keys are bound to the device in question is just as important in an environment where private keys are never exposed.

While there are many ways to manage keys in a TPM (see [Platform-DevID-TPM-2.0]), RIV includes support for "zero touch" provisioning (also known as zero-touch onboarding) of fielded devices (e.g., Secure ZTP, [RFC8572]), where keys which have predictable trust properties are provisioned by the device vendor.

Device identity in RIV is based on IEEE 802.1AR Device Identity (DevID). This specification provides several elements:

* A DevID requires a unique key pair for each device, accompanied by an X.509 certificate,

* The private portion of the DevID key is to be stored in the device, in a manner that provides confidentiality (Section 6.2.5 [IEEE-802-1AR])

The X.509 certificate contains several components:

* The public part of the unique DevID key assigned to that device allows a challenge of identity.

* An identifying string that's unique to the manufacturer of the device. This is normally the serial number of the unit, which might also be printed on a label on the device.

* The certificate must be signed by a key traceable to the manufacturer's root key.

With these elements, the device's manufacturer and serial number can be identified by analyzing the DevID certificate plus the chain of intermediate certificates leading back to the manufacturer's root certificate. As is conventional in TLS or SSH connections, a random nonce must be signed by the device in response to a challenge, proving possession of its DevID private key.

RIV uses the DevID to validate a TLS or SSH connection to the device as the attestation session begins. Security of this process derives from TLS or SSH security, with the DevID, containing a device serial number, providing proof that the session terminates on the intended device. See [RFC8446], [RFC4253].

Evidence of software integrity is delivered in the form of a quote signed by the TPM itself, accompanied by an IAK certificate containing the same identity information as the DevID. Because the contents of the quote are signed inside the TPM, any external modification (including reformatting to a different data format) after measurements have been taken will be detected as tampering. An unbroken chain of trust is essential to ensuring that blocks of code that are taking measurements have been verified before execution (see Figure 1).

Requiring measurements of the operating software to be signed by a key known only to the TPM also removes the need to trust the device's operating software (beyond the first measurement in the RTM; see below); any changes to the quote, generated and signed by the TPM itself, made by malicious device software, or in the path back to the Verifier, will invalidate the signature on the quote.

A critical feature of the YANG model described in [I-D.ietf-rats-yang-tpm-charra] is the ability to carry TPM data structures in their TCG-defined format, without requiring any changes to the structures as they were signed and delivered by the TPM. While alternate methods of conveying TPM quotes could compress out redundant information, or add another layer of signing using external keys, the implementation MUST preserve the TPM signing, so that tampering anywhere in the path between the TPM itself and the Verifier can be detected.

5.2. Prevention of Spoofing and Person-in-the-Middle Attacks

Prevention of spoofing attacks against attestation systems is also important. There are several cases to consider:

* The entire device could be spoofed. If the Verifier goes to appraise a specific Attester, it might be redirected to a different Attester.

* A compromised device could have a valid DevID, but substitute a quote from a knonwn-good device, instead of returning its own, as described in [RFC6813].

* A device with a compromised OS could return a fabricated quote providing spoofed attestation Evidence.

Use of the 802.1AR Device Identity (DevID) in the TPM provides protection against the case of a spoofed device, by ensuring that the Verifier's TLS or SSH session is in fact terminating on the right device.

Protection against spoofed quotes from a device with valid identity is a bit more complex. An identity key must be available to sign any kind of nonce or hash offered by the Verifier, and consequently, could be used to sign a fabricated quote. To block a spoofed Attestation Result, the quote generated inside the TPM must be signed by a key that's different from the DevID, called an Attestation Key (AK).

Given separate Attestation and DevID keys, the binding between the AK and the same device must also be proven to prevent a person-in-the-middle attack (e.g., the 'Asokan Attack' [RFC6813]).

This is accomplished in RIV through use of an AK certificate with the same elements as the DevID (same manufacturer's serial number, signed by the same manufacturer's key), but containing the device's unique AK public key instead of the DevID public key. This binding between DevID and AK certificates is critical to reliable attestation.

The TCG document TPM 2.0 Keys for Device Identity and Attestation [Platform-DevID-TPM-2.0] specifies OIDs for Attestation Certificates that allow the CA to mark a key as specifically known to be an Attestation key.

These two key-pairs and certificates are used together:

* The DevID is used to validate a TLS connection terminating on the device with a known serial number.

* The AK is used to sign attestation quotes, providing proof that the attestation evidence comes from the same device.

5.3. Replay Attacks

Replay attacks, where results of a previous attestation are submitted in response to subsequent requests, are usually prevented by inclusion of a random nonce in the request to the TPM for a quote. Each request from the Verifier includes a new random number (a nonce). The resulting quote signed by the TPM contains the same nonce, allowing the Verifier to determine freshness, (i.e., that the resulting quote was generated in response to the Verifier's specific request). Time-Based Uni-directional Attestation [I-D.birkholz-rats-tuda] provides an alternate mechanism to verify freshness without requiring a request/response cycle.

5.4. Owner-Signed Keys

Although device manufacturers must pre-provision devices with easily verified DevID and AK certificates if zero-touch provisioning such as described in [RFC8572] is to be supported, use of those credentials is not mandatory. IEEE 802.1AR incorporates the idea of an Initial Device ID (IDevID), provisioned by the manufacturer, and a Local Device ID (LDevID) provisioned by the owner of the device. RIV and [Platform-DevID-TPM-2.0] extends that concept by defining an Initial Attestation Key (IAK) and Local Attestation Key (LAK) with the same properties.

Device owners can use any method to provision the Local credentials.

* TCG document [Platform-DevID-TPM-2.0] shows how the initial Attestation keys can be used to certify LDevID and LAK keys. Use of the LDevID and LAK allows the device owner to use a uniform identity structure across device types from multiple manufacturers (in the same way that an "Asset Tag" is used by many enterprises to identify devices they own). TCG document [Provisioning-TPM-2.0] also contains guidance on provisioning Local identity keys in TPM 2.0. Owners should follow the same practice of binding Local DevID and Local AK as the manufacturer would for IDevID and IAK. See Section Section 2.2.

* Device owners, however, can use any other mechanism they want to assure themselves that local identity certificates are inserted into the intended device, including physical inspection and programming in a secure location, if they prefer to avoid placing trust in the manufacturer-provided keys.

Clearly, local keys can’t be used for secure Zero Touch provisioning; installation of the local keys can only be done by some process that runs before the device is installed for network operation, or using procedures such as those outlined in Bootstrapping Remote Secure Key Infrastructure (BRSKI) [RFC8995].

On the other end of the device life cycle, provision should be made to wipe local keys when a device is decommissioned, to indicate that the device is no longer owned by the enterprise. The manufacturer’s Initial identity keys must be preserved, as they contain no information that’s not already printed on the device’s serial number plate.

5.5. Other Factors for Trustworthy Operation

In addition to trustworthy provisioning of keys, RIV depends on a number of other factors for trustworthy operation.

* Secure identity depends on mechanisms to prevent per-device secret keys from being compromised. The TPM provides this capability as a Root of Trust for Storage.

* Attestation depends on an unbroken chain of measurements, starting from the very first measurement. See Section 9.1 for background on TPM practices.

* That first measurement is made by code called the Root of Trust for Measurement, typically done by trusted firmware stored in boot flash. Mechanisms for maintaining the trustworthiness of the RTM are out of scope for RIV, but could include immutable firmware, signed updates, or a vendor-specific hardware verification technique. See Section 9.2 for background on roots of trust.

* The device owner SHOULD provide some level of physical defense for the device. If a TPM that has already been programmed with an authentic DevID is stolen and inserted into a counterfeit device, attestation of that counterfeit device may become indistinguishable from an authentic device.

RIV also depends on reliable Reference Values, as expressed by the RIM [RIM]. The definition of trust procedures for RIMs is out of scope for RIV, and the device owner is free to use any policy to validate a set of reference measurements. It should also be noted that, while RIV can provide a reliable indication that a known software package is in use by the device, and that the package has not been tampered, it is the device owner's responsibility to determine that it's the correct package for the application.

RIMs may be conveyed out-of-band or in-band, as part of the attestation process (see Section 3.1.3). But for network devices, where software is usually shipped as a self-contained package, RIMs signed by the manufacturer and delivered in-band may be more convenient for the device owner.

The validity of RIV attestation results is also influenced by procedures used to create Reference Values:

* While the RIM itself is signed, supply-chains SHOULD be carefully scrutinized to ensure that the values are not subject to unexpected manipulation prior to signing. Insider-attacks against code bases and build chains are particularly hard to spot.

* Designers SHOULD guard against hash collision attacks. Reference Integrity Manifests often give hashes for large objects of indeterminate size; if one of the measured objects can be replaced with an implant engineered to produce the same hash, RIV will be

unable to detect the substitution. TPM1.2 uses SHA-1 hashes only, which have been shown to be susceptible to collision attack. TPM2.0 will produce quotes with SHA-256, which so far has resisted such attacks. Consequently, RIV implementations SHOULD use TPM2.0.

6. IANA Considerations

This document has no IANA actions.

7. Conclusion

TCG technologies can play an important part in the implementation of Remote Integrity Verification. Standards for many of the components needed for implementation of RIV already exist:

* Platform identity can be based on IEEE 802.1AR Device Identity, coupled with careful supply-chain management by the manufacturer.

* Complex supply chains can be certified using TCG Platform Certificates [Platform-Certificates].

* The TCG TAP mechanism coupled with [I-D.ietf-rats-yang-tpm-charra] can be used to retrieve attestation evidence.

* Reference Values must be conveyed from the software authority (e.g., the manufacturer) in Reference Integrity Manifests, to the system in which verification will take place. IETF and TCG SWID and CoSWID work ([I-D.ietf-sacm-coswid], [RIM]) forms the basis for this function.

8. Acknowledgements

The authors wish to thank numerous reviewers for generous assistance, including William Bellingrath, Mark Baushke, Ned Smith, Henk Birkholz, Tom Laffey, Dave Thaler, Wei Pan, Michael Eckel, Thomas Hardjono, Bill Sulzen, Willard (Monty) Wiseman, Kathleen Moriarty, Nancy Cam-Winget and Shwetha Bhandari

9. Appendix

9.1. Using a TPM for Attestation

The Trusted Platform Module and surrounding ecosystem provide three interlocking capabilities to enable secure collection of evidence from a remote device, Platform Configuration Registers (PCRs), a Quote mechanism, and a standardized Event Log.

Each TPM has at least eight and at most twenty-four PCRs (depending on the profile and vendor choices), each one large enough to hold one hash value (SHA-1, SHA-256, and other hash algorithms can be used, depending on TPM version). PCRs can't be accessed directly from outside the chip, but the TPM interface provides a way to "extend" a new security measurement hash into any PCR, a process by which the existing value in the PCR is hashed with the new security measurement hash, and the result placed back into the same PCR. The result is a composite fingerprint comprising the hash of all the security measurements extended into each PCR since the system was reset.

Every time a PCR is extended, an entry should be added to the corresponding Event Log. Logs contain the security measurement hash plus informative fields offering hints as to which event generated the security measurement. The Event Log itself is protected against accidental manipulation, but it is implicitly tamper-evident - any verification process can read the security measurement hash from the log events, compute the composite value and compare that to what ended up in the PCR. If there's no discrepancy, the logs do provide an accurate view of what was placed into the PCR.

Note that the composite hash-of-hashes recorded in PCRs is order-dependent, resulting in different PCR values for different ordering of the same set of events (e.g. Event A followed by Event B yields a different PCR value than B followed by A). For single-threaded code, where both the events and their order are fixed, a Verifier may validate a single PCR value, and use the log only to diagnose a mismatch from Reference Values. However, operating system code is usually non-deterministic, meaning that there may never be a single "known good" PCR value. In this case, the Verifier may have to verify that the log is correct, and then analyze each item in the log to determine if it represents an authorized event.

In a conventional TPM Attestation environment, the first measurement must be made and extended into the TPM by trusted device code (called the Root of Trust for Measurement, RTM). That first measurement should cover the segment of code that is run immediately after the RTM, which then measures the next code segment before running it, and so on, forming an unbroken chain of trust. See [TCGRoT] for more on Mutable vs Immutable roots of trust.

The TPM provides another mechanism called a Quote that can read the current value of the PCRs and package them, along with the Verifier's nonce, into a TPM-specific data structure signed by an Attestation private key, known only to the TPM.

As noted above in Section 5 Security Considerations, it's important to note that the Quote data structure is signed inside the TPM. The trust model is preserved by retrieving the Quote in a way that does not invalidate the signature, as specified in [I-D.ietf-rats-yang-tpm-charra]. The structure of the command and response for a quote, including its signature, as generated by the TPM, can be seen in [TPM1.2] Part 3, Section 16.5, and [TPM2.0] Section 18.4.2.

The Verifier uses the Quote and Log together. The Quote contains the composite hash of the complete sequence of security measurement hashes, signed by the TPM's private Attestation Key. The Log contains a record of each measurement extended into the TPM's PCRs. By computing the composite hash of all the measurements, the Verifier can verify the integrity of the Event Log, even though the Event Log itself is not signed. Each hash in the validated Event Log can then be compared to corresponding expected values in the set of Reference Values to validate overall system integrity.

A summary of information exchanged in obtaining quotes from TPM1.2 and TPM2.0 can be found in [TAP], Section 4. Detailed information about PCRs and Quote data structures can be found in [TPM1.2], [TPM2.0]. Recommended log formats include [PC-Client-BIOS-TPM-2.0], and [Canonical-Event-Log].

