

RATS Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 7, 2020

H. Birkholz  
Fraunhofer SIT  
M. Wiseman  
GE Global Research  
H. Tschofenig  
ARM Ltd.  
N. Smith  
Intel  
M. Richardson  
Sandelman Software Works  
November 04, 2019

Remote Attestation Procedures Architecture  
draft-birkholz-rats-architecture-03

Abstract

An entity (a relying party) requires a source of truth and evidence about a remote peer to assess the peer's trustworthiness. The evidence is typically a believable set of claims about its host, software or hardware platform. This document describes an architecture for such remote attestation procedures (RATS).

Status of This Memo

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

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

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

This Internet-Draft will expire on May 7, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Motivation . . . . .	3
1.2. Opportunities . . . . .	3
1.3. Overview of Document . . . . .	4
1.4. RATS in a Nutshell . . . . .	5
1.5. Remote Attestation Workflow . . . . .	5
1.6. Message Flows . . . . .	7
1.6.1. Passport Model . . . . .	7
1.6.2. Background Check . . . . .	8
2. Terminology . . . . .	9
3. Reference use cases . . . . .	10
3.1. Device Capabilities/Firmware Attestation . . . . .	11
3.2. IETF TEEP WG Use-Case . . . . .	11
3.3. Safety Critical Systems . . . . .	12
3.4. Virtualized Multi-Tenant Hosts . . . . .	12
3.5. Cryptographic Key Attestation . . . . .	13
3.6. Geographic Evidence . . . . .	13
3.7. Device Provenance Attestation . . . . .	14
4. Conceptual Overview . . . . .	14
4.1. Two Types of Environments . . . . .	15
4.2. Evidence Creation Prerequisites . . . . .	16
4.3. Trustworthiness . . . . .	17
4.4. Workflow . . . . .	17
4.5. Interoperability between RATS . . . . .	18
5. RATS Architecture . . . . .	18
5.1. Goals . . . . .	18
5.2. Attestation Principles . . . . .	18
5.3. Attestation Workflow . . . . .	19
5.3.1. Roles . . . . .	19
5.3.2. Role Messages . . . . .	20
5.4. Principals (Entities?) - Containers for the Roles . . . . .	22
6. Privacy Considerations . . . . .	23
7. Security Considerations . . . . .	23
8. Acknowledgements . . . . .	23
9. References . . . . .	23
9.1. Normative References . . . . .	24
9.2. Informative References . . . . .	24
Authors' Addresses . . . . .	25

## 1. Introduction

Remote Attestation provides a way for an entity (the Relying Party) to determine the health and provenance of an endpoint/host (the Attester). Knowledge of the health of the endpoint allows for a determination of trustworthiness of the endpoint.

### 1.1. Motivation

The IETF has long spent it's time focusing on threats to the communication channel (see [RFC3552] and [DOLEV-YAO]), assuming that endpoints could be trusted and were under the observation of trusted, well-trained professionals. This assumption has not been true since the days of the campus mini-computer. For some time after the desktop PC became ubiquitous, the threat to the endpoints has been dealt with as an internal matter, with generally poor results. Enterprises have done some deployment of Network Endpoint Assessment ([RFC5209]) to assess the security posture about an endpoint, but it has not been ubiquitous.

The movement towards personal mobile devices ("smartphones") and the continuing threat from unmanaged residential desktops has resulted in a renewed interest in standardizing internet-scale endpoint remote attestation. Additionally, the rise of the Internet of Things (IoT) has made this issue even more critical: some skeptics have even renamed it to the Internet of Threats [iothreats] :-). IoT devices have poor or non-existent user interfaces, as such as there are not even good ways to assess the health of the devices manually: a need to determine the health via remote attestation is now critical.

In addition to the health of the device, knowledge of its provenance helps to determine the level of trust, and prevents attacks to the supply chain.

### 1.2. Opportunities

The Trusted Platform Module (TPM) is now a commonly available peripheral on many commodity compute platforms, both servers and desktops. Smartphones commonly have either an actual TPM, or have the ability to emulate one in software running in a Trusted Execution Environment [I-D.ietf-teep-architecture]. There are now few barriers to creating a standards-based system for remote attestation procedures.

A number of niche solutions have emerged that provide for use-case specific remote attestation, but none have the generality needed to be used across the Internet.

### 1.3. Overview of Document

The architecture described in this document (along with the accompanying solution and reference documents) enables the use of common formats for communicating Claims about an Attester to a Relying Party. [FIXME Attester? Flows? To what end?]

Existing transports were not designed to carry attestation Claims. It is therefore necessary to design serializations of Claims that fit into a variety of transports, for instance: X.509 certificates, TLS negotiations, YANG modules or EtherNet/IP. There are also new, greenfield uses for remote attestation. Transport and serialization of these can be done without retrofitting. This is (will be) described in [INSERT reference to adopted document on transport].

While it is not anticipated that the existing niche solutions described in the use cases section Section 3 will exchange claims directly, the use of a common format enables common code. As some of the code needs to be in intentionally hard to modify trusted modules, the use of a common formats and transfer protocols significantly reduces the cost of adoption to all parties. This commonality also significantly reduces the incidence of critical bugs.

In some environments the collection of Evidence by the Attester to be provided to the Verifier is part of an existing protocol: this document does not change that, rather embraces those legacy mechanisms as part of the specification. This is an evolutionary path forward, not revolutionary. Yet in other greenfield environments, there is a desire to have a standard for Evidence as well as for Attestation Results, and this architecture outlines how that is done.

This introduction gives an overview of the message flows and roles involved. Following this, is a terminology section that is referenced normatively by other documents and is a key part of this document. There is then a section on use cases and how they leverage the roles and workflows described.

In this document, terms defined within this document are consistently Capitalized [work in progress. please raise issues, if there are Blatant inconsistencies].

Current verticals that use remote attestation include:

- o The Trusted Computing Group "Network Device Attestation Workflow" [I-D.fedorkow-rats-network-device-attestation]
- o Android Keystore [keystore]

- o Fast Identity Online (FIDO) Alliance attestation [fido]
- o A number of Intel SGX niche systems based upon OTRP.

#### 1.4. RATS in a Nutshell

1. Remote Attestation message flows typically convey Claims that contain the trustworthiness properties associated with an Attested Environment (Evidence).
2. A corresponding provisioning message flows conveys Reference trustworthiness claims that can be compared with attestation Evidence. Reference Values typically consist of firmware or software digests and details about what makes the attesting module a trusted source of Evidence.
3. Relying Parties are performing tasks such as managing a resource, controlling access, and/or managing risk. Attestation Results helps Relying Parties determine levels of trust.

#### 1.5. Remote Attestation Workflow

The logical information flow is from Attester to Verifier to Relying Party. There are variations presented below on how this integrates into actual protocols.