9.2. Root of Trust for Measurement

The measurements needed for attestation require that the device being attested is equipped with a Root of Trust for Measurement, that is, some trustworthy mechanism that can compute the first measurement in the chain of trust required to attest that each stage of system startup is verified, a Root of Trust for Storage (i.e., the TPM PCRs) to record the results, and a Root of Trust for Reporting to report the results.

While there are many complex aspects of Roots of Trust ([TCGRoT], [SP800-155], [SP800-193]), two aspects that are important in the case of attestation are:

* The first measurement computed by the Root of Trust for Measurement, and stored in the TPM's Root of Trust for Storage, must be assumed to be correct.

* There must not be a way to reset the Root of Trust for Storage without re-entering the Root of Trust for Measurement code.

The first measurement must be computed by code that is implicitly trusted; if that first measurement can be subverted, none of the remaining measurements can be trusted. (See [SP800-155])

It's important to note that the trustworthiness of the RTM code cannot be assured by the TPM or TPM supplier - code or procedures external to the TPM must guarantee the security of the RTM.

9.3. Layering Model for Network Equipment Attester and Verifier

Retrieval of identity and attestation state uses one protocol stack, while retrieval of Reference Values uses a different set of protocols. Figure 5 shows the components involved.

```
   +-----------------------+              +-------------------------+
   |                       |              |                         |
   |       Attester        |<-------------|        Verifier         |
   |       (Device)        |------------->|   (Management Station)  |
   |                       |      |       |                         |
   +-----------------------+      |       +-------------------------+
                                  |
             -------------------- --------------------
             |                                      |
   -------------------------------    ---------------------------------
   |Reference Values             |    |         Attestation           |
   -------------------------------    ---------------------------------

   ********************************************************************
   *         IETF Attestation Reference Interaction Diagram           *
   ********************************************************************

       .......................          .......................
       . Reference Integrity   .          .  TAP (PTS2.0) Info    .
       .      Manifest         .          . Model and Canonical   .
       .                       .          .     Log Format        .
       .......................          .......................

       *************************          *************************
       * YANG SWID Module      *          * YANG Attestation      *
       * I-D.ietf-sacm-coswid  *          * Module                *
       *                       *          * I-D.ietf-rats-        *
       *                       *          * yang-tpm-charra       *
       *************************          *************************

       *************************          *************************
       * XML, JSON, CBOR (etc) *          * XML, JSON, CBOR (etc) *
       *************************          *************************

       *************************          *************************
       *   RESTCONF/NETCONF    *          *   RESTCONF/NETCONF    *
       *************************          *************************

       *************************          *************************
       *       TLS, SSH        *          *       TLS, SSH        *
       *************************          *************************
```

Figure 6: RIV Protocol Stacks

IETF documents are captured in boxes surrounded by asterisks. TCG documents are shown in boxes surrounded by dots.

9.4. Implementation Notes

Figure 7 summarizes many of the actions needed to complete an Attestation system, with links to relevant documents. While documents are controlled by several standards organizations, the implied actions required for implementation are all the responsibility of the manufacturer of the device, unless otherwise noted.

As noted, SWID tags can be generated many ways, but one possible tool is [SWID-Gen]

Component	Controlling Specification
Make a Secure execution environment o Attestation depends on a secure root of trust for measurement outside the TPM, as well as roots for storage and reporting inside the TPM. o Refer to TCG Root of Trust for Measurement. o NIST SP 800-193 also provides guidelines on Roots of Trust	TCG RoT UEFI.org
Provision the TPM as described in TCG documents.	[Platform-DevID-TPM-2.0] TCG Platform Certificate
Put a DevID or Platform Cert in the TPM o Install an Initial Attestation Key at the same time so that Attestation can work out of the box o Equipment suppliers and owners may want to implement Local Device ID as well as Initial Device ID	TCG TPM DevID TCG Platform Certificate IEEE 802.1AR
Connect the TPM to the TLS stack o Use the DevID in the TPM to authenticate TAP connections, identifying the device	Vendor TLS stack (This action is configuring TLS to use the DevID as its client certificate)
Make CoSWID tags for BIOS/Loader/Kernel objects o Add reference measurements into SWID tags o Manufacturer should sign the SWID tags	IETF CoSWID ISO/IEC 19770-2 NIST IR 8060

```
   |    o  The TCG RIM-IM identifies further          |                 |
   |       procedures to create signed RIM            |                 |
   |       documents that provide the necessary       |                 |
   |       reference information                      |                 |
   ---------------------------------------------------------------------
   |  Package the SWID tags with a vendor software    | Retrieve tags   |
   |  release                                         | with            |
   |    o  A tag-generator plugin such          | I-D.ietf-sacm-coswid|
   |     as [SWID-Gen] can be used                    |----------------|
   |                                                  | TCG PC Client   |
   |                                                  | RIM             |
   ---------------------------------------------------------------------
   |  Use PC Client measurement definitions           | TCG PC Client   |
   |  to define the use of PCRs                       | BIOS            |
   |  (although Windows  OS is rare on Networking     |                 |
   |  Equipment, UEFI BIOS is not)                    |                 |
   ---------------------------------------------------------------------
   |  Use TAP to retrieve measurements                |                 |
   |    o  Map to YANG                                | YANG Module for |
   |  Use Canonical Log Format                        |   Basic         |
   |                                                  |   Attestation   |
   |                                                  | TCG Canonical   |
   |                                                  |   Log Format    |
   ---------------------------------------------------------------------
   | Posture Collection Server (as described in IETF |                 |
   |  SACMs ECP) should request the                   |                 |
   |  attestation and analyze the result              |                 |
   | The Management application might be broken down |                 |
   |  to several more components:                     |                 |
   |    o  A Posture Manager Server                   |                 |
   |       which collects reports and stores them in |                 |
   |       a database                                 |                 |
   |    o  One or more Analyzers that can look at the|                 |
   |       results and figure out what it means.      |                 |
   ---------------------------------------------------------------------
```

Figure 7: Component Status

10. References

10.1. Normative References

[Canonical-Event-Log]
Trusted Computing Group, "Canonical Event Log Format Version 1.0 Revision .41, February 25, 2022", December 2020, <https://trustedcomputinggroup.org/resource/canonical-event-log-format/>.

[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-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-18, 20 March 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-yang-tpm-charra-18.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-21, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-21.txt>.

[IEEE-802-1AR]
Seaman, M., "802.1AR-2018 - IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity, IEEE Computer Society", August 2018.

[IMA] dsafford, kds_etu, mzohar, reinersailer, and serge_hallyn, "Integrity Measurement Architecture", June 2019, <https://sourceforge.net/p/linux-ima/wiki/Home/>.

[PC-Client-BIOS-TPM-2.0]
Trusted Computing Group, "PC Client Specific Platform Firmware Profile Specification Family "2.0", Level 00 Revision 1.05 Revision 23, May 7, 2021", May 2021, <https://trustedcomputinggroup.org/resource/pc-client-specific-platform-firmware-profile-specification/>.

[PC-Client-EFI-TPM-1.2]
Trusted Computing Group, "TCG EFI Platform Specification for TPM Family 1.1 or 1.2, Specification Version 1.22, Revision 15", January 2014, <https://trustedcomputinggroup.org/resource/tcg-efi-platform-specification/>.

[PC-Client-RIM]
Trusted Computing Group, "TCG PC Client Reference Integrity Manifest Specification, v1.04, Nov 4, 2020", December 2019, <https://trustedcomputinggroup.org/resource/tcg-pc-client-reference-integrity-manifest-specification/>.

[Platform-DevID-TPM-2.0]
Trusted Computing Group, "TPM 2.0 Keys for Device Identity and Attestation, Specification Version 1.0, Revision 2", September 2020, <https://trustedcomputinggroup.org/resource/tpm-2-0-keys-for-device-identity-and-attestation/>.

[Platform-ID-TPM-1.2]
Trusted Computing Group, "TPM Keys for Platform Identity for TPM 1.2, Specification Version 1.0, Revision 3", August 2015, <https://trustedcomputinggroup.org/resource/tpm-keys-for-platform-identity-for-tpm-1-2-2/>.

[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>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[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>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[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>.

[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>.

[RIM] Trusted Computing Group, "TCG Reference Integrity Manifest (RIM) Information Model, v1.0, Revision 0.16, Nov 12, 2020", June 2019, <https://trustedcomputinggroup.org/resource/tcg-reference-integrity-manifest-rim-information-model/>.

[SWID] The International Organization for Standardization/ International Electrotechnical Commission, "Information Technology Software Asset Management Part 2: Software Identification Tag, ISO/IEC 19770-2", October 2015, <https://www.iso.org/standard/65666.html>.

[TAP] Trusted Computing Group, "TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0, Version 1.0, Revision 0.36", October 2018, <https://trustedcomputinggroup.org/resource/tcg-tap-information-model/>.

10.2. Informative References

[AK-Enrollment]
Trusted Computing Group, "TCG Infrastructure Working Group - A CMC Profile for AIK Certificate Enrollment Version 1.0, Revision 7", March 2011, <https://trustedcomputinggroup.org/resource/tcg-infrastructure-working-group-a-cmc-profile-for-aik-certificate-enrollment/>.

[I-D.birkholz-rats-network-device-subscription]
Birkholz, H., Voit, E., and W. Pan, "Attestation Event Stream Subscription", Work in Progress, Internet-Draft, draft-birkholz-rats-network-device-subscription-03, 17 August 2021, <https://www.ietf.org/archive/id/draft-birkholz-rats-network-device-subscription-03.txt>.

[I-D.birkholz-rats-reference-interaction-model]
Birkholz, H., Eckel, M., Newton, C., and L. Chen, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-birkholz-rats-reference-interaction-model-03, 7 July 2020, <https://www.ietf.org/archive/id/draft-birkholz-rats-reference-interaction-model-03.txt>.

[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>.

[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-12, 24 February 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-eat-12.txt>.

[I-D.richardson-rats-usecases]
Richardson, M., Wallace, C., and W. Pan, "Use cases for Remote Attestation common encodings", Work in Progress, Internet-Draft, draft-richardson-rats-usecases-08, 2 November 2020, <https://www.ietf.org/archive/id/draft-richardson-rats-usecases-08.txt>.

[IEEE-802.1AE]
Seaman, M., "802.1AE MAC Security (MACsec)", 2018, <https://1.ieee802.org/security/802-1ae/>.

[IEEE-802.1X]
IEEE Computer Society, "802.1X-2020 - IEEE Standard for Local and Metropolitan Area Networks--Port-Based Network Access Control", February 2020, <https://standards.ieee.org/standard/802_1X-2020.html>.

[LLDP] IEEE Computer Society, "802.1AB-2016 - IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery", March 2016, <https://standards.ieee.org/standard/802_1AB-2016.html>.

[NetEq] Trusted Computing Group, "TCG Guidance for Securing Network Equipment, Version 1.0, Revision 29", January 2018, <https://trustedcomputinggroup.org/resource/tcg-guidance-securing-network-equipment/>.

[NIST-IR-8060]
National Institute for Standards and Technology, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags", April 2016, <https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf>.

[Platform-Certificates]
Trusted Computing Group, "TCG Platform Attribute Credential Profile, Specification Version 1.0, Revision 16", January 2018, <https://trustedcomputinggroup.org/resource/tcg-platform-attribute-credential-profile/>.

[Provisioning-TPM-2.0]
Trusted Computing Group, "TCG TPM v2.0 Provisioning Guidance, Version 1.0, Revision 1.0", March 2015, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-TPM-v2.0-Provisioning-Guidance-Published-v1r1.pdf>.

[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>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6813] Salowey, J. and S. Hanna, "The Network Endpoint Assessment (NEA) Asokan Attack Analysis", RFC 6813, DOI 10.17487/RFC6813, December 2012, <https://www.rfc-editor.org/info/rfc6813>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <https://www.rfc-editor.org/info/rfc8572>.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <https://www.rfc-editor.org/info/rfc8995>.

[SP800-155]
National Institute of Standards and Technology, "BIOS Integrity Measurement Guidelines (Draft)", December 2011, <https://csrc.nist.gov/csrc/media/publications/sp/800-155/draft/documents/draft-sp800-155_dec2011.pdf>.

[SP800-193]
National Institute for Standards and Technology, "NIST Special Publication 800-193: Platform Firmware Resiliency Guidelines", April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-193.pdf>.

[SWID-Gen] Labs64, Munich, Germany, "SoftWare IDentification (SWID) Tags Generator (Maven Plugin)", n.d., <https://github.com/Labs64/swid-maven-plugin>.

[TCGRoT] Trusted Computing Group, "DRAFT: TCG Roots of Trust Specification", October 2018, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_Roots_of_Trust_Specification_v0p20_PUBLIC_REVIEW.pdf>.

[TPM1.2] Trusted Computing Group, "TPM Main Specification Level 2 Version 1.2, Revision 116", March 2011, <https://trustedcomputinggroup.org/resource/tpm-main-specification/>.

[TPM2.0] Trusted Computing Group, "Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59", November 2019, <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

Authors' Addresses

Guy Fedorkow (editor)
Juniper Networks, Inc.
10 Technology Park Drive
Westford, Massachusetts 01886
United States of America
Email: gfedorkow@juniper.net

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Jessica Fitzgerald-McKay
National Security Agency
9800 Savage Road
Ft. Meade, Maryland 20755
United States of America
Email: jmfitz2@nsa.gov

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 September 2024

H. Birkholz
Fraunhofer SIT
J. O'Donoghue
Qualcomm Technologies Inc.
N. Cam-Winget
Cisco Systems
C. Bormann
Universität Bremen TZI
4 March 2024

A CBOR Tag for Unprotected CWT Claims Sets
draft-ietf-rats-uccs-09

Abstract

When transported over secure channels, CBOR Web Token (CWT, RFC 8392) Claims Sets may 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.