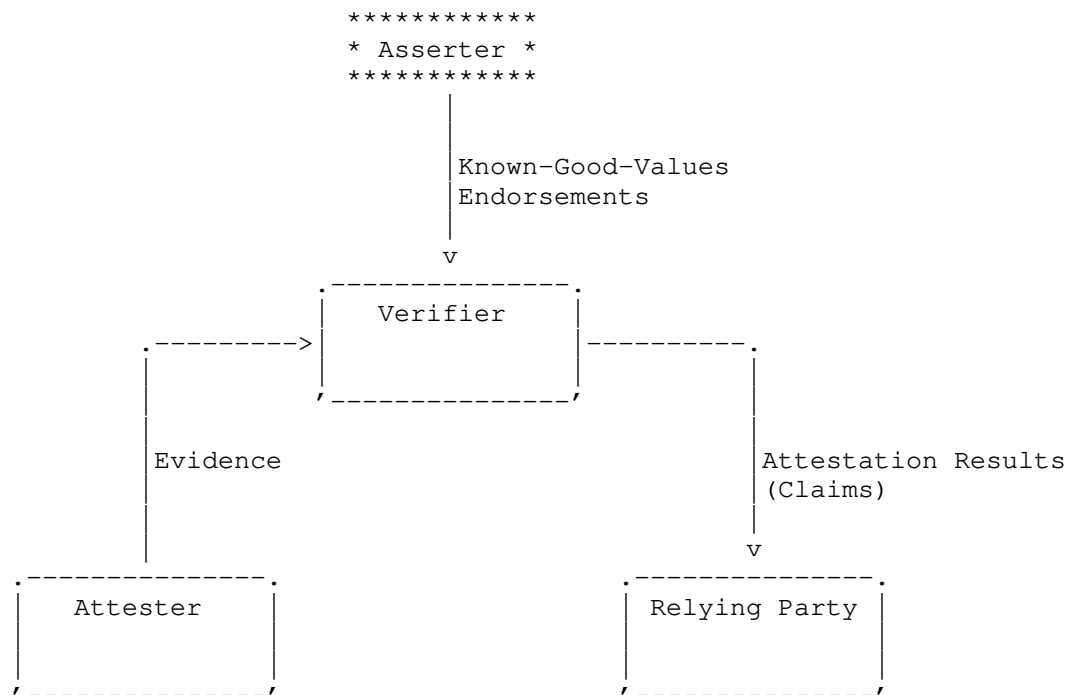


Figure 1: RATS Workflow

In the architecture shown above, specific content items (payload conveyed in message flows) are identified:

- o Evidence is as set of believable Claims about distinguishable Environments made by an Attester.
- o Known-Good-Values are reference Claims used to appraise Evidence by an Verifier.
- o Endorsements are reference Claims about the type of protection that enables an Attester to create believable Evidence. Endorsements enable trust relationships towards system components or environments Evidence cannot be created for by an Attester.
- o Attestation Results are the output from the appraisal of Evidence, Known-Good-Values and Endorsements and are consumed by Relying Parties.

Attestation Results are the output of RATS.

Assessment of Attestation Results is be multi-faceted and out-of-scope for the architecture.

If appropriate Endorsements about the Attester are available, Known-Good-Values about the Attester are available, and if the Attester is capable of creating believable Evidence - then the Verifier is able to create Attestation Results that enable Relying Parties to establish a level of confidence in the trustworthiness of the Attester.

The Asserter role and the format for Known-Good-Values and Endorsements are not subject to standardization at this time. The current verticals already include provisions for encoding and/or distributing these objects.

#### 1.6. Message Flows

Two distinct flows have been identified for passage of Evidence and production of Attestation Results. It is possible that there are additional situations which are not captured by these two flows.

##### 1.6.1. Passport Model

In the Passport Model message flow the Attester provides it's Evidence directly to the Verifier. The Verifier will evaluate the Evidence and then sign an Attestation Result. This Attestation Result is returned to the Attester, and it is up to the Attester to communicate the Attestation Result (potentially including the Evidence, if disclosable) to the Relying Party.

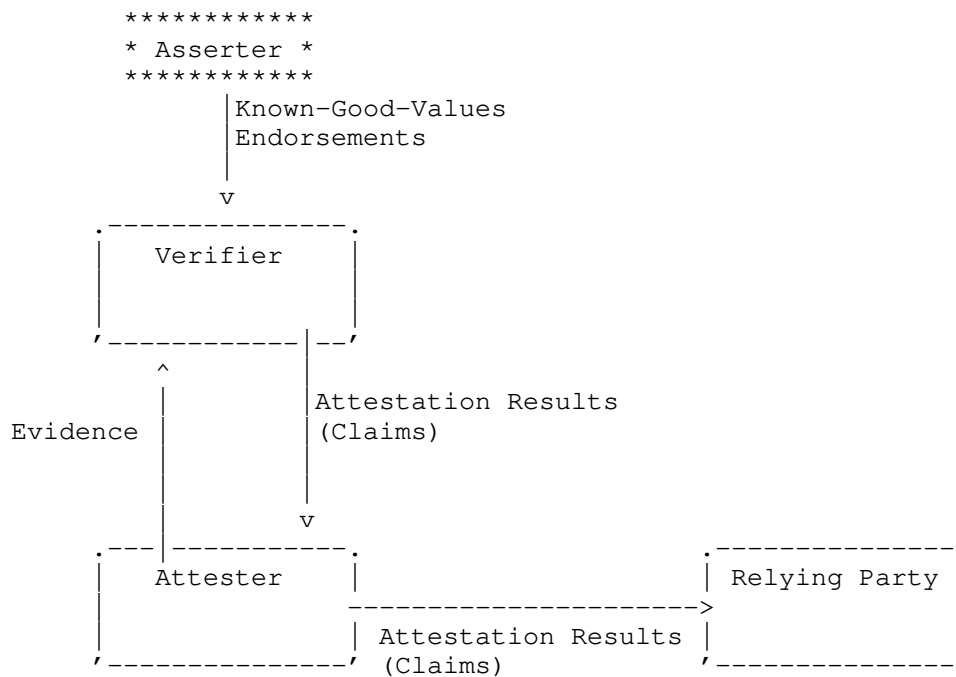


Figure 2: RATS Passport Flow

This flow is named in this way because of the resemblance of how Nations issue Passports to their citizens. The nature of the Evidence that an individual needs to provide to it's local authority is specific to the country involved. The citizen retains control of the resulting document and presents it to other entities when it needs to assert a citizenship or identity claim.

#### 1.6.2. Background Check

In the Background-Check message flow the Attester provides it's Evidence to the Relying Party. The Relying Party sends this evidence to a Verifier of its choice. The Verifier will evaluate the Evidence and then sign an Attestation Result. This Attestation Result is returned to the Relying Party, which processes it directly.



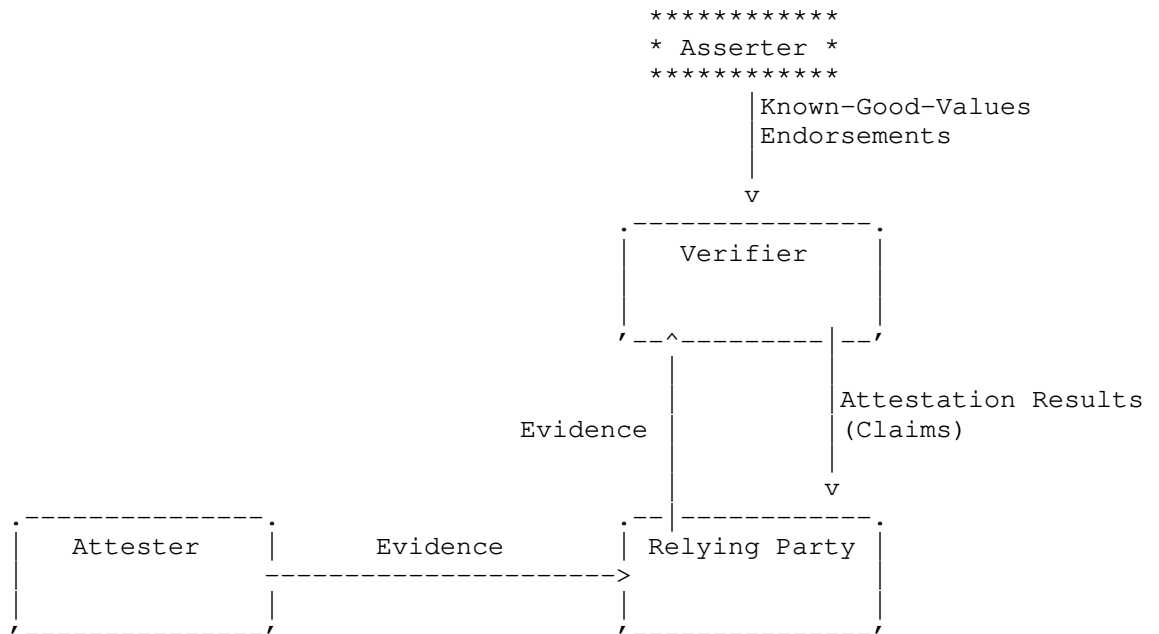


Figure 3: RATS Background Check Flow

This flow is named in this way 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.

## 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.

**Appraisal:** A Verifier process that compares Evidence to Reference values while taking into account Endorsements and produces Attestation Results.

**Asserter:** See Section 5.3.1.2.

**Attester:** See Section 5.3.1.1.

**Attested Environment:** A target environment that is observed or controlled by an Attesting Environment.

**Attesting Environment:** An environment capable of making trustworthiness Claims about an Attested Environment.

**Background-Check Message Flow:** An attestation workflow where the Attester provides Evidence to a Relying Party, who consults one or more Verifiers who supply Attestation Results to the Relying Party. See Section 1.6.2.

**Claim:** A statement about the construction, composition, validation or behavior of an Entity that affects trustworthiness. Evidence, Reference Values and Attestation Results are expressions that consists of one or more Claims.

**Conveyance:** The process of transferring Evidence, Reference Values and Attestation Results between Entities participating in attestation workflow.

**Entity:** A device, component (see System Component [RFC4949]), or environment that implements one or more Roles.

**Evidence:** See Section 5.3.2.1.

**Passport Message Flow:** An attestation workflow where the Attester provides Evidence to a Verifier who returns Attestation Results that are then forwarded to one or more Relying Parties. See Section 1.6.1.

**Reference Values:** See Section 5.3.2.2. Also referred to as Known-Good-Values.

**Relying Party:** See Section 5.3.1.4.

**Attestation Results:** See Section 5.3.2.3.

**Role:** A function or process in an attestation workflow, typically described by: Attester, Verifier, Relying Party and Asserter.

**Verifier:** See Section 5.3.1.3.

### 3. Reference use cases

This section provides an overview of a number of distinct use cases that benefit from a standardized claim format. In addition to outlining the user, the specific message flow is identified from among the flows detailed in Section 1.6.

### 3.1. Device Capabilities/Firmware Attestation

This is a large category of claims that includes a number of subcategories, not detailed here.

Use case name: Device Identity

Who will use it: Network Operators, larger enterprises

Attester: varies

Message Flow: sometimes passport and sometimes background check

Relying Party: varies

Description: Network operators want a trustworthy report of identity and version of information of the hardware and software on the machines attached to their network. The process starts with some kind of Root of Trust that provides device identity and protected storage for measurements. The mechanism performs a series of measurements, and expresses this with an attestation as to the hardware and firmware/software which is running.

This is a general description for which there are many specific use cases, including [I-D.fedorkow-rats-network-device-attestation] section 1.2, "Software Inventory"

### 3.2. IETF TEEP WG Use-Case

Use case name: TAM validation

Who will use it: The TAM server

Message Flow: background check

Attester: Trusted Execution Environment (TEE)

Relying Party: end-application

Description: The "Trusted Application Manager (TAM)" server wants to verify the state of a TEE, or applications in the TEE, of a device. The TEE attests to the TAM, which can then decide whether to install sensitive data in the TEE, or whether the TEE is out of compliance and the TAM needs to install updated code in the TEE to bring it back into compliance with the TAM's policy.

### 3.3. Safety Critical Systems

Use case name: Safety Critical Systems

Who will use it: Power plants and other systems that need to assert their current state, but which can not accept any inputs from the outside. The corollary system is a black-box (such as in an aircraft), which needs to log the state of a system, but which can never initiate a handshake.

Message Flow: background check

Attester: web services and other sources of status/sensor information

Relying Party: open

Claims used as Evidence: the beginning and ending time as endorsed by a Time Stamp Authority, represented by a time stamp token. The real time clock of the system itself. A Root of Trust for time; the TPM has a relative time from startup.

Description: These requirements motivate the creation of the Time-Base Unidirectional Attestation (TUDA) [I-D.birkholz-rats-tuda], the output of TUDA is typically a secure audit log, where freshness is determined by synchronization to a trusted source of external time.

The freshness is preserved in the Evidence by the use of a Time Stamp Authority (TSA) which provides Time Stamp Tokens (TST).

### 3.4. Virtualized Multi-Tenant Hosts

Use case name: Multi-Tenant Hosts

Who will use it: Virtual machine systems

Message Flow: passport

Attester: virtual machine hypervisor

Relying Party: network operators

Description: The host system will do verification as per Section 3.1

The tenant virtual machines will do verification as per Section 3.1.

The network operator wants to know if the system \_as a whole\_ is free of malware, but the network operator is not allowed to know who the tenants are.

This is contrasted to the Chassis + Line Cards case (To Be Defined: TBD).

Multiple Line Cards, but a small attestation system on the main card can combine things together. This is a kind of proxy.