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/. Subscribe at https://www.ietf.org/mailman/listinfo/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

Copyright Notice

Table of Contents

1. Introduction

A CBOR Web Token (CWT) as specified by [RFC8392] is always wrapped in a CBOR Object Signing and Encryption (COSE, [RFC9052]) envelope. COSE provides -- amongst other things -- end-to-end data origin authentication and integrity protection employed by RFC 8392 as well as 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 effectively turns the channel into a mutually secured channel.

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 [RFC9334]) for the conveyance of RATS Conceptual Messages.

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 secure 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, Evidence, Relying Party and Verifier are used as in [RFC9334].

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: [NIST-SP800-90Ar1] defines a Secure Channel as follows:

> "A path for transferring data between two entities or components that ensures confidentiality, integrity and replay protection, as well as mutual authentication between the entities or components. The secure channel may be provided using approved cryptographic, physical or procedural methods, or a combination thereof"

For the purposes of the present document, we focus on a protected communication channel used for conveyance that can ensure the same qualities as CWT without having the COSE protection available: mutual authentication, integrity protection, confidentiality. (Replay protection can be added by including a nonce claim such as Nonce (claim 10 [IANA.cwt]).) Examples include conveyance via PCIe (Peripheral Component Interconnect Express) IDE (Integrity and Data Encryption), or a TLS tunnel.

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. Deployment and Usage of UCCS

Usage scenarios involving the conveyance of Claims, in particular RATS, require a standardized data definition and encoding format that can be transferred and transported using different communication channels. As these are Claims, the Claims sets defined in [RFC8392] are 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 [RFC9397]) or a Trusted Platform Module (TPM, see [TPM2]). 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 elaborate on Secure

Channel characteristics in general and further describe RATS usage scenarios 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. (Where the handshake did not provide a mutually secure channel, further authentication information can be conveyed by the party not yet authenticated, leading to a mutually secured 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 RATS Secure Channels, the following section 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.

4. UCCS in RATS Conceptual Message Conveyance

This section describes a detailed usage scenario for UCCS in the context of RATS in conjunction with its attendant security requirements. The use of UCCS tag CPA601 outside of the RATS context MUST come with additional instruction leaflets and security considerations.

For the purposes of this section, any RATS role can be the sender or the receiver 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.

In the scope of RATS Claims, the receiver MUST authenticate the sender as part of the establishment of the Secure Channel. Furthermore, the channel MUST provide integrity of the communication between the communicating RATS roles. For data confidentiality [RFC4949], the receiving side MUST be authenticated as well; this is achieved if the sender and receiver mutually authenticate when establishing the Secure Channel. The quality of the receiver's authentication and authorization will influence whether the sender can disclose the UCCS.

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. The assurance provided to a Relying Party depends on the authenticity and integrity properties of the Secure Channel used for conveying the UCCS to it.

Ultimately, it is up to the receiver's policy to determine whether to accept a UCCS from the sender 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 end of the Secure Channel. However, the policy controls and definitions are out of scope for this document.

Where an Attesting Environment serves as an endpoint of a Secure Channel used to convey a UCCS, the security assurance required of that Attesting Environment by a Relying Party generally calls for the Attesting Environment to be implemented using techniques designed to provide enhanced protection from an attacker wishing to tamper with or forge UCCS originating from that Attesting Environment. A possible approach might be to implement the Attesting Environment in a hardened environment such as a TEE [RFC9397] or a TPM [TPM2].

When UCCS emerge from the Secure Channel and into the receiver, the security properties of the secure channel no longer protect the UCCS, which now are subject to the same security properties as any other unprotected data in the Verifier environment. If the receiver subsequently forwards UCCS, they are treated as though they originated within the receiver.

The Secure Channel context does not govern fully formed CWTs in the same way it governs UCCS. As with EATs nested in other EATs (Section 4.2.18.3 (Nested Tokens) 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 (or a nested EAT) 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.)

5. Considerations for Using UCCS in Other RATS Contexts

This section discusses two additional usage scenarios for UCCS in the context of RATS.

5.1. Delegated Attestation

Another usage scenario is that of a sub-Attester that has no signing keys (for example, to keep the implementation complexity to a minimum) and has a Secure Channel, such as local inter-process communication, to interact with a lead Attester (see Composite Device, Section 3.3 of [RFC9334]). The sub-Attester produces a UCCS with the required CWT Claims Set and sends the UCCS through the Secure Channel to the lead Attester. The lead Attester then computes a cryptographic hash of the UCCS and protects that hash using its signing key for Evidence, for example, using a Detached-Submodule-Digest or Detached EAT Bundle (Section 5 of [I-D.ietf-rats-eat]).

5.2. Privacy Preservation

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, when a privacy preserving Secure Channel is employed for conveying a conceptual message the receiver cannot correlate the message with the senders of other received UCCS messages.

An 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 Claim Section 4.2.1 of [I-D.ietf-rats-eat] 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 user device belonging to a child).

6. IANA Considerations

In the CBOR Tags registry [IANA.cbor-tags] as defined in Section 9.2 of [RFC8949], IANA is requested to allocate the tag in Table 1 from the Specification Required space (1+2 size), with the present document as the specification reference.

Tag	Data Item	Semantics
CPA601	map (Claims-Set as per Appendix A of [RFCthis])	Unprotected CWT Claims Set [RFCthis]

Table 1: Values for Tags

// RFC-Editor: This document uses the CPA (code point allocation) // convention described in [I-D.bormann-cbor-draft-numbers]. For // each usage of the term "CPA", please remove the prefix "CPA" from // the indicated value and replace the residue with the value // assigned by IANA; perform an analogous substitution for all other // occurrences of the prefix "CPA" in the document. Finally, please // remove this note.

7. Security Considerations

The security considerations of [RFC8949] apply. The security considerations of [RFC8392] need to be applied analogously, replacing the function of COSE with that of the Secure 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 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. In order to be able to use the UCCS CBOR tag in another such scope, the secure channel and/or the application protocol (e.g., TLS and the protocol identified by ALPN) MUST specify the roles of the endpoints in a fashion that the security properties of conveying UCCS via a Secure Channel between the roles are well-defined.

7.1. 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 [RFC9052] 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.

* Evidence in a UCCS conveyed in a Secure Channel generally cannot be used to support trust in the credentials that were used to establish that secure channel, as this would create a circular dependency.

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.

* Using cryptographic algorithms appropriately.

* Using key material 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.

The remaining subsections of this section highlight some aspects of specific cryptography choices that are detailed further in [RFC9053].

7.2. 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 [RFC9053] contains a detailed explanation of these considerations.

7.3. AES-GCM

* The key and nonce pair is 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 [RFC9053] contains a detailed explanation of these considerations.

7.4. AES-CCM

* The key and nonce pair is 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 [RFC9053] contains a detailed explanation of these considerations.

7.5. 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 [RFC9053] contains a detailed explanation of these considerations.

8. References

8.1. Normative References

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

[IANA.cwt] IANA, "CBOR Web Token (CWT) Claims", <https://www.iana.org/assignments/cwt>.

[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>.

[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>.

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

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[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/rfc/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/rfc/rfc8949>.

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

8.2. Informative References

[I-D.ietf-rats-eat]
Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-25, 15 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-25>.

[NIST-SP800-90Ar1]
Barker, E. and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", National Institute of Standards and Technology, DOI 10.6028/nist.sp.800-90ar1, June 2015, <https://doi.org/10.6028/nist.sp.800-90ar1>.

[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>.

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

[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/rfc/rfc8747>.

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

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

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

[RFC9397] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", RFC 9397, DOI 10.17487/RFC9397, July 2023, <https://www.rfc-editor.org/rfc/rfc9397>.

[TPM2] "Trusted Platform Module Library Specification, Family 2.0, Level 00, Revision 01.59 ed., Trusted Computing Group", 2019.

Appendix A. CDDL

The Concise Data Definition Language (CDDL), as defined in [RFC8610] and [RFC9165], provides an easy and unambiguous way to express structures for protocol messages and data formats that use CBOR or JSON.

[RFC8392] does not define CDDL for CWT Claims Sets.

```
// RFC-Editor: This document uses the CPA (code point allocation)
// convention described in [I-D.bormann-cbor-draft-numbers].  Please
// replace the number 601 in the code blocks below by the value that
// has been assigned for CPA601 and remove this note.
```

This specification proposes using the definitions in Figure 1 for the CWT 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.

```
UCCS-Untagged = Claims-Set
UCCS-Tagged = #6.601(UCCS-Untagged)

Claims-Set = {
 * $$Claims-Set-Claims
 * Claim-Label .feature "extended-claims-label" => any
}
Claim-Label = CBOR-ONLY<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

;;; Insert the required CDDL from RFC 9052 to complete these
;;; definitions.  This can be done manually or automated by a
;;; tool that implements an import directive such as:
;# import rfc9052
```

Appendix B. Example

This appendix is informative.

The example CWT Claims Set from Appendix A.1 of [RFC8392] can be turned into a UCCS by enclosing it with a tag number CPA601:

```
 601(
   {
     / 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'
   }
 )
```

Appendix C. JSON Support

This appendix is informative.

The above definitions, concepts and security considerations all may be applied to define a JSON-encoded Claims-Set. Such an unsigned Claims-Set can be referred to as a "UJCS", an "Unprotected JWT Claims Set". The CDDL definition in Figure 1 can be used for a "UJCS".

```
UJCS = Claims-Set
```

Appendix D. EAT

This appendix is informative.

The following CDDL adds UCCS-format and UJCS-format tokens to EAT using its predefined extension points (see Section 4.2.18 (submods) of [I-D.ietf-rats-eat]).

```
$EAT-CBOR-Tagged-Token /= UCCS-Tagged
$EAT-CBOR-Untagged-Token /= UCCS-Untagged

$JSON-Selector /= [type: "UJCS", nested-token: UJCS]
```

Acknowledgements

Laurence Lundblade suggested some improvements to the CDDL. Carl Wallace provided a very useful review.

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: 30 August 2024

H. Birkholz
M. Eckel
Fraunhofer SIT
S. Bhandari
ThoughtSpot
E. Voit
B. Sulzen
Cisco
L. Xia
Huawei
T. Laffey
HPE
G. Fedorkow
Juniper

27 February 2024

A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs

draft-ietf-rats-yang-tpm-charra-22

Abstract

This document defines YANG Remote Procedure Calls (RPCs) and a few configuration nodes required to retrieve attestation evidence about integrity measurements from a device, following the operational context defined in TPM-based Network Device Remote Integrity Verification. Complementary measurement logs are also provided by the YANG RPCs, originating from one or more roots of trust for measurement (RTMs). The module defined requires at least one TPM 1.2 or TPM 2.0 as well as a corresponding TPM Software Stack (TSS), or equivalent hardware implementations that include the protected capabilities as provided by TPMs as well as a corresponding software stack, included in the device components of the composite device the YANG server is running on.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document is based on the general terminology defined in the [RFC9334] and uses the operational context defined in [I-D.ietf-rats-tpm-based-network-device-attest] as well as the interaction model and information elements defined in [I-D.ietf-rats-reference-interaction-models]. The currently supported hardware security modules (HSMs) are the Trusted Platform Modules (TPMs) [TPM1.2] and [TPM2.0] as specified by the Trusted Computing Group (TCG). One TPM, or multiple TPMs in the case of a

Composite Device, are required in order to use the YANG module defined in this document. Each TPM is used as a root of trust for storage (RTS) in order to store system security measurement Evidence. And each TPM is used as a root of trust for reporting (RTR) in order to retrieve attestation Evidence. This is done by using a YANG RPC to request a quote which exposes a rolling hash of the security measurements held internally within the TPM.

Specific terms imported from [RFC9334] and used in this document include: Attester, Composite Device, Evidence.

Specific terms imported from [TPM2.0-Key] and used in this document include: Endorsement Key (EK), Initial Attestation Key (IAK), Attestation Identity Key (AIK), Local Attestation Key (LAK).

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. The YANG Module for Basic Remote Attestation Procedures

One or more TPMs MUST be embedded in a Composite Device that provides attestation Evidence via the YANG module defined in this document. The ietf-tpm-remote-attestation YANG module enables a composite device to take on the role of an Attester, in accordance with the Remote Attestation Procedures (RATS) architecture [RFC9334], and the corresponding challenge-response interaction model defined in the [I-D.ietf-rats-reference-interaction-models] document. A fresh nonce with an appropriate amount of entropy [NIST-915121] MUST be supplied by the YANG client in order to enable a proof-of-freshness with respect to the attestation Evidence provided by the Attester running the YANG datastore. Further, this nonce is used to prevent replay attacks. The method for communicating the relationship of each individual TPM to specific measured component within the Composite Device is out of the scope of this document.

2.1. YANG Modules

In this section the several YANG modules are defined.