### 3.5. Cryptographic Key Attestation

Cryptographic Attestation includes subcategories such as Device Type Attestation (the FIDO use case), and Key storage Attestation (the Android Keystore use case), and End-User Authorization.

Use case name: Key Attestation

Who will use it: network authentication systems

Message Flow: passport

Attester: device platform

Relying Party: internet peers

Description: The relying party wants to know how secure a private key that identifies an entity is. Unlike the network attestation, the relying party is not part of the network infrastructure, nor do they necessarily have a business relationship (such as ownership) over the end device.

The Device Type Attestation is provided by a Firmware TPM performing the Verifier function, creating Attestation Results that indicate a particular model/type of device. In TCG terms, this is called Implicit Attestation, in this case the Attested Environment is the (smartphone) Rich Execution Environment (REE) ([I-D.ietf-teep-architecture] section 2), and the Attesting Environment is within the TEE.

### 3.6. Geographic Evidence

Use case name: Location Evidence

Who will use it: geo-fenced systems

Message Flow: passport (probably)

Attester: secure GPS system(s)

Relying Party: internet peers

Description: The relying party wants to know the physical location (on the planet earth, using a geodetic system) of the device. This may be provided directly by a GPS/GLONASS/BeiDou/Galileo module that is incorporated into a TPM. This may also be provided by collecting other proximity messages from other device that the relying party can form a trust relationship with.

### 3.7. Device Provenance Attestation

Use case name: RIV - Device Provenance

Who will use it: Industrial IoT devices

Message Flow: passport

Attester: network management station

Relying Party: a network entity

Description: A newly manufactured device needs to be onboarded into a network where many if not all device management duties are performed by the network owner. The device owner wants to verify the device originated from a legitimate vendor. A cryptographic device identity such as an IEEE802.1AR is embedded during manufacturing and a certificate identifying the device is delivered to the owner onboarding agent. The device authenticates using its 802.1AR IDevID to prove it originated from the expected vendor.

The device chain of custody from the original device manufacturer to the new owner may also be verified as part of device provenance attestation. The chain of custody history may be collected by a cloud service or similar capability that the supply chain and owner agree to use.

[I-D.fedorkow-rats-network-device-attestation] section 1.2 refers to this as "Provable Device Identity", and section 2.3 details the parties.

## 4. Conceptual Overview

In network protocol exchanges, it is often the case that one entity (a Relying Party) requires an assessment of the trustworthiness of a remote entity (an Attester or specific system components [RFC4949])

thereof). Remote Attestation procedures (RATS) enable Relying Parties to establish a level of confidence in the trustworthiness of Attesters through the

- o Creation,
- o Conveyance, and
- o Appraisal

of attestation Evidence.

**Qualities of Evidence:** Evidence is composed of Claims about trustworthiness (the set of Claims is unbounded). The system characteristics of Attesters – the Environments they are composed of, and their continuous development – have an impact on the veracity of trustworthiness Claims included in valid Evidence.

Valid Evidence about the intactness of an Attester must be impossible to create by an untrustworthy or compromised Environment of an Attester.

**Qualities of Environments:** The resilience of Environments that are part of an Attester can vary widely – ranging from those highly resistant to attacks to those having little or no resistance to attacks. Configuration options, if set poorly, can result in a highly resistant environment being operationally less resistant. When a trustworthy Environment changes, it is possible that it transitions from being trustworthy to being untrustworthy.

An untrustworthy or compromised Environment must never be able to create valid Evidence expressing the intactness of an Attester.

The architecture provides a framework for anticipating when a relevant change with respect to a trustworthiness attribute occurs, what exactly changed and how relevant it is. The architecture also creates a context for enabling an appropriate response by applications, system software and protocol endpoints when changes to trustworthiness attributes do occur.

Detailed protocol specifications for message flows are defined in separate documents.

#### 4.1. Two Types of Environments

An Attester produces Evidence about its own integrity, which means "it measures itself". To disambiguate this recursive or circular

looking relationships, two types of Environments inside an Attester are distinguished:

The attest-ED Environments and the attest-ING Environments.

Attested Environments are measured. They provide the raw values and the information to be represented in Claims and ultimately expressed as Evidence.

Attesting Environments conduct the measuring. They collect the Claims, format them appropriately, and typically use key material and cryptographic functions, such as signing or cipher algorithms, to create Evidence.

Attesting Environments use system components that have to be trusted. As a result, Evidence includes Claims about the Attested and the Attesting Environments. Claims about the Attested Environments are appraised using Reference Values and Claims about the Attesting Environments are appraised using Endorsements. It is not mandated that both Environments have to be separate, but it is highly encouraged. Examples of separated Environments that can be used as Attesting Environments include: Trusted Execution Environments (TEE), embedded Secure Elements (eSE), or Hardware Security Modules (HSM).

In summary, the majority of the creation of evidence can take place in an Attested Environments. Exemplary duties include the collection and formatting of Claim values, or the trigger for creating Evidence. A trusted sub-set of the creation of evidence can take place in an Attesting Environment, that provide special protection with respect to key material, identity documents, or primitive functions to create the Evidence itself.

#### 4.2. Evidence Creation Prerequisites

One or more Environments that are part of an Attester must be able to conduct the following duties in order to create Evidence:

- o monitoring trustworthiness attributes of other Environments,
- o collecting trustworthiness attributes and create Claims about them,
- o serialize Claims using interoperable representations,
- o provide integrity protection for the sets of Claims, and
- o add appropriate attestation provenance attributes about the sets of Claims.



#### 4.3. Trustworthiness

The trustworthiness of an Attester and therefore the believability of the Evidence it creates relies on the protection methods in place to shield and restrict the use of key material and the duties conducted by the Attesting Environment. In order to assess trustworthiness effectively, it is mandatory to understand the trustworthiness properties of the environments of an Attester. The corresponding appraisal of Evidence that leads to such an assessment of trustworthiness is the duty of a Verifier.

Trusting the assessment of a Verifier might come from trusting the Verifier's key material (direct trust), or trusting an Entity that the Verifier is associated with via a certification path (indirect trust).

The trustworthiness of corresponding Attestation Results also relies on trust towards manufacturers and those manufacturer's hardware in order to assess the integrity and resilience of that manufacturer's devices.

A stronger level of security comes when information can be vouched for by hardware or by (unchangeable) firmware, especially if such hardware is physically resistant to hardware tampering. The component that is implicitly trusted is often referred to as a Root of Trust.

#### 4.4. Workflow

The basic function of RATS is creation, conveyance and appraisal of attestation Evidence. An Attester creates attestation Evidence that are conveyed to a Verifier for appraisal. The appraisals compare Evidence with expected Known-Good-Values obtained from Asserters (e.g. Principals that are Supply Chain Entities). There can be multiple forms of appraisal (e.g., software integrity verification, device composition and configuration verification, device identity and provenance verification). Attestation Results are the output of appraisals. Attestation Results are signed and conveyed to Relying Parties. Attestation Results provide the basis by which the Relying Party may determine a level of confidence to place in the application data or operations that follow.