2.1.1. 'ietf-tpm-remote-attestation'

This YANG module imports modules from [RFC6991] with prefix 'yang', [RFC8348] with prefix 'hw', [I-D.ietf-netconf-keystore] with prefix 'ks', and 'ietf-tcg-algs.yang' Section 2.1.2.3 with prefix 'taa'. Additionally, references are made to [RFC8032], [RFC8017], [RFC6933], [TPM1.2-Commands], [TPM2.0-Arch], [TPM2.0-Structures], [TPM2.0-Key], [TPM1.2-Structures], [bios-log], [BIOS-Log-Event-Type], as well as Appendix A and Appendix B.

2.1.1.1. Features

This module supports the following features:

* 'mtpm': Indicates that multiple TPMs on the device can support remote attestation. For example, this feature could be used in cases where multiple line cards are present, each with its own TPM.

* 'bios': Indicates that the device supports the retrieval of BIOS/UEFI event logs. [bios-log]

* 'ima': Indicates that the device supports the retrieval of event logs from the Linux Integrity Measurement Architecture (IMA, see Appendix A).

* 'netequip_boot': Indicates that the device supports the retrieval of netequip boot event logs. See Appendix A and Appendix B.

2.1.1.2. Identities

This module supports the following types of attestation event logs: 'bios', 'ima', and 'netequip_boot'.

2.1.1.3. Remote Procedure Calls (RPCs)

In the following, RPCs for both TPM 1.2 and TPM 2.0 attestation procedures are defined.

2.1.1.3.1. 'tpm12-challenge-response-attestation'

This RPC allows a Verifier to request signed TPM PCRs (_TPM Quote_ operation) from a TPM 1.2 compliant cryptoprocessor. Where the feature 'mtpm' is active, and one or more 'certificate-name' is not provided, all TPM 1.2 compliant cryptoprocessors will respond. A YANG tree diagram of this RPC is as follows:

```
+---x tpm12-challenge-response-attestation {taa:tpm12}?
  +---w input
  |  +---w tpm12-attestation-challenge
  |     +---w pcr-index*          pcr
  |     +---w nonce-value         binary
  |     +---w certificate-name*   certificate-name-ref
  |             {tpm:mtpm}?
  +--ro output
     +--ro tpm12-attestation-response* []
        +--ro certificate-name    certificate-name-ref
        +--ro up-time?            uint32
        +--ro TPM_QUOTE2?         binary
```

2.1.1.3.2. 'tpm20-challenge-response-attestation'

This RPC allows a Verifier to request signed TPM PCRs (_TPM Quote_ operation) from a TPM 2.0 compliant cryptoprocessor. Where the feature 'mtpm' is active, and one or more 'certificate-name' is not provided, all TPM 2.0 compliant cryptoprocessors will respond. A YANG tree diagram of this RPC is as follows:

```
+---x tpm20-challenge-response-attestation {taa:tpm20}?
   +---w input
   |  +---w tpm20-attestation-challenge
   |     +---w nonce-value            binary
   |     +---w tpm20-pcr-selection* []
   |     |  +---w tpm20-hash-algo?   identityref
   |     |  +---w pcr-index*         pcr
   |     +---w certificate-name*      certificate-name-ref
   |             {tpm:mtpm}?
   +--ro output
      +--ro tpm20-attestation-response* []
         +--ro certificate-name       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
```

An example of an RPC challenge requesting PCRs 0-7 from a SHA-256 bank could look like the following:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tpm20-attestation-challenge
      xmlns="urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation">
    <certificate-name>
      (identifier of a TPM signature key with which the Attester is
      supposed to sign the attestation data)
    </certificate-name>
    <nonce-value>
      0xe041307208d9f78f5b1bbecd19e2d152ad49de2fc5a7d8dbf769f6b8ffdeab9
    </nonce-value>
    <tpm20-pcr-selection>
      <tpm20-hash-algo
          xmlns="urn:ietf:params:xml:ns:yang:ietf-tcg-algs">
        TPM_ALG_SHA256
      </tpm20-hash-algo>
      <pcr-index>0</pcr-index>
      <pcr-index>1</pcr-index>
      <pcr-index>2</pcr-index>
      <pcr-index>3</pcr-index>
      <pcr-index>4</pcr-index>
      <pcr-index>5</pcr-index>
      <pcr-index>6</pcr-index>
      <pcr-index>7</pcr-index>
    </tpm20-pcr-selection>
  </tpm20-attestation-challenge>
</rpc>
```

A successful response could be formatted as follows:

```
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tpm20-attestation-response
      xmlns="urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation">
    <certificate-name
        xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
      (instance of Certificate name in the Keystore)
    </certificate-name>
    <attestation-data>
      (raw attestation data, i.e., the TPM quote; this includes,
      among other information, a composite digest of requested PCRs,
      the nonce, and TPM 2.0 clock information.)
    </attestation-data>
    <quote-signature>
      (signature over attestation-data using the TPM key
      identified by sig-key-id)
    </quote-signature>
  </tpm20-attestation-response>
</rpc-reply>
```

2.1.1.4. 'log-retrieval'

This RPC allows a Verifier to acquire the Evidence which was extended into specific TPM PCRs. A YANG tree diagram of this RPC is as follows:

```
+---x log-retrieval
   +---w input
   |  +---w log-type        identityref
   |  +---w log-selector* []
   |     +---w name*                      string
   |     +---w (index-type)?
   |     |  +--:(last-entry)
   |     |  |  +---w last-entry-value?    binary
   |     |  +--:(index)
   |     |  |  +---w last-index-number?   uint64
   |     |  +--:(timestamp)
   |     |     +---w timestamp?           yang:date-and-time
   |     +---w log-entry-quantity?        uint16
   +--ro output
      +--ro system-event-logs
         +--ro node-data* []
            +--ro name?         string
            +--ro up-time?      uint32
            +--ro log-result
               +--ro (attested_event_log_type)
                  +--:(bios) {bios}?
                  |  +--ro bios-event-logs
                  |     +--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*     binary
                  +--:(ima) {ima}?
                  |  +--ro ima-event-logs
                  |     +--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) {netequip_boot}?
                        +--ro boot-event-logs
                           +--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
```

2.1.1.5. Data Nodes

This section provides a high level description of the data nodes containing the configuration and operational objects with the YANG model. For more details, please see the YANG model itself in Figure 1.

Container 'rats-support-structures': This houses the set of information relating to remote attestation for a device. This includes specific device TPM(s), the compute nodes (such as line cards) on which the TPM(s) reside, and the algorithms supported across the platform.

Container 'tpms': Provides configuration and operational details for each supported TPM, including the tpm-firmware-version, PCRs which may be quoted, certificates which are associated with that TPM, and the current operational status. Of note are the certificates which are associated with that TPM. As a certificate is associated with a particular TPM attestation key, knowledge of the certificate allows a specific TPM to be identified.

```
+--rw tpms
   +--rw tpm* [name]
      +--rw name                string
      +--ro hardware-based      boolean
      +--ro physical-index?     int32 {hw:entity-mib}?
      +--ro path?               string
      +--ro compute-node        compute-node-ref {tpm:mtpm}?
      +--ro manufacturer?       string
      +--rw firmware-version    identityref
      +--rw tpm12-hash-algo?    identityref {taa:tpm12}?
      +--rw tpm12-pcrs*         pcr
      +--rw tpm20-pcr-bank* [tpm20-hash-algo]  {taa:tpm20}?
      |  +--rw tpm20-hash-algo    identityref
      |  +--rw pcr-index*         tpm:pcr
      +--ro status              enumeration
      +--rw certificates
         +--rw certificate* [name]
            +--rw name            string
            +--rw keystore-ref?   leafref {ks:asymmetric-keys}?
            +--rw type?           enumeration
```

container 'attester-supported-algos' - Identifies which TCG hash algorithms are available for use on the Attesting platform. An operator will use this information to limit algorithms available for use by RPCs to just a desired set from the universe of all allowed hash algorithms by the TCG.

```
+--rw attester-supported-algos
   +--rw tpm12-asymmetric-signing*   identityref {taa:tpm12}?
   +--rw tpm12-hash*                 identityref {taa:tpm12}?
   +--rw tpm20-asymmetric-signing*   identityref {taa:tpm20}?
   +--rw tpm20-hash*                 identityref {taa:tpm20}?
```

container 'compute-nodes' - When there is more than one TPM supported, this container maintains the set of information related to the compute node associated with a specific TPM. This allows each specific TPM to identify to which 'compute-node' it belongs.

```
+--rw compute-nodes {tpm:mtpm}?
   +--ro compute-node* [node-id]
      +--ro node-id                string
      +--ro node-physical-index?   int32 {hw:entity-mib}?
      +--ro node-name?             string
      +--ro node-location?         string
```

2.1.1.6. YANG Module

```
<CODE BEGINS> file "ietf-tpm-remote-attestation.yang"
module ietf-tpm-remote-attestation {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation";
  prefix tpm;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-hardware {
    prefix hw;
  }
  import ietf-keystore {
    prefix ks;
  }
  import ietf-tcg-algs {
    prefix taa;
  }

  organization
    "IETF RATS (Remote ATtestation procedureS) Working Group";
  contact
    "WG Web  : <https://datatracker.ietf.org/wg/rats/>
     WG List : <mailto:rats@ietf.org>
     Author  : Eric Voit <evoit@cisco.com>
     Author  : Henk Birkholz <henk.birkholz@ietf.contact>
     Author  : Michael Eckel <michael.eckel@sit.fraunhofer.de>
     Author  : Shwetha Bhandari <shwetha.bhandari@thoughtspot.com>
     Author  : Bill Sulzen <bsulzen@cisco.com>
     Author  : Liang Xia (Frank) <frank.xialiang@huawei.com>
     Author  : Tom Laffey <tom.laffey@hpe.com>
     Author  : Guy Fedorkow <gfedorkow@juniper.net>";
  description
    "A YANG module to enable a TPM 1.2 and TPM 2.0 based
     remote attestation procedure using a challenge-response
     interaction model and the TPM 1.2 and TPM 2.0 Quote
     primitive operations.

     Copyright (c) 2022 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 Revised 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.

        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 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

     revision 2022-05-17 {
       description
         "Initial version";
       reference
         "RFC XXXX: A YANG Data Model for Challenge-Response-based Remote
          Attestation Procedures using TPMs";
     }

     /*****************/
     /*   Features    */
     /*****************/

     feature mtpm {
       description
         "The device supports the remote attestation of multiple
          TPM based cryptoprocessors.";
     }

     feature bios {
       description
         "The device supports the bios logs.";
       reference
         "bios-log:
          https://trustedcomputinggroup.org/wp-content/uploads/
          PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf
          Section 9.4.5.2";
     }

     feature ima {
       description
         "The device supports Integrity Measurement Architecture logs.
          Many variants of IMA logs exist in the deployment.  Each encodes
          the log entry contents as the specific measurements which get
          hashed into a PCRs as Evidence.  See the reference below for
          one example of such an encoding.";
       reference
         "ima-log:
          https://www.trustedcomputinggroup.org/wp-content/uploads/
          TCG_IWG_CEL_v1_r0p41_pub.pdf  Section 5.1.6";
```

```
     }

     feature netequip_boot {
       description
         "The device supports the netequip_boot logs.";
       reference
         "netequip-boot-log:
          RFC XXXX  Appendix B";
     }

     /*****************/
     /*   Typedefs    */
     /*****************/

     typedef pcr {
       type uint8 {
         range "0..31";
       }
       description
         "Valid index number for a PCR.  A {{TPM2.0}} compliant PCR index
          extends from 0-31.  At this time a typical TPM would have no
          more than 32 PCRS.";
     }

     typedef compute-node-ref {
       type leafref {
         path "/tpm:rats-support-structures/tpm:compute-nodes"
            + "/tpm:compute-node/tpm:node-id";
       }
       description
         "This type is used to reference a hardware node.  Note that an
          implementer might include an alternative leafref pointing to a
          different YANG module node specifying hardware structures.";
     }

     typedef certificate-name-ref {
       type leafref {
         path "/tpm:rats-support-structures/tpm:tpms/tpm:tpm"
            + "/tpm:certificates/tpm:certificate/tpm:name";
       }
       description
         "A type which allows identification of a TPM based certificate.";
     }

     /******************/
     /*   Identities   */
     /******************/
```

```
identity attested_event_log_type {
  description
    "Base identity allowing categorization of the reasons why an
     attested measurement has been taken on an Attester.";
}

identity ima {
  base attested_event_log_type;
  description
    "An event type recorded in IMA.";
}

identity bios {
  base attested_event_log_type;
  description
    "An event type associated with BIOS/UEFI.";
}

identity netequip_boot {
  base attested_event_log_type;
  description
    "An event type associated with Network Equipment Boot.";
}

/*****************/
/*   Groupings   */
/*****************/

grouping tpm20-hash-algo {
  description
    "The cryptographic algorithm used to hash the TPM2 PCRs.  This
     must be from the list of platform supported options.";
  leaf tpm20-hash-algo {
    type identityref {
      base taa:hash;
    }
    must '. = /tpm:rats-support-structures'
       + '/tpm:attester-supported-algos/tpm:tpm20-hash' {
      error-message "This platform does not support tpm20-hash-algo";
    }
    description
      "The hash scheme that is used to hash a TPM2.0 PCR. This
       must be one of those supported by a platform.
       Where this object does not appear, the default value of
       'taa:TPM_ALG_SHA256' will apply.";
  }
}
```

```
     grouping tpm12-hash-algo {
       description
         "The cryptographic algorithm used to hash the TPM1.2 PCRs.";
       leaf tpm12-hash-algo {
         type identityref {
           base taa:hash;
         }
         must '. = /tpm:rats-support-structures'
            + '/tpm:attester-supported-algos/tpm:tpm12-hash' {
           error-message "This platform does not support tpm12-hash-algo";
         }
         description
           "The hash scheme that is used to hash a TPM1.2 PCR. This
            MUST be one of those supported by a platform.
            Where this object does not appear, the default value of
            'taa:TPM_ALG_SHA1' will apply.";
       }
     }

     grouping nonce {
       description
         "A random number intended to guarantee freshness and for use
          as part of a replay-detection mechanism.";
       leaf nonce-value {
         type binary;
         mandatory true;
         description
           "A cryptographically generated random number which should
            not be predictable prior to its issuance from a random
            number generation function. The random number MUST be
            derived from an entropy source external to the Attester.

            Note that a nonce sent into a TPM will typically be 160 or 256
            binary digits long.  (This is 20 or 32 bytes.) So if fewer
            binary digits are sent, this nonce object will be padded
            with leading zeros within Quotes returned from the TPM.
            Additionally if more bytes are sent, the nonce will be trimmed
            to the most significant binary digits.";
       }
     }

     grouping tpm12-pcr-selection {
       description
         "A Verifier can request one or more PCR values using its
          individually created Attestation Key Certificate (AC).
          The corresponding selection filter is represented in this
          grouping.";
       leaf-list pcr-index {
```

```
        type pcr;
        description
          "The numbers/indexes of the PCRs. In addition, any selection
           of PCRs MUST verify that the set of PCRs requested are a
           subset the set of PCRs exposed by in the leaf-list
           /tpm:rats-support-structures
           /tpm:tpms/tpm:tpm[name=current()]/tpm:tpm12-pcrs";
      }
    }

    grouping tpm20-pcr-selection {
      description
        "A Verifier can acquire one or more PCR values, which are hashed
         together in a TPM2B_DIGEST coming from the TPM2.  The selection
         list of desired PCRs and the Hash Algorithm is represented in
         this grouping.";
      list tpm20-pcr-selection {
        unique "tpm20-hash-algo";
        description
          "Specifies the list of PCRs and Hash Algorithms that can be
           returned within a TPM2B_DIGEST.";
        reference
          "TPM2.0-Structures:
           https://www.trustedcomputinggroup.org/wp-content/uploads/
           TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.9.7";
        uses tpm20-hash-algo;
        leaf-list pcr-index {
          type pcr;
          description
            "The numbers of the PCRs that which are being tracked
             with a hash based on the tpm20-hash-algo.  In addition,
             any selection of PCRs MUST verify that the set of PCRs
             requested are a subset the set of PCR indexes selected
             are available for that specific TPM.";
        }
      }
    }

    grouping certificate-name-ref {
      description
        "Identifies a certificate in a keystore.";
      leaf certificate-name {
        type certificate-name-ref;
        mandatory true;
        description
          "Identifies a certificate in a keystore.";
      }
    }
```

```
     grouping tpm-name {
       description
         "A unique TPM on a device.";
       leaf name {
         type string;
         description
           "Unique system generated name for a TPM on a device.";
       }
     }

     grouping node-uptime {
       description
         "Uptime in seconds of the node.";
       leaf up-time {
         type uint32;
         description
           "Uptime in seconds of this node reporting its data";
       }
     }

     grouping tpm12-attestation {
       description
         "Contains an instance of TPM1.2 style signed cryptoprocessor
          measurements.  It is supplemented by unsigned Attester
          information.";
       uses node-uptime;
       leaf pcr-data {
         type binary;
         description
           "The value created and signed for the quote (type TPM_PCR_INFO_SHORT),
            i.e., the 'pcrData' part of a TPM1.2 Quote2 operation result.";
         reference
           "TPM1.2-Commands:
            TPM1.2 commands rev116 July 2007, Section 16.5
            https://trustedcomputinggroup.org/wp-content/uploads
            /TPM-Main-Part-3-Commands_v1.2_rev116_01032011.pdf";
       }
       leaf version-info {
         type binary;
         description
           "The version info (type TPM_CAP_VERSION_INFO),
            i.e., the 'versionInfo' part of a TPM1.2 Quote2 operation result.";
         reference
           "TPM1.2-Commands:
            TPM1.2 commands rev116 July 2007, Section 16.5
            https://trustedcomputinggroup.org/wp-content/uploads
            /TPM-Main-Part-3-Commands_v1.2_rev116_01032011.pdf";
       }
```

```
      leaf sig {
        type binary;
        description
          "The signed data blob, i.e., the signature
           i.e., the 'sig' part of a TPM1.2 Quote2 operation result.";
        reference
          "TPM1.2-Commands:
           TPM1.2 commands rev116 July 2007, Section 16.5
           https://trustedcomputinggroup.org/wp-content/uploads
           /TPM-Main-Part-3-Commands_v1.2_rev116_01032011.pdf";
      }
    }

    grouping tpm20-attestation {
      description
        "Contains an instance of TPM2 style signed cryptoprocessor
         measurements.  It is supplemented by unsigned Attester
         information.";
      leaf quote-data {
        type binary;
        mandatory true;
        description
          "A hash of the latest PCR values (and the hash algorithm used)
           which have been returned from an Attester for the selected PCRs
           and Hash Algorithms.";
        reference
          "TPM2.0-Structures:
           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;
        description
          "Quote signature returned by TPM Quote.  The signature was
           generated using the key associated with the
           certificate 'name'.";
        reference
          "TPM2.0-Structures:
           https://www.trustedcomputinggroup.org/wp-content/uploads/
           TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 11.2.1";
      }
      uses node-uptime;
      list unsigned-pcr-values {
        description
          "PCR values in each PCR bank. This might appear redundant with
           the TPM2B_DIGEST, but that digest is calculated across multiple
           PCRs.  Having to verify across multiple PCRs does not
           necessarily make it easy for a Verifier to appraise just the
```

```
           minimum set of PCR information which has changed since the last
           received TPM2B_DIGEST.  Put another way, why should a Verifier
           reconstruct the proper value of all PCR Quotes when only a
           single PCR has changed?
           To help this happen, if the Attester does know specific PCR
           values, the Attester can provide these individual values via
           'unsigned-pcr-values'.   By comparing this information to
           what has previously been validated, it is possible for a
           Verifier to confirm the Attester's signature while eliminating

           significant processing.  Note that there should never be a
           result where an unsigned PCR value differs from what may be
           reconstructed from the within the PCR quote and the event logs.
           If there is a difference, a signed result which has been
           verified from retrieved logs is considered definitive.";
        uses tpm20-hash-algo;
        list pcr-values {
          key "pcr-index";
          description
            "List of one PCR bank.";
          leaf pcr-index {
            type pcr;
            description
              "PCR index number.";
          }
          leaf pcr-value {
            type binary;
            description
              "PCR value.";
            reference
              "TPM2.0-Structures:
               https://www.trustedcomputinggroup.org/wp-content/uploads/
               TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.9.7";
          }
        }
      }
    }

    grouping log-identifier {
      description
        "Identifier for type of log to be retrieved.";
      leaf log-type {
        type identityref {
          base attested_event_log_type;
        }
        mandatory true;
        description
          "The corresponding measurement log type identity.";
```

```
      }
    }

    grouping boot-event-log {
      description
        "Defines a specific instance of an event log entry
         and corresponding to the information used to
         extend the PCR";
      leaf event-number {
        type uint32;
        description
          "Unique event number of this event which monotonically
           increases within a given event log.  The maximum event
           number should not be reached, nor is wrapping back to
           an earlier number supported.";
      }
      leaf event-type {
        type uint32;
        description
          "BIOS Log Event Type:
           https://trustedcomputinggroup.org/wp-content/uploads/
           TCG_PCClient_PFP_r1p05_v23_pub.pdf  Section 10.4.1";
      }
      leaf pcr-index {
        type pcr;
        description
          "Defines the PCR index that this event extended";
      }
      list digest-list {
        description
          "Hash of event data";
        leaf hash-algo {
          type identityref {
            base taa:hash;
          }
          description
            "The hash scheme that is used to compress the event data in
             each of the leaf-list digest items.";
        }
        leaf-list digest {
          type binary;
          description
            "The hash of the event data using the algorithm of the
             'hash-algo' against 'event data'.";
        }
      }
      leaf event-size {
        type uint32;
```

```
        description
          "Size of the event data";
      }
      leaf-list event-data {
        type binary;
        description
          "The event data.  This is a binary structure
           of size 'event-size'. For more on what
           might be recorded within this object
           see [bios-log] Section 9 which details
           viable events which might be recorded.";
      }
    }

    grouping bios-event-log {
      description
        "Measurement log created by the BIOS/UEFI.";
      list bios-event-entry {
        key "event-number";
        description
          "Ordered list of TCG described event log
           that extended the PCRs in the order they
           were logged";
        uses boot-event-log;
      }
    }

    grouping ima-event {
      description
        "Defines a hash log extend event for IMA measurements";
      reference
        "ima-log:
         https://www.trustedcomputinggroup.org/wp-content/uploads/
         TCG_IWG_CEL_v1_r0p41_pub.pdf  Section 4.3";
      leaf event-number {
        type uint64;
        description
          "Unique event number of this event which monotonically
           increases.  The maximum event number should not be
           reached, nor is wrapping back to an earlier number
           supported.";
      }
      leaf ima-template {
        type string;
        description
          "Name of the template used for event logs
           for e.g. ima, ima-ng, ima-sig";
      }
```

```
      leaf filename-hint {
        type string;
        description
          "File name (including the path) that was measured.";
      }
      leaf filedata-hash {
        type binary;
        description
          "Hash of filedata as updated based upon the
           filedata-hash-algorithm";
      }
      leaf filedata-hash-algorithm {
        type string;
        description
          "Algorithm used for filedata-hash";
      }
      leaf template-hash-algorithm {
        type string;
        description
          "Algorithm used for template-hash";
      }
      leaf template-hash {
        type binary;
        description
          "hash(filedata-hash, filename-hint)";
      }
      leaf pcr-index {
        type pcr;
        description
          "Defines the PCR index that this event extended";
      }
      leaf signature {
        type binary;
        description
          "Digital file signature which provides a
           fingerprint for the file being measured.";
      }
    }

    grouping ima-event-log {
      description
        "Measurement log created by IMA.";
      list ima-event-entry {
        key "event-number";
        description
          "Ordered list of ima event logs by event-number";
        uses ima-event;
      }
```

```
     }

     grouping network-equipment-boot-event-log {
       description
         "Measurement log created by Network Equipment Boot. The Network
          Equipment Boot format is identical to the IMA format. In
          contrast to the IMA log, the Network Equipment Boot log
          includes every measurable event from an Attester, including
          the boot stages of BIOS, Bootloader, etc. In essence, the scope
          of events represented in this format combines the scope of BIOS
          events and IMA events.";
       list boot-event-entry {
         key "event-number";
         description
           "Ordered list of Network Equipment Boot event logs
            by event-number, using the IMA event format.";
         uses ima-event;
       }
     }

     grouping event-logs {
       description
         "A selector for the log and its type.";
       choice attested_event_log_type {
         mandatory true;
         description
           "Event log type determines the event logs content.";
         case bios {
           if-feature "bios";
           description
             "BIOS/UEFI event logs";
           container bios-event-logs {
             description
               "BIOS/UEFI event logs";
             uses bios-event-log;
           }
         }
         case ima {
           if-feature "ima";
           description
             "IMA event logs.";
           container ima-event-logs {
             description
               "IMA event logs.";
             uses ima-event-log;
           }
         }
         case netequip_boot {
```

```
          if-feature "netequip_boot";
          description
            "Network Equipment Boot event logs";
          container boot-event-logs {
            description
              "Network equipment boot event logs.";
            uses network-equipment-boot-event-log;
          }
        }
      }
    }

    /**********************/
    /*   RPC operations   */
    /**********************/

    rpc tpm12-challenge-response-attestation {
      if-feature "taa:tpm12";
      description
        "This RPC accepts the input for TSS TPM 1.2 commands made to the
         attesting device.";
      input {
        container tpm12-attestation-challenge {
          description
            "This container includes every information element defined
             in the reference challenge-response interaction model for
             remote attestation. Corresponding values are based on
             TPM 1.2 structure definitions";
          uses tpm12-pcr-selection;
          uses nonce;
          leaf-list certificate-name {
            if-feature "tpm:mtpm";
            type certificate-name-ref;
            must "/tpm:rats-support-structures/tpm:tpms"
               + "/tpm:tpm[tpm:firmware-version='taa:tpm12']"
               + "/tpm:certificates/"
               + "/tpm:certificate[name=current()]" {
              error-message "Not an available TPM1.2 AIK certificate.";
            }
            description
              "When populated, the RPC will only get a Quote for the
               TPMs associated with these certificate(s).";
          }
        }
      }
      output {
        list tpm12-attestation-response {
          unique "certificate-name";
```