The architecture defines attestation Roles: Attester, Verifier, Asserter and Relying Party. Roles exchange messages, but their structure is not defined in this document. The detailed definition of the messages is in an appropriate document, such as [I-D.ietf-rats-eat] or other protocols to be defined. Roles can be combined in various ways into Principals, depending upon the needs of

the use case. Information Model representations are realized as data structure and conveyance protocol specifications.

#### 4.5. Interoperability between RATS

The RATS architecture anticipates use of information modeling techniques that describe computing structures - their components/ computational elements and corresponding capabilities - so that verification operations may rely on the information model as an interoperable way to navigate the structural complexity.

### 5. RATS Architecture

#### 5.1. Goals

RATS architecture has the following goals:

- o Enable semantic interoperability of attestation semantics through information models about computing environments and trustworthiness.
- o Enable data structure interoperability related to claims, endpoint composition / structure, and end-to-end integrity and confidentiality protection mechanisms.
- o Enable programmatic assessment of trustworthiness. (Note: Mechanisms that manage risk, justify a level of confidence, or determine a consequence of an attestation result are out of scope).
- o Provide the building blocks, including Roles and Principals that enable the composition of service-chains/hierarchies and workflows that can create and appraise evidence about the trustworthiness of devices and services.
- o Use-case driven architecture and design (see [I-D.richardson-rats-usecases] and Section 3)
- o Terminology conventions that are consistently applied across RATS specifications.
- o Reinforce trusted computing principles that include attestation.

#### 5.2. Attestation Principles

Specifications developed by the RATS working group apply the following principles:

- o Freshness - replay of previously asserted Claims about an Attested Environment can be detected.
- o Identity - the Attesting Environment is identifiable (non-anonymous).
- o Context - the Attested Environment is well-defined (unambiguous).
- o Provenance - the origin of Claims with respect to the Attested and Attesting Environments are known.
- o Validity - the expected lifetime of Claims about an Attested Environment is known.
- o Veracity - the believability (level of confidence) of Claims is based on verifiable proofs.

### 5.3. Attestation Workflow

Attestation workflow helps a Relying Party make better decisions by providing insight about the trustworthiness of endpoints participating in a distributed system. The workflow consists primarily of four roles; Relying Party, Verifier, Attester and Asserter. Attestation messages contain information useful for appraising the trustworthiness of an Attester endpoint and informing the Relying Party of the appraisal result.

This section details the primary roles of an attestation workflow and the messages they exchange.

#### 5.3.1. Roles

An endpoint system (a.k.a., Entity) may implement one or more attestation Roles to accommodate a variety of possible message flows. Exemplary message flows are described in Section 1.6.1 and Section 1.6.2. Role messages are secured by the Entity that generated it. Entities possess credentials (e.g., cryptographic keys) that authenticate, integrity protect and optionally confidentiality protect attestation messages.

##### 5.3.1.1. Attester

The Attester consists of both an Attesting Environment and an Attested Environment. In some implementations these environments may be combined. Other implementations may have multiples of Attesting and Attested environments. Although endpoint environments can be complex, and that complexity is security relevant, the basic function

of an Attester is to create Evidence that captures operational conditions affecting trustworthiness.

#### 5.3.1.2. Asserter

The Asserter role is out of scope. The mechanism by which an Asserter communicates Known-Good-Values to a Verifier is also not subject to standardization. Users of the RATS architecture are assumed to have pre-existing mechanisms.

#### 5.3.1.3. Verifier

The Verifier workflow function accepts Evidence from an Attester and accepts Reference Values from one or more Asserters. Reference values may be supplied a priori, cached or used to create policies. The Verifier performs an appraisal by matching Claims found in Evidence with Claims found in Reference Values and policies. If an attested Claim value differs from an expected Claim value, the Verifier flags this as a change possibly impacting trust level.

Endorsements may not have corresponding Claims in Evidence (because of their intrinsic nature). Consequently, the Verifier need only authenticate the endpoint and verify the Endorsements match the endpoint identity.

The result of appraisals and Endorsements, informed by owner policies, produces a new set of Claims that a Relying Party is suited to consume.

#### 5.3.1.4. Relying Party

A Role in an attestation workflow that accepts Attestation Results from a Verifier that may be used by the Relying Party to inform application specific decision making. How Attestation Results are used to inform decision making is out-of-scope for this architecture.

### 5.3.2. Role Messages

#### 5.3.2.1. Evidence

Claims that are formatted and protected by an Attester.

Evidence SHOULD satisfy Verifier expectations for freshness, identity, context, provenance, validity, and veracity.

#### 5.3.2.2. Reference Values

Reference-values are Claims that a manufacturer, vendor or other supply chain entity makes that affects the trustworthiness of an Attester endpoint.

Claims may be persistent properties of the endpoint due to the physical nature of how it was manufactured or may reflect the processes that were followed as part of moving the endpoint through a supply-chain; e.g., validation or compliance testing. This class of Reference-values is known as Endorsements.

Another class of Reference-values identifies the firmware and software that could be installed in the endpoint after its manufacture. A digest of the the firmware or software can be an effective identifier for keeping track of the images produced by vendors and installed on an endpoint. This class of Reference-value is referred to as Known-Good-Value (KGV).

**Known-Good-Values:** Claims about the Attested Environment.

Typically, Known-Good-Value (KGV) Claims are message digests of firmware, software or configuration data supplied by various vendors. If an Attesting Environment implements cryptography, they include Claims about key material.

Like Claims, Known-Good-Values SHOULD satisfy a Verifier's expectations for freshness, identity, context, provenance, validity, relevance and veracity. Known-Good-Values are reference Claims that are - like Evidence - well formatted and protected (e.g. signed).

**Endorsements:** Claims about immutable and implicit characteristics of the Attesting Environment. Typically, endorsement Claims are created by manufacturing or supply chain entities.

Endorsements are intended to increase the level of confidence with respect to Evidence created by an Attester.

#### 5.3.2.3. Attestation Results

Statements about the output of an appraisal of Evidence that are created, formatted and protected by a Verifier.

Attestation Results provide the basis for a Relying Party to establish a level of confidence in the trustworthiness of an Attester. Attestation Results SHOULD satisfy Relying Party expectations for freshness, identity, context, provenance, validity, relevance and veracity.

#### 5.4. Principals (Entities?) - Containers for the Roles

[The authors are unhappy with the term Principal, and have been looking for something else. JOSE/JWT uses the term Principal]

Principals are Containers for the Roles.

Principals are users, organizations, devices and computing environments (e.g., devices, platforms, services, peripherals).

Principals may implement one or more Roles. Message flows occurring within the same Principal are out-of-scope.

The methods whereby Principals may be identified, discovered, authenticated, connected and trusted, though important, are out-of-scope.

Principal operations that apply resiliency, scaling, load balancing or replication are generally believed to be out-of-scope.

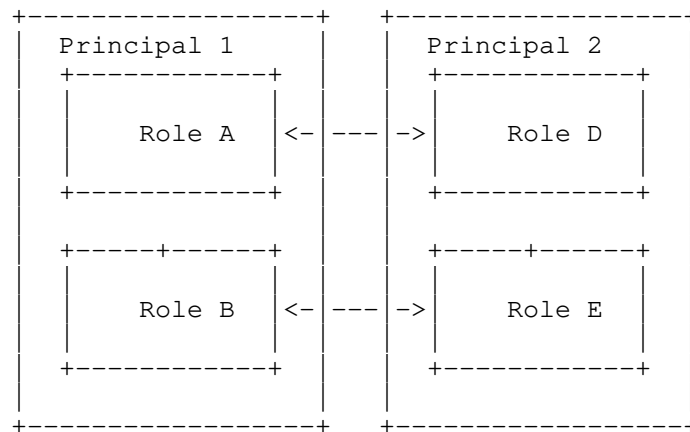


Figure 4: Principals-Role Composition

Principals have the following properties:

- o Multiplicity - Multiple instances of Principals that possess the same Roles can exist.
- o Composition - Principals possessing different Roles can be combined into a singleton Principal possessing the union of Roles. Message flows between combined Principals is uninteresting.

- o Decomposition - A singleton Principal possessing multiple Roles can be divided into multiple Principals.

## 6. Privacy Considerations

The conveyance of Evidence and the resulting Attestation Results reveal a great deal of information about the internal state of a device. In many cases the whole point of the Attestation process is to provide reliable evidence about the type of the device and the firmware that the device is running. This information is particularly interesting to many attackers: knowing that a device is running a weak version of a the firmware provides a way to aim attacks better.

Just knowing the existence of a device is itself a disclosure.

Conveyance protocols must detail what kinds of information is disclosed, and to whom it is exposed.

## 7. Security Considerations

Evidence, Verifiable Assertions and Attestation Results SHOULD use formats that support end-to-end integrity protection and MAY support end-to-end confidentiality protection.

Replay attacks are a concern that protocol implementations MUST deal with. This is typically done via a Nonce Claim, but the details belong to the protocol.

All other attacks involving RATS structures are not explicitly addressed by the architecture.

Additional security protections MAY be required of conveyance mechanisms. For example, additional means of authentication, confidentiality, integrity, replay, denial of service and privacy protection of RATS payloads and Principals may be needed.

## 8. Acknowledgements

Dave Thaler created the concepts of "Passport" and "Background Check".

## 9. References

## 9.1. Normative References

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

## 9.2. Informative References

- [ABLP] Abadi, M., Burrows, M., Lampson, B., and G. Plotkin, "A Calculus for Access Control in Distributed Systems", Springer Annual International Cryptology Conference, page 1-23, DOI 10.1.1.36.691, 1991.
- [DOLEV-YAO] Dolev, D. and A. Yao, "On the security of public key protocols", IEEE Transactions on Information Theory Vol. 29, pp. 198-208, DOI 10.1109/tit.1983.1056650, March 1983.
- [fido] FIDO Alliance, ., "FIDO Specification Overview", 2019, <<https://fidoalliance.org/specifications/>>.
- [I-D.birkholz-rats-tuda] Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-rats-tuda-01 (work in progress), September 2019.
- [I-D.fedorkow-rats-network-device-attestation] Fedorkow, G. and J. Fitzgerald-McKay, "Network Device Attestation Workflow", draft-fedorkow-rats-network-device-attestation-00 (work in progress), July 2019.
- [I-D.ietf-rats-eat] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-01 (work in progress), July 2019.
- [I-D.ietf-teep-architecture] Pei, M., Tschofenig, H., Wheeler, D., Atyeo, A., and D. Liu, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-03 (work in progress), July 2019.



- [I-D.richardson-rats-usecases]  
Richardson, M., Wallace, C., and W. Pan, "Use cases for Remote Attestation common encodings", draft-richardson-rats-usecases-05 (work in progress), October 2019.
- [iothreats]  
GDN, ., "The Internet of Things or the Internet of threats?", 2016, <<https://gcn.com/articles/2016/05/03/internet-of-threats.aspx>>.
- [keystore]  
Google, ., "Android Keystore System", 2019, <<https://developer.android.com/training/articles/keystore>>.
- [Lampson2007]  
Lampson, B., "Practical Principles for Computer Security", IOSPress Proceedings of Software System Reliability and Security, page 151-195, DOI 10.1.1.63.5360, 2007.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [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>>.
- [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/info/rfc5209>>.

#### Authors' Addresses

Henk Birkholz  
Fraunhofer SIT  
Rheinstrasse 75  
Darmstadt 64295  
Germany

Email: [henk.birkholz@sit.fraunhofer.de](mailto:henk.birkholz@sit.fraunhofer.de)

Monty Wiseman  
GE Global Research  
USA

Email: [monty.wiseman@ge.com](mailto:monty.wiseman@ge.com)

Hannes Tschofenig  
ARM Ltd.  
110 Fulbourn Rd  
Cambridge CB1 9NJ  
UK

Email: [hannes.tschofenig@gmx.net](mailto:hannes.tschofenig@gmx.net)

Ned Smith  
Intel Corporation  
USA

Email: [ned.smith@intel.com](mailto:ned.smith@intel.com)

Michael Richardson  
Sandelman Software Works  
Canada

Email: [mcr+iETF@sandelman.ca](mailto:mcr+iETF@sandelman.ca)

RATS Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 9, 2020

H. Birkholz  
M. Eckel  
Fraunhofer SIT  
S. Bhandari  
B. Sulzen  
E. Voit  
Cisco  
L. Xia  
Huawei  
T. Laffey  
HPE  
G. Fedorkow  
Juniper  
July 08, 2019

YANG Module for Basic Challenge-Response-based Remote Attestation  
Procedures  
draft-birkholz-rats-basic-yang-module-01

Abstract

This document defines a YANG RPC and a minimal datastore tree required to retrieve attestation evidence about integrity measurements from a composite device with one or more roots of trust for reporting. Complementary measurement logs are also provided by the YANG RPC originating from one or more roots of trust of measurement. The module defined requires a TPM 2.0 and corresponding Trusted Software Stack included in the device components of the composite device the YANG server is running on.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

## Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	2
1.1. Requirements notation . . . . .	3
2. The YANG Module for Basic Remote Attestation Procedures . . .	3
2.1. Tree format . . . . .	3
2.2. Raw Format . . . . .	7
3. IANA considerations . . . . .	29
4. Security Considerations . . . . .	29
5. Acknowledgements . . . . .	29
6. Change Log . . . . .	29
7. References . . . . .	29
7.1. Normative References . . . . .	29
7.2. Informative References . . . . .	30
Authors' Addresses . . . . .	30

## 1. Introduction

This document is based on the terminology defined in the [I-D.birkholz-attestation-terminology] and uses the interaction model and information elements defined in the [I-D.birkholz-rats-reference-interaction-model] document. The currently supported hardware security module (HWM) - sometimes also referred to as an embedded secure element (eSE) - is the Trusted Platform Module (TPM) 2.0 specified by the Trusted Computing Group (TCG). One or more TPM 2.0 embedded in the components of a composite device - sometimes also referred to as an aggregate device - are required in order to use the YANG module defined in this document. A TPM 2.0 is used as a root of trust for reporting (RTR) in order to retrieve attestation evidence from a composite device. Additionally, it is used as a root of trust for measurement (RTM) in order to provide event logs - sometimes also referred to as measurement logs.