```
          description
            "The binary output of TPM 1.2 TPM_Quote/TPM_Quote2, including
             the PCR selection and other associated attestation evidence
             metadata";
          uses certificate-name-ref {
            description
              "Certificate associated with this tpm12-attestation.";
          }
          uses tpm12-attestation;
        }
      }
    }

    rpc tpm20-challenge-response-attestation {
      if-feature "taa:tpm20";
      description
        "This RPC accepts the input for TSS TPM 2.0 commands of the
         managed device. ComponentIndex from the hardware manager YANG
         module is used to refer to dedicated TPM in composite devices,
         e.g. smart NICs, is not covered.";
      input {
        container tpm20-attestation-challenge {
          description
            "This container includes every information element defined
             in the reference challenge-response interaction model for
             remote attestation. Corresponding values are based on
             TPM 2.0 structure definitions";
          uses nonce;
          uses tpm20-pcr-selection;
          leaf-list certificate-name {
            if-feature "tpm:mtpm";
            type certificate-name-ref;
            must "/tpm:rats-support-structures/tpm:tpms"
               + "/tpm:tpm[tpm:firmware-version='taa:tpm20']"
               + "/tpm:certificates/"
               + "/tpm:certificate[name=current()]" {
              error-message "Not an available TPM2.0 AIK certificate.";
            }
            description
              "When populated, the RPC will only get a Quote for the
               TPMs associated with the certificates.";
          }
        }
      }
      output {
        list tpm20-attestation-response {
          unique "certificate-name";
          description
```

```
            "The binary output of TPM2_Quote from one TPM of the
             node which identified by node-id. An TPMS_ATTEST structure
             including a length, encapsulated in a signature";
          uses certificate-name-ref {
            description
              "Certificate associated with this tpm20-attestation.";
          }
          uses tpm20-attestation;
        }
      }
    }

    rpc log-retrieval {
      description
        "Logs Entries are either identified via indices or via providing
         the last line received. The number of lines returned can be
         limited. The type of log is a choice that can be augmented.";
      input {
        uses log-identifier;
        list log-selector {
          description
            "Only log entries which meet all the selection criteria
             provided are to be returned by the RPC output.";
          leaf-list name {
            type string;
            description
              "Name of one or more unique TPMs on a device.  If this
               object exists, a selection should pull only the objects
               related to these TPM(s).  If it does not exist, all
               qualifying TPMs that are 'hardware-based' equals true
               on the device are selected. When this selection
               criteria is provided, it will be considered as a logical
               AND with any other selection criteria provided.";
          }
          choice index-type {
            description
              "Last log entry received, log index number, or timestamp.";
            case last-entry {
              description
                "The last entry of the log already retrieved.";
              leaf last-entry-value {
                type binary;
                description
                  "Content of a log event which matches 1:1 with a
                   unique event record contained within the log.  Log
                   entries after this will be passed to the
                   requester.  Note: if log entry values are not unique,
                   this MUST return an error.";
```

```
              }
            }
            case index {
              description
                "Numeric index of the last log entry retrieved, or
                 zero.";
              leaf last-index-number {
                type uint64;
                description
                  "The last numeric index number of a log entry.
                   Zero means to start at the beginning of the log.
                   Entries after this will be passed to the
                   requester.";
              }
            }
            case timestamp {
              leaf timestamp {
                type yang:date-and-time;
                description
                  "Timestamp from which to start the extraction.  The
                   next log entry after this timestamp is to
                   be sent.";
              }
              description
                "Timestamp from which to start the extraction.";
            }
          }
          leaf log-entry-quantity {
            type uint16;
            description
              "The number of log entries to be returned. If omitted, it
               means all of them.";
          }
        }
      }
      output {
        container system-event-logs {
          description
            "The requested data of the measurement event logs";
          list node-data {
            unique "name";
            description
              "Event logs of a node in a distributed system
               identified by the node name";
            uses tpm-name;
            uses node-uptime;
            container log-result {
              description
```

```
                "The requested entries of the corresponding log.";
              uses event-logs;
            }
          }
        }
      }
    }

    /***************************************/
    /*   Config & Oper accessible nodes   */
    /***************************************/

    container rats-support-structures {
      description
        "The datastore definition enabling verifiers or relying
         parties to discover the information necessary to use the
         remote attestation RPCs appropriately.";
      container compute-nodes {
        if-feature "tpm:mtpm";
        description
          "Holds the set of device subsystems/components in this
           composite device that support TPM operations.";
        list compute-node {
          key "node-id";
          unique "node-name";
          config false;
          min-elements 2;
          description
            "A component within this composite device which
             supports TPM operations.";
          leaf node-id {
            type string;
            description
              "ID of the compute node, such as Board Serial Number.";
          }
          leaf node-physical-index {
            if-feature "hw:entity-mib";
            type int32 {
              range "1..2147483647";
            }
            config false;
            description
              "The entPhysicalIndex for the compute node.";
            reference
              "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
          }
          leaf node-name {
            type string;
```

```
            description
              "Name of the compute node.";
          }
          leaf node-location {
            type string;
            description
              "Location of the compute node, such as slot number.";
          }
        }
      }
      container tpms {
        description
          "Holds the set of TPMs within an Attester.";
        list tpm {
          key "name";
          unique "path";
          description
            "A list of TPMs in this composite device that RATS
             can be conducted with.";
          uses tpm-name;
          leaf hardware-based {
            type boolean;
            config false;
            mandatory true;
            description
              "System generated indication of whether this is a
               hardware based TPM.";
          }
          leaf physical-index {
            if-feature "hw:entity-mib";
            type int32 {
              range "1..2147483647";
            }
            config false;
            description
              "The entPhysicalIndex for the TPM.";
            reference
              "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
          }
          leaf path {
            type string;
            config false;
            description
              "Device path to a unique TPM on a device.  This can change
               across reboots.";
          }
          leaf compute-node {
            if-feature "tpm:mtpm";
```

```
            type compute-node-ref;
            config false;
            mandatory true;
            description
              "Indicates the compute node measured by this TPM.";
          }
          leaf manufacturer {
            type string;
            config false;
            description
              "TPM manufacturer name.";
          }
          leaf firmware-version {
            type identityref {
              base taa:cryptoprocessor;
            }
            mandatory true;
            description
              "Identifies the cryptoprocessor API set supported.  This
               is automatically configured by the device and should not
               be changed.";
          }
          uses tpm12-hash-algo {
            when "derived-from-or-self(firmware-version, 'taa:tpm12')";
            if-feature "taa:tpm12";
            refine "tpm12-hash-algo" {
              description
                "The hash algorithm overwrites the default used for PCRs
                 on this TPM1.2 compliant cryptoprocessor.";
            }
          }
          leaf-list tpm12-pcrs {
            when
              "derived-from-or-self(../firmware-version, 'taa:tpm12')";
            if-feature "taa:tpm12";
            type pcr;
            description
              "The PCRs which may be extracted from this TPM1.2
               compliant cryptoprocessor.";
          }
          list tpm20-pcr-bank {
            when
              "derived-from-or-self(../firmware-version, 'taa:tpm20')";
            if-feature "taa:tpm20";
            key "tpm20-hash-algo";
            description
              "Specifies the list of PCRs that may be extracted for
               a specific Hash Algorithm on this TPM2 compliant
```

```
                cryptoprocessor.  A bank is a set of PCRs which are
                extended using a particular hash algorithm.";
            reference
              "TPM2.0-Structures:
               https://www.trustedcomputinggroup.org/wp-content/uploads/
               TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.9.7";
            leaf tpm20-hash-algo {
              type identityref {
                base taa:hash;
              }
              must '/tpm:rats-support-structures'
                 + '/tpm:attester-supported-algos'
                 + '/tpm:tpm20-hash' {
                error-message "This platform does not support tpm20-hash-algo";
              }
              description
                "The hash scheme actively being used to hash a
                 one or more TPM2.0 PCRs.";
            }
            leaf-list pcr-index {
              type tpm:pcr;
              description
                "Defines what TPM2 PCRs are available to be extracted.";
            }
          }
          leaf status {
            type enumeration {
              enum operational {
                value 0;
                description
                  "The TPM currently is running normally and
                   is ready to accept and process TPM quotes.";
                reference
                  "TPM2.0-Arch:
                   https://trustedcomputinggroup.org/wp-content/uploads/
                   TCG_TPM2_r1p59_Part1_Architecture_pub.pdf
                   Section 12";
              }
              enum non-operational {
                value 1;
                description
                  "TPM is in a state such as startup or shutdown which
                   precludes the processing of TPM quotes.";
              }
            }
            config false;
            mandatory true;
            description
```

```
              "TPM chip self-test status.";
          }
          container certificates {
            description
              "The TPM's certificates, including EK certificates
               and Attestation Key certificates.";
            list certificate {
              key "name";
              description
                "Three types of certificates can be accessed via
                 this statement, including Initial Attestation
                 Key Certificate, Local Attestation Key Certificate or
                 Endorsement Key Certificate.";
              leaf name {
                type string;
                description
                  "An arbitrary name uniquely identifying a certificate
                   associated within key within a TPM.";
              }
              leaf keystore-ref {
                if-feature "ks:central-keystore-supported";
                if-feature "ks:asymmetric-keys";
                type leafref {
                  path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
                     + "/ks:name";
                }
                description
                  "A reference to a specific certificate of an
                   asymmetric key in the Keystore.";
              }
              leaf type {
                type enumeration {
                  enum endorsement-certificate {
                    value 0;
                    description
                      "Endorsement Key (EK) Certificate type.";
                    reference
                      "TPM2.0-Key:
                       https://trustedcomputinggroup.org/wp-content/
                       uploads/TPM-2p0-Keys-for-Device-Identity-
                       and-Attestation_v1_r12_pub10082021.pdf
                       Section 3.11";
                  }
                  enum initial-attestation-certificate {
                    value 1;
                    description
                      "Initial Attestation key (IAK) Certificate type.";
                    reference
```

```
                      "TPM2.0-Key:
                       https://trustedcomputinggroup.org/wp-content/
                       uploads/TPM-2p0-Keys-for-Device-Identity-
                       and-Attestation_v1_r12_pub10082021.pdf
                       Section 3.2";
                  }
                  enum local-attestation-certificate {
                    value 2;
                    description
                      "Local Attestation Key (LAK) Certificate type.";
                    reference
                      "TPM2.0-Key:
                       https://trustedcomputinggroup.org/wp-content/
                       uploads/TPM-2p0-Keys-for-Device-Identity-
                       and-Attestation_v1_r12_pub10082021.pdf
                       Section 3.2";
                  }
                }
                description
                  "Function supported by this certificate from within the
                   TPM.";
              }
            }
          }
        }
      }
      container attester-supported-algos {
        description
          "Identifies which TPM algorithms are available for use on an
           attesting platform.";
        leaf-list tpm12-asymmetric-signing {
          when "../../tpm:tpms"
             + "/tpm:tpm[tpm:firmware-version='taa:tpm12']";
          if-feature "taa:tpm12";
          type identityref {
            base taa:asymmetric;
          }
          description
            "Platform Supported TPM12 asymmetric algorithms.";
        }
        leaf-list tpm12-hash {
          when "../../tpm:tpms"
             + "/tpm:tpm[tpm:firmware-version='taa:tpm12']";
          if-feature "taa:tpm12";
          type identityref {
            base taa:hash;
          }
          description
```

```
            "Platform supported TPM12 hash algorithms.";
        }
        leaf-list tpm20-asymmetric-signing {
          when "../../tpm:tpms"
             + "/tpm:tpm[tpm:firmware-version='taa:tpm20']";
          if-feature "taa:tpm20";
          type identityref {
            base taa:asymmetric;
          }
          description
            "Platform Supported TPM20 asymmetric algorithms.";
        }
        leaf-list tpm20-hash {
          when "../../tpm:tpms"
             + "/tpm:tpm[tpm:firmware-version='taa:tpm20']";
          if-feature "taa:tpm20";
          type identityref {
            base taa:hash;
          }
          description
            "Platform supported TPM20 hash algorithms.";
        }
      }
    }
  }
  <CODE ENDS>
```

Figure 1

2.1.2. 'ietf-tcg-algs'

This document has encoded the TCG Algorithm definitions of [TCG-Algos], revision 1.32. By including this full table as a separate YANG file within this document, it is possible for other YANG models to leverage the contents of this model. Specific references to [RFC2104], [RFC8017], [ISO-IEC-9797-1], [ISO-IEC-9797-2], [ISO-IEC-10116], [ISO-IEC-10118-3], [ISO-IEC-14888-3], [ISO-IEC-15946-1], [ISO-IEC-18033-3], [IEEE-Std-1363-2000], [IEEE-Std-1363a-2004], [NIST-PUB-FIPS-202], [NIST-SP800-38C], [NIST-SP800-38D], [NIST-SP800-38F], [NIST-SP800-56A], [NIST-SP800-108], [bios-log], as well as Appendix A and Appendix B exist within the YANG Model.

2.1.2.1. Features

There are two types of features supported: 'TPM12' and 'TPM20'. Support for either of these features indicates that a cryptoprocessor supporting the corresponding type of TCG TPM API is present on an Attester. Most commonly, only one type of cryptoprocessor will be available on an Attester.

2.1.2.2. Identities

There are three types of identities in this model:

1. Cryptographic functions supported by a TPM algorithm; these include: 'asymmetric', 'symmetric', 'hash', 'signing', 'anonymous_signing', 'encryption_mode', 'method', and 'object_type'. The definitions of each of these are in Table 2 of [TCG-Algos].

2. API specifications for TPM types: 'tpm12' and 'tpm20'

3. Specific algorithm types: Each algorithm type defines what cryptographic functions may be supported, and on which type of API specification. It is not required that an implementation of a specific TPM will support all algorithm types. The contents of each specific algorithm mirrors what is in Table 3 of [TCG-Algos].

2.1.2.3. YANG Module

```
<CODE BEGINS> file "ietf-tcg-algs@2022-03-23.yang"
module ietf-tcg-algs {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcg-algs";
  prefix taa;

  organization
    "IETF RATS (Remote ATtestation procedureS) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/rats/>
     WG List:  <mailto:rats@ietf.org>
     Author:   Eric Voit <mailto:evoit@cisco.com>";
  description
    "This module defines identities for asymmetric algorithms.