### 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 RFC 2119, BCP 14 [RFC2119].

## 2. The YANG Module for Basic Remote Attestation Procedures

One or more TPM 2.0 MUST be embedded in the composite device that is providing attestation evidence via the YANG module defined in this document. The ietf-basic-remote-attestation YANG module enables a composite device to take on the role of Claimant and Attester in accordance with the Remote Attestation Procedures (RATS) architecture [I-D.birkholz-attestation-terminology] and the corresponding challenge-response interaction model defined in the [I-D.birkholz-rats-reference-interaction-model] document. A fresh nonce with an appropriate amount of entropy 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. The functions of this YANG module are restricted to 0-1 TPM 2.0 per hardware component.

### 2.1. Tree format

<CODE BEGINS>

```
module: ietf-basic-remote-attestation
  +--ro rats-support-structures
    +--ro supported-algos*   uint16
    +--ro tpms* [tpm_name]
      |   +--ro tpm_name          string
      |   +--ro tpm-physical-index?  int32 {ietfhw:entity-mib}?
      |   +--ro certificates* []
      |     +--ro certificate
      |       +--ro certificate-name?   string
      |       +--ro certificate-type?   enumeration
      |       +--ro certificate-value?  ietfct:end-entity-cert-cms
    +--ro compute-nodes* [node-name]
      +--ro node-name          string
      +--ro node-physical-index?  int32 {ietfhw:entity-mib}?

  rpcs:
    +---x tpm12-challenge-response-attestation
      |   +---w input
      |     +---w tpm1-attestation-challenge
      |       +---w pcr-indices*          uint8
      |       +---w nonce-value          binary
      |       +---w TPM_SIG_SCHEME-value  uint8
```

```

+---w (key-identifier)?
+---:(public-key)
| +---w pub-key-id?          binary
+---:(TSS_UUID)
+---w TSS_UUID-value
+---w ulTimeLow?             uint32
+---w usTimeMid?             uint16
+---w usTimeHigh?           uint16
+---w bClockSeqHigh?         uint8
+---w bClockSeqLow?          uint8
+---w rgbNode*               uint8
+---w add-version?           boolean
+---w tpm_name?              string
+---w tpm-physical-index?    int32 {ietfhw:entity-mib}?
+---ro output
+---ro tpm12-attestation-response* [tpm_name]
+---ro tpm_name              string
+---ro tpm-physical-index?    int32 {ietfhw:entity-mib}?
+---ro up-time?              uint32
+---ro node-name?            string
+---ro node-physical-index?   int32 {ietfhw:entity-mib}?
+---ro fixed?                binary
+---ro external-data?         binary
+---ro signature-size?        uint32
+---ro signature?             binary
+---ro (tpm12-quote)
+---:(tpm12-quote1)
+---ro version* []
+---ro major?                uint8
+---ro minor?                uint8
+---ro revMajor?             uint8
+---ro revMinor?             uint8
+---ro digest-value?         binary
+---ro TPM_PCR_COMPOSITE* []
+---ro pcr-indices*          uint8
+---ro value-size?           uint32
+---ro tpm12-pcr-value*      binary
+---:(tpm12-quote2)
+---ro tag?                  uint8
+---ro pcr-indices*          uint8
+---ro locality-at-release?   uint8
+---ro digest-at-release?     binary
+---x tpm20-challenge-response-attestation
+---w input
+---w tpm20-attestation-challenge
+---w pcr-list* []
+---w pcr
+---w pcr-indices*           uint8

```

```

      +---w (algo-registry-type)
      +---: (tcg)
      |   +---w tcg-hash-algo-id?          uint16
      +---: (ietf)
      |   +---w ietf-ni-hash-algo-id?      uint8
      +---w nonce-value                    binary
      +---w (signature-identifier-type)
      |   +---: (TPM_ALG_ID)
      |   |   +---w TPM_ALG_ID-value?      uint16
      +---: (COSE_Algorithm)
      |   +---w COSE_Algorithm-value?      int32
      +---w (key-identifier)?
      |   +---: (public-key)
      |   |   +---w pub-key-id?            binary
      +---: (uuid)
      |   +---w uuid-value?                binary
      +---w tpms* [tpm_name]
      +---w tpm_name                       string
      +---w tpm-physical-index?            int32 {ietfhw:entity-mib}?
+---ro output
+---ro tpm20-attestation-response* [tpm_name]
+---ro tpm_name                           string
+---ro tpm-physical-index?                int32 {ietfhw:entity-mib}?
+---ro up-time?                           uint32
+---ro node-name?                         string
+---ro node-physical-index?               int32 {ietfhw:entity-mib}?
+---ro tpms-attest
|   +---ro pcrdigest?                     binary
|   +---ro tpms-attest-result?            binary
|   +---ro tpms-attest-result-length?     uint32
+---ro tpmt-signature?                    binary
+---x basic-trust-establishment
+---w input
+---w nonce-value                         binary
+---w (signature-identifier-type)
|   +---: (TPM_ALG_ID)
|   |   +---w TPM_ALG_ID-value?          uint16
+---: (COSE_Algorithm)
|   +---w COSE_Algorithm-value?          int32
+---w tpm_name?                           string
+---w tpm-physical-index?                  int32 {ietfhw:entity-mib}?
+---w certificate-name?                    string
+---ro output
+---ro attestation-certificates* [tpm_name]
+---ro tpm_name                           string
+---ro tpm-physical-index?                int32 {ietfhw:entity-mib}?
+---ro up-time?                           uint32
+---ro node-name?                         string