     Copyright (c) 2022 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 Revised
        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.

        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 (RFC 2119)
        (RFC 8174) when, and only when, they appear in all
        capitals, as shown here.";

     revision 2022-03-23 {
       description
         "Initial version";
       reference
         "RFC XXXX: A YANG Data Model for Challenge-Response-based Remote
          Attestation Procedures using TPMs";
     }

     /*****************/
     /*   Features    */
     /*****************/

     feature tpm12 {
       description
         "This feature indicates algorithm support for the TPM 1.2 API
          as per Section 4.8 of TPM1.2-Structures:
          TPM Main Part 2 TPM Structures
          https://trustedcomputinggroup.org/wp-content/uploads/TPM-
          Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf";
     }

     feature tpm20 {
       description
         "This feature indicates algorithm support for the TPM 2.0 API
          as per Section 11.4 of Trusted Platform Module Library
          Part 1: Architecture. See TPM2.0-Arch:
          https://trustedcomputinggroup.org/wp-content/uploads/
          TCG_TPM2_r1p59_Part1_Architecture_pub.pdf";
     }

     /*****************/
     /*  Identities   */
```

```
/*****************/

identity asymmetric {
  description
    "A TCG recognized asymmetric algorithm with a public and
     private key.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2,
     https://trustedcomputinggroup.org/resource/
     tcg-algorithm-registry/TCG-_Algorithm_Registry_r1p32_pub";
}

identity symmetric {
  description
    "A TCG recognized symmetric algorithm with only a private key.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2";
}

identity hash {
  description
    "A TCG recognized hash algorithm that compresses input data to
     a digest value or indicates a method that uses a hash.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2";
}

identity signing {
  description
    "A TCG recognized signing algorithm";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2";
}

identity anonymous_signing {
  description
    "A TCG recognized anonymous signing algorithm.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2";
}

identity encryption_mode {
  description
    "A TCG recognized encryption mode.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2";
}
```

```
identity method {
  description
    "A TCG recognized method such as a mask generation function.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2";
}

identity object_type {
  description
    "A TCG recognized object type.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 2";
}

identity cryptoprocessor {
  description
    "Base identity identifying a crytoprocessor.";
}

identity tpm12 {
  if-feature "tpm12";
  base cryptoprocessor;
  description
    "Supportable by a TPM1.2.";
  reference
    "TPM1.2-Structures:
     https://trustedcomputinggroup.org/wp-content/uploads/
     TPM-Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf
     TPM_ALGORITHM_ID values, Section 4.8";
}

identity tpm20 {
  if-feature "tpm20";
  base cryptoprocessor;
  description
    "Supportable by a TPM2.";
  reference
    "TPM2.0-Structures:
     https://trustedcomputinggroup.org/wp-content/uploads/
     TPM-Rev-2.0-Part-2-Structures-01.38.pdf";
}

identity TPM_ALG_RSA {
  if-feature "tpm12 or tpm20";
  base tpm12;
  base tpm20;
  base asymmetric;
  base object_type;
```

```
      description
        "RSA algorithm";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         RFC 8017. ALG_ID: 0x0001";
    }

    identity TPM_ALG_TDES {
      if-feature "tpm12";
      base tpm12;
      base symmetric;
      description
        "Block cipher with various key sizes (Triple Data Encryption
         Algorithm, commonly called Triple Data Encryption Standard)
         Note: was banned in TPM1.2 v94";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 18033-3. ALG_ID: 0x0003";
    }

    identity TPM_ALG_SHA1 {
      if-feature "tpm12 or tpm20";
      base hash;
      base tpm12;
      base tpm20;
      description
        "SHA1 algorithm - Deprecated due to insufficient cryptographic
         protection.  However, it is still useful for hash algorithms
         where protection is not required.";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 10118-3. ALG_ID: 0x0004";
    }

    identity TPM_ALG_HMAC {
      if-feature "tpm12 or tpm20";
      base tpm12;
      base tpm20;
      base hash;
      base signing;
      description
        "Hash Message Authentication Code (HMAC) algorithm";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3,
         ISO/IEC 9797-2 and RFC2104. ALG_ID: 0x0005";
    }

    identity TPM_ALG_AES {
```

```
      if-feature "tpm12";
      base tpm12;
      base symmetric;
      description
        "The AES algorithm with various key sizes";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3,
         ISO/IEC 18033-3. ALG_ID: 0x0006";
    }

    identity TPM_ALG_MGF1 {
      if-feature "tpm20";
      base tpm20;
      base hash;
      base method;
      description
        "hash-based mask-generation function";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3,
         IEEE Std 1363-2000 and IEEE Std 1363a-2004.
         ALG_ID: 0x0007";
    }

    identity TPM_ALG_KEYEDHASH {
      if-feature "tpm20";
      base tpm20;
      base hash;
      base object_type;
      description
        "An encryption or signing algorithm using a keyed hash.  These
         may use XOR for encryption or an HMAC for signing and may
         also refer to a data object that is neither signing nor
         encrypting.";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3,
         ALG_ID: 0x0008";
    }

    identity TPM_ALG_XOR {
      if-feature "tpm12 or tpm20";
      base tpm12;
      base tpm20;
      base hash;
      base symmetric;
      description
        "The XOR encryption algorithm.";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3.
```

```
          ALG_ID: 0x000A";
     }

     identity TPM_ALG_SHA256 {
       if-feature "tpm20";
       base tpm20;
       base hash;
       description
         "The SHA 256 algorithm";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
          ISO/IEC 10118-3. ALG_ID: 0x000B";
     }

     identity TPM_ALG_SHA384 {
       if-feature "tpm20";
       base tpm20;
       base hash;
       description
         "The SHA 384 algorithm";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
          ISO/IEC 10118-3. ALG_ID: 0x000C";
     }

     identity TPM_ALG_SHA512 {
       if-feature "tpm20";
       base tpm20;
       base hash;
       description
         "The SHA 512 algorithm";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
          ISO/IEC 10118-3. ALG_ID: 0x000D";
     }

     identity TPM_ALG_NULL {
       if-feature "tpm20";
       base tpm20;
       description
         "NULL algorithm";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3.
          ALG_ID: 0x0010";
     }

     identity TPM_ALG_SM3_256 {
       if-feature "tpm20";
```

```
      base tpm20;
      base hash;
      description
        "The SM3 hash algorithm.";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 10118-3:2018. ALG_ID: 0x0012";
    }

    identity TPM_ALG_SM4 {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      description
        "SM4 symmetric block cipher";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3.
         ALG_ID: 0x0013";
    }

    identity TPM_ALG_RSASSA {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base signing;
      description
        "RFC 8017 Signature algorithm defined in section 8.2
         (RSASSAPKCS1-v1_5)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         RFC 8017.  ALG_ID: 0x0014";
    }

    identity TPM_ALG_RSAES {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base encryption_mode;
      description
        "RFC 8017 Signature algorithm defined in section 7.2
         (RSAES-PKCS1-v1_5)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         RFC 8017. ALG_ID: 0x0015";
    }

    identity TPM_ALG_RSAPSS {
      if-feature "tpm20";
```

```
      base tpm20;
      base asymmetric;
      base signing;
      description
        "Padding algorithm defined in section 8.1 (RSASSA PSS)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         RFC 8017. ALG_ID: 0x0016";
    }

    identity TPM_ALG_OAEP {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base encryption_mode;
      description
        "Padding algorithm defined in section 7.1 (RSASSA OAEP)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         RFC 8017. ALG_ID: 0x0017";
    }

    identity TPM_ALG_ECDSA {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base signing;
      description
        "Signature algorithm using elliptic curve cryptography (ECC)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 14888-3. ALG_ID: 0x0018";
    }

    identity TPM_ALG_ECDH {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base method;
      description
        "Secret sharing using ECC";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         NIST SP800-56A. ALG_ID: 0x0019";
    }

    identity TPM_ALG_ECDAA {
      if-feature "tpm20";
```

```
      base tpm20;
      base asymmetric;
      base signing;
      base anonymous_signing;
      description
        "Elliptic-curve based anonymous signing scheme";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         TCG TPM 2.0 library specification. ALG_ID: 0x001A";
    }

    identity TPM_ALG_SM2 {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base signing;
      base encryption_mode;
      base method;
      description
        "SM2 - depending on context, either an elliptic-curve based,
         signature algorithm, an encryption scheme, or a key exchange
         protocol";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3.
         ALG_ID: 0x001B";
    }

    identity TPM_ALG_ECSCHNORR {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base signing;
      description
        "Elliptic-curve based Schnorr signature";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3.
         ALG_ID: 0x001C";
    }

    identity TPM_ALG_ECMQV {
      if-feature "tpm20";
      base tpm20;
      base asymmetric;
      base method;
      description
        "Two-phase elliptic-curve key";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
```

```
          NIST SP800-56A. ALG_ID: 0x001D";
     }

     identity TPM_ALG_KDF1_SP800_56A {
       if-feature "tpm20";
       base tpm20;
       base hash;
       base method;
       description
         "Concatenation key derivation function";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
          NIST SP800-56A  (approved alternative1) section 5.8.1.
          ALG_ID: 0x0020";
     }

     identity TPM_ALG_KDF2 {
       if-feature "tpm20";
       base tpm20;
       base hash;
       base method;
       description
         "Key derivation function";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
          IEEE 1363a-2004 KDF2 section 13.2. ALG_ID: 0x0021";
     }

     identity TPM_ALG_KDF1_SP800_108 {
       base TPM_ALG_KDF2;
       description
         "A key derivation method";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
          NIST SP800-108 - Section 5.1 KDF. ALG_ID: 0x0022";
     }

     identity TPM_ALG_ECC {
       if-feature "tpm20";
       base tpm20;
       base asymmetric;
       base object_type;
       description
         "Prime field ECC";
       reference
         "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
          ISO/IEC 15946-1. ALG_ID: 0x0023";
     }
```

```
identity TPM_ALG_SYMCIPHER {
  if-feature "tpm20";
  base tpm20;
  base symmetric;
  base object_type;
  description
    "Object type for a symmetric block cipher";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
     TCG TPM 2.0 library specification. ALG_ID: 0x0025";
}

identity TPM_ALG_CAMELLIA {
  if-feature "tpm20";
  base tpm20;
  base symmetric;
  description
    "The Camellia algorithm";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
     ISO/IEC 18033-3. ALG_ID: 0x0026";
}

identity TPM_ALG_SHA3_256 {
  if-feature "tpm20";
  base tpm20;
  base hash;
  description
    "ISO/IEC 10118-3 - the SHA 256 algorithm";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
     NIST PUB FIPS 202. ALG_ID: 0x0027";
}

identity TPM_ALG_SHA3_384 {
  if-feature "tpm20";
  base tpm20;
  base hash;
  description
    "The SHA 384 algorithm";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
     NIST PUB FIPS 202. ALG_ID: 0x0028";
}

identity TPM_ALG_SHA3_512 {
  if-feature "tpm20";
  base tpm20;
```

```
      base hash;
      description
        "The SHA 512 algorithm";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         NIST PUB FIPS 202. ALG_ID: 0x0029";
    }

    identity TPM_ALG_CMAC {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base signing;
      description
        "block Cipher-based Message Authentication Code (CMAC)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 9797-1:2011 Algorithm 5. ALG_ID: 0x003F";
    }

    identity TPM_ALG_CTR {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base encryption_mode;
      description
        "Counter mode";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 10116. ALG_ID: 0x0040";
    }

    identity TPM_ALG_OFB {
      base tpm20;
      base symmetric;
      base encryption_mode;
      description
        "Output Feedback mode";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 10116. ALG_ID: 0x0041";
    }

    identity TPM_ALG_CBC {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base encryption_mode;
```

```
      description
        "Cipher Block Chaining mode";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 10116. ALG_ID: 0x0042";
    }

    identity TPM_ALG_CFB {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base encryption_mode;
      description
        "Cipher Feedback mode";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 10116. ALG_ID: 0x0043";
    }

    identity TPM_ALG_ECB {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base encryption_mode;
      description
        "Electronic Codebook mode";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         ISO/IEC 10116. ALG_ID: 0x0044";
    }

    identity TPM_ALG_CCM {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base signing;
      base encryption_mode;
      description
        "Counter with Cipher Block Chaining-Message Authentication
         Code (CCM)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         NIST SP800-38C. ALG_ID: 0x0050";
    }

    identity TPM_ALG_GCM {
      if-feature "tpm20";
      base tpm20;
```

```
      base symmetric;
      base signing;
      base encryption_mode;
      description
        "Galois/Counter Mode (GCM)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         NIST SP800-38D. ALG_ID: 0x0051";
    }

    identity TPM_ALG_KW {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base signing;
      base encryption_mode;
      description
        "AES Key Wrap (KW)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         NIST SP800-38F. ALG_ID: 0x0052";
    }

    identity TPM_ALG_KWP {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base signing;
      base encryption_mode;
      description
        "AES Key Wrap with Padding (KWP)";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         NIST SP800-38F. ALG_ID: 0x0053";
    }

    identity TPM_ALG_EAX {
      if-feature "tpm20";
      base tpm20;
      base symmetric;
      base signing;
      base encryption_mode;
      description
        "Authenticated-Encryption Mode";
      reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
         NIST SP800-38F. ALG_ID: 0x0054";
    }
```

```
  identity TPM_ALG_EDDSA {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    description
      "Edwards-curve Digital Signature Algorithm (PureEdDSA)";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
       RFC 8032. ALG_ID: 0x0060";
  }
}
<CODE ENDS>
```

Note that not all cryptographic functions are required for use by ietf-tpm-remote-attestation.yang. However, the full definition of Table 3 of [TCG-Algos] will allow use by additional YANG specifications.

3. IANA Considerations

This document registers the following namespace URIs in the [xml-registry] as per [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation

Registrant Contact: The IESG.

XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tcg-algs

Registrant Contact: The IESG.

XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG modules in the registry [yang-parameters] as per Section 14 of [RFC6020]:

Name: ietf-tpm-remote-attestation

Namespace: urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation

Prefix: tpm

Reference: draft-ietf-rats-yang-tpm-charra (RFC form)

Name: ietf-tcg-algs

Namespace: urn:ietf:params:xml:ns:yang:ietf-tcg-algs

Prefix: taa

Reference: draft-ietf-rats-yang-tpm-charra (RFC form)

4. Security Considerations

The YANG module ietf-tpm-remote-attestation.yang specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., _config true_, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., _edit-config_) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes as well as their sensitivity/vulnerability:

Container '/rats-support-structures/attester-supported-algos': 'tpm12-asymmetric-signing', 'tpm12-hash', 'tpm20-asymmetric-signing', and 'tpm20-hash'. All could be populated with algorithms that are not supported by the underlying physical TPM installed by the equipment vendor. A vendor should restrict the ability to configure unsupported algorithms.

Container: '/rats-support-structures/tpms': 'name': Although shown as 'rw', it is system generated. Therefore, it should not be possible for an operator to add or remove a TPM from the configuration.

'tpm20-pcr-bank': It is possible to configure PCRs for extraction which are not being extended by system software. This could unnecessarily use TPM resources.

'certificates': It is possible to provision a certificate which does not correspond to an Attestation Identity Key (AIK) within the TPM 1.2, or an Attestation Key (AK) within the TPM 2.0 respectively. In such a case, calls to an RPC requesting this specific certificate could result in either no response or a response for an unexpected TPM.

RPC 'tpm12-challenge-response-attestation': The receiver of the RPC response must verify that the certificate is for an active AIK, i.e., the certificate has been confirmed by a third party as being able to support Attestation on the targeted TPM 1.2.

RPC 'tpm20-challenge-response-attestation': The receiver of the RPC response must verify that the certificate is for an active AK, i.e., the private key confirmation of the quote signature within the RPC response has been confirmed by a third party to belong to an entity legitimately able to perform Attestation on the targeted TPM 2.0.

RPC 'log-retrieval': Requesting a large volume of logs from the Attester could require significant system resources and create a denial of service.

Information collected through the RPCs above could reveal that specific versions of software and configurations of endpoints that could identify vulnerabilities on those systems. Therefore, RPCs should be protected by NACM [RFC8341] with a default setting of deny-all to limit the extraction of attestation data by only authorized Verifiers.

For the YANG module ietf-tcg-algs.yang, please use care when selecting specific algorithms. The introductory section of [TCG-Algos] highlights that some algorithms should be considered legacy, and recommends implementers and adopters diligently evaluate available information such as governmental, industrial, and academic research before selecting an algorithm for use.

5. References

5.1. Normative References

[bios-log] "TCG PC Client Platform Firmware Profile Specification, Section 9.4.5.2", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf>.

[BIOS-Log-Event-Type] "TCG PC Client Platform Firmware Profile Specification", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TCG_PCClient_PFP_r1p05_v23_pub.pdf>.

[cel] "Canonical Event Log Format, Section 4.3", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TCG_IWG_CEL_v1_r0p41_pub.pdf>.

[I-D.ietf-netconf-keystore]
Watsen, K., "A YANG Data Model for a Keystore and Keystore Operations", Work in Progress, Internet-Draft, draft-ietf-netconf-keystore-32, 8 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-keystore-32>.

[I-D.ietf-rats-tpm-based-network-device-attest]
Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", Work in Progress, Internet-Draft, draft-ietf-rats-tpm-based-network-device-attest-14, 22 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-tpm-based-network-device-attest-14>.

[IEEE-Std-1363-2000]
"IEEE 1363-2000 - IEEE Standard Specifications for Public-Key Cryptography", n.d., <https://standards.ieee.org/standard/1363-2000.html>.

[IEEE-Std-1363a-2004]
"1363a-2004 - IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques", n.d., <https://ieeexplore.ieee.org/document/1335427>.

[ISO-IEC-10116]
"ISO/IEC 10116:2017 - Information technology", n.d., <https://www.iso.org/standard/64575.html>.

[ISO-IEC-10118-3]
"Dedicated hash-functions - ISO/IEC 10118-3:2018", n.d., <https://www.iso.org/standard/67116.html>.

[ISO-IEC-14888-3]
"ISO/IEC 14888-3:2018 - Digital signatures with appendix", n.d., <https://www.iso.org/standard/76382.html>.

[ISO-IEC-15946-1]
"ISO/IEC 15946-1:2016 - Information technology", n.d., <https://www.iso.org/standard/65480.html>.

[ISO-IEC-18033-3]
"ISO/IEC 18033-3:2010 - Encryption algorithms", n.d., <https://www.iso.org/standard/54531.html>.

[ISO-IEC-9797-1]
"Message Authentication Codes (MACs) - ISO/IEC 9797-1:2011", n.d., <https://www.iso.org/standard/50375.html>.

[ISO-IEC-9797-2]
"Message Authentication Codes (MACs) - ISO/IEC 9797-2:2011", n.d., <https://www.iso.org/standard/51618.html>.

[NIST-PUB-FIPS-202]
"SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", n.d., <https://csrc.nist.gov/publications/detail/fips/202/final>.

[NIST-SP800-108]
"Recommendation for Key Derivation Using Pseudorandom Functions", n.d., <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf>.

[NIST-SP800-38C]
"Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality", n.d., <https://csrc.nist.gov/publications/detail/sp/800-38c/final>.

[NIST-SP800-38D]
"Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", n.d., <https://csrc.nist.gov/publications/detail/sp/800-38d/final>.

[NIST-SP800-38F]
"Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping", n.d., <https://csrc.nist.gov/publications/detail/sp/800-38f/final>.

[NIST-SP800-56A]
"Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", n.d., <https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

[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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/rfc/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/rfc/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/rfc/rfc6242>.

[RFC6933] Bierman, A., Romascanu, D., Quittek, J., and M. Chandramouli, "Entity MIB (Version 4)", RFC 6933, DOI 10.17487/RFC6933, May 2013, <https://www.rfc-editor.org/rfc/rfc6933>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/rfc/rfc6991>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/rfc/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/rfc/rfc8040>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/rfc/rfc8341>.

[RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <https://www.rfc-editor.org/rfc/rfc8348>.

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

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

[TCG-Algos] "TCG Algorithm Registry", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TCG-_Algorithm_Registry_r1p32_pub.pdf>.

[TPM1.2] TCG, "TPM 1.2 Main Specification", 2 October 2003, <https://trustedcomputinggroup.org/resource/tpm-main-specification/>.

[TPM1.2-Commands] "TPM Main Part 3 Commands", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-3-Commands_v1.2_rev116_01032011.pdf>.

[TPM1.2-Structures] "TPM Main Part 2 TPM Structures", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf>.

[TPM2.0] TCG, "TPM 2.0 Library Specification", 15 March 2013, <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

[TPM2.0-Arch] "Trusted Platform Module Library - Part 1: Architecture", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf>.

[TPM2.0-Key] TCG, "TPM 2.0 Keys for Device Identity and Attestation, Rev12", 8 October 2021, <https://trustedcomputinggroup.org/wp-content/uploads/TPM-2p0-Keys-for-Device-Identity-and-Attestation_v1_r12_pub10082021.pdf>.

[TPM2.0-Structures] "Trusted Platform Module Library - Part 2: Structures", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-2-Structures-01.38.pdf>.

[UEFI-Secure-Boot] "Unified Extensible Firmware Interface (UEFI) Specification Version 2.9 (March 2021), Section 32.1 (Secure Boot)", n.d., <https://uefi.org/sites/default/files/resources/UEFI_Spec_2_9_2021_03_18.pdf>.

5.2. Informative References

[I-D.ietf-rats-reference-interaction-models] Birkholz, H., Eckel, M., Pan, W., and E. Voit, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-ietf-rats-reference-interaction-models-08, 10 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-reference-interaction-models-08>.

[IMA-Kernel-Source] "Linux Integrity Measurement Architecture (IMA): Kernel Sourcecode", n.d., <https://github.com/torvalds/linux/blob/df0cc57e057f18e44dac8e6c18aba47ab53202f9/security/integrity/ima/>.

[NIST-915121]
"True Randomness Cant be Left to Chance: Why entropy is important for information security", n.d., <https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915121>.

[xml-registry]
"IETF XML Registry", n.d., <https://www.iana.org/assignments/xml-registry/xml-registry.xhtml>.

[yang-parameters]
"YANG Parameters", n.d., <https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml>.

Appendix A. Integrity Measurement Architecture (IMA)

IMA extends the principles of Measured Boot [TPM2.0-Arch] and Secure Boot [UEFI-Secure-Boot] to the Linux operating system, applying it to operating system applications and files. IMA has been part of the Linux integrity subsystem of the Linux kernel since 2009 (kernel version 2.6.30). The IMA mechanism represented by the YANG module in this specification is rooted in the kernel version 5.16 [IMA-Kernel-Source]. IMA enables the protection of system integrity by collecting (commonly referred to as measuring) and storing measurements (called Claims in the context of IETF RATS) of files before execution so that these measurements can be used later, at system runtime, in remote attestation procedures. IMA acts in support of the appraisal of Evidence (which includes measurement Claims) by leveraging Reference Values stored in extended file attributes.

In support of the appraisal of Evidence, IMA maintains an ordered list (with no duplicates) of measurements in kernel-space, the Stored Measurement Log (SML), for all files that have been measured before execution since the operating system was started. Although IMA can be used without a TPM, it is typically used in conjunction with a TPM to anchor the integrity of the SML in a hardware-protected secure storage location, i.e., Platform Configuration Registers (PCRs) provided by TPMs. IMA provides the SML in both binary and ASCII representations in the Linux security file system _securityfs_ (/sys/kernel/security/ima/).

IMA templates define the format of the SML, i.e., which fields are included in a log record. Examples are file path, file hash, user ID, group ID, file signature, and extended file attributes. IMA comes with a set of predefined template formats and also allows a

custom format, i.e., a format consisting of template fields supported by IMA. Template usage is typically determined by boot arguments passed to the kernel. Alternatively, the format can also be hard-coded into custom kernels. IMA templates and fields are extensible in the kernel source code. As a result, more template fields can be added in the future.

IMA policies define which files are measured using the IMA policy language. Built-in policies can be passed as boot arguments to the kernel. Custom IMA policies can be defined once during runtime or be hard-coded into a custom kernel. If no policy is defined, no measurements are taken and IMA is effectively disabled.

A comprehensive description of the content fields in native Linux IMA TLV format can be found in Table 16 of the Canonical Event Log (CEL) specification [cel]. The CEL specification also illustrates the use of templates to enable extended or customized IMA TLV formats in Section 5.1.6.

Appendix B. IMA for Network Equipment Boot Logs

Network equipment can generally implement similar IMA-protected functions to generate measurements (Claims) about the boot process of a device and enable corresponding remote attestation. Network Equipment Boot Logs combine the measurement and logging of boot components and operating system components (executables and files) into a single log file in a format identical to the IMA format. Note that the format used for logging measurement of boot components in this scheme differs from the boot logging strategy described elsewhere in this document.

During the boot process of the network device, i.e., from BIOS to the end of the operating system and user-space, all files executed can be measured and logged in the order of their execution. When the Verifier initiates a remote attestation process (e.g., challenge-response remote attestation as defined in this document), the network equipment takes on the role of an Attester and can convey to the Verifier Claims that comprise the measurement log as well as the corresponding PCR values (Evidence) of a TPM.

The Verifier can appraise the integrity (compliance with the Reference Values) of each executed file by comparing its measured value with the Reference Value. Based on the execution order, the Verifier can compute a PCR Reference Value (by replaying the log) and compare it to the Measurement Log Claims obtained in conjunction with the PCR Evidence to assess their trustworthiness with respect to an intended operational state.

Network equipment usually executes multiple components in parallel. This holds not only during the operating system loading phase, but also even during the BIOS boot phase. With this measurement log mechanism, network equipment can take on the role of an Attester, proving to the Verifier the trustworthiness of its boot process. Using the measurement log, Verifiers can precisely identify mismatching log entries to infer potentially tampered components.

This mechanism also supports scenarios that modify files on the Attester that are subsequently executed during the boot phase (e.g., updating/patching) by simply updating the appropriate Reference Values in Reference Integrity Manifests that inform Verifiers about how an Attester is composed.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@ietf.contact

Michael Eckel
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: michael.eckel@sit.fraunhofer.de

Shwetha Bhandari
ThoughtSpot
Email: shwetha.bhandari@thoughtspot.com

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Bill Sulzen
Cisco Systems
Email: bsulzen@cisco.com

Liang Xia (Frank)
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing
Jiangsu, 210012
China
Email: Frank.Xialiang@huawei.com

Tom Laffey
Hewlett Packard Enterprise
Email: tom.laffey@hpe.com

Guy C. Fedorkow
Juniper Networks
10 Technology Park Drive
Westford
Email: gfedorkow@juniper.net