```

```

|         +---ro node-physical-index?          int32 {ietfhw:entity-mib}?
|         +---ro certificate-name?             string
|         +---ro attestation-certificate?      ietfct:end-entity-cert-cms
|         +---ro (key-identifier)?
|             +---:(public-key)
|                 | +---ro pub-key-id?         binary
|             +---:(uuid)
|                 +---ro uuid-value?           binary
+---x log-retrieval
+---w input
|   +---w log-selector* [node-name]
|   |   +---w node-name                       string
|   |   +---w node-physical-index?            int32 {ietfhw:entity-mib}?
|   |   +---w (index-type)?
|   |       +---:(last-entry)
|   |           | +---w last-entry-value?     binary
|   |       +---:(index)
|   |           | +---w index-number?          uint64
|   |       +---:(timestamp)
|   |           +---w timestamp?               yang:date-and-time
|   +---w log-type                           identityref
|   +---w pcr-list* []
|   |   +---w pcr
|   |       +---w pcr-indices*                 uint8
|   |       +---w (algo-registry-type)
|   |           +---:(tcg)
|   |               | +---w tcg-hash-algo-id?  uint16
|   |           +---:(ietf)
|   |               +---w ietf-ni-hash-algo-id? uint8
|   +---w log-entry-quantity?  uint16
+---ro output
+---ro system-event-logs
|   +---ro node-data* [node-name tpm_name]
|   |   +---ro node-name                     string
|   |   +---ro node-physical-index?          int32 {ietfhw:entity-mib}?
|   |   +---ro up-time?                      uint32
|   |   +---ro tpm_name                      string
|   |   +---ro tpm-physical-index?           int32 {ietfhw:entity-mib}?
|   |   +---ro log-result
|   |       +---ro (log-type)
|   |           +---:(bios)
|   |               +---ro bios-event-logs
|   |                   +---ro bios-event-entry* [event-number]
|   |                       +---ro event-number    uint32
|   |                       +---ro event-type?      uint32
|   |                       +---ro pcr-index?       uint16
|   |                       +---ro digest-list* []
|   |                           | +---ro (algo-registry-type)

```



```
| | | +--:(tcg)
| | |   |--ro tcg-hash-algo-id?      uint16
| | |   +--:(ietf)
| | |     |--ro ietf-ni-hash-algo-id?  uint8
| | |     |--ro digest*                binary
|--ro event-size?          uint32
|--ro event-data*         uint8
+--:(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 template-hash-algorithm? string
      |--ro template-hash?          binary
      |--ro pcr-index?              uint16
      |--ro signature?              binary
```

&lt;CODE ENDS&gt;

## 2.2. Raw Format

&lt;CODE BEGINS&gt;

```

module ietf-basic-remote-attestation {
  namespace "urn:ietf:params:xml:ns:yang:ietf-basic-remote-attestation";
  prefix "yang-brat";

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-hardware {
    prefix ietfhw;
  }
  import ietf-crypto-types {
    prefix ietfct;
  }

  organization
    "Fraunhofer SIT";
  contact
    "Henk Birkholz
    Fraunhofer Institute for Secure Information Technology
    Email: henk.birkholz@sit.fraunhofer.de";
  description
    "A YANG module to enable TPM 1.2 and TPM 2.0 based
    remote attestation procedures.
    Copyright (C) Fraunhofer SIT (2019).";
  revision "2019-07-08" {

```

```
description
  "Second version";
reference
  "draft-birkholz-rats-basic-yang-module";
}

grouping hash-algo {
  description
    "A selector for the hashing algorithm";
  choice algo-registry-type {
    mandatory true;
    description
      "Unfortunately, both IETF and TCG have registries here.
      Choose your weapon wisely.";
    case tcg {
      description
        "you chose the east door, the tcg space opens up to
        you.";
      leaf tcg-hash-algo-id {
        type uint16;
        description
          "This is an index referencing the TCG Algorithm
          Registry based on TPM_ALG_ID.";
      }
    }
    case ietf {
      description
        "you chose the west door, the ietf space opens up to
        you.";
      leaf ietf-ni-hash-algo-id {
        type uint8;
        description
          "This is an index referencing the Named Information
          Hash Algorithm Registry.";
      }
    }
  }
}

grouping hash {
  description
    "The hash value including hash-algo identifier";
  list hash-digests {
    description
      "The list of hashes.";
    container hash-digest {
      description
        "A hash value based on a hash algorithm registered by an
```

```
        SDO.";
        uses hash-algo;
        leaf hash-value {
            type binary;
            description
                "The binary representation of the hash value.";
        }
    }
}

grouping nonce {
    description
        "A nonce to show freshness and counter replays.";
    leaf nonce-value {
        type binary;
        mandatory true;
        description
            "This nonce SHOULD be generated via a registered
            cryptographic-strength algorithm. In consequence, the length
            of the nonce depends on the hash algorithm used. The algorithm
            used in this case is independent from the hash algorithm used to
            create the hash-value in the response of the attester.";
    }
}

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.
        Requesting a PCR value that is not in scope of the AC used, detailed
        exposure via error msg should be avoided.";
    leaf-list pcr-indices {
        type uint8;
        description
            "The numbers/indexes of the PCRs. At the moment this is limited
            to 32.";
    }
}

grouping tpm20-pcr-selection {
    description
        "A Verifier can request one or more PCR values uses its
        individually created AC. The corresponding selection filter is
        represented in this grouping. Requesting a PCR value that is not
        in scope of the AC used, detailed exposure via error msg should
        be avoided.";
```

```
list pcr-list {
  description
    "For each PCR in this list an individual list of banks
    (hash-algo) can be requested. It depends on the datastore, if
    every bank in this grouping is included per PCR (crude), or if
    each requested bank set is returned for each PCR individually
    (elegant).";
  container pcr {
    description
      "The composite of a PCR number and corresponding bank
      numbers.";
    leaf-list pcr-indices {
      type uint8;
      description
        "The number of the PCR. At the moment this is limited
        32";
    }
    uses hash-algo;
  }
}

grouping pcr-selector {
  description
    "A Verifier can request the generation of an attestation
    certificate (a signed public attestation key
    (non-migratable, tpm-resident) wrt one or more PCR values.
    The corresponding creation input is represented in this grouping.
    Requesting a PCR value that is not supported results in an error,
    detailed exposure via error msg should be avoided.";
  list pcr-list {
    description
      "For each PCR in this list an individual hash-algo can be
      requested.";
    container pcr {
      description
        "The composite of a PCR number and corresponding bank
        numbers.";
      leaf-list pcr-index {
        type uint8;
        description
          "The numbers of the PCRs that are associated with
          the created key. At the moment the highest number is 32";
      }
      uses hash-algo;
    }
  }
}
```

```
grouping tpm12-signature-scheme {
  description
    "The signature scheme used to sign the evidence via a TPM 1.2.";
  leaf TPM_SIG_SCHEME-value {
    type uint8;
    mandatory true;
    description
      "Selects the signature scheme that is used to sign the TPM quote
      information response. Allowed values can be found in the table at
      the bottom of page 32 in the TPM 1.2 Structures specification
      (Level 2 Revision 116, 1 March 2011).";
  }
}

grouping tpm20-signature-scheme {
  description
    "The signature scheme used to sign the evidence.";
  choice signature-identifier-type {
    mandatory true;
    description
      "There are multiple ways to reference a signature type.
      This used to select the signature algo to sign the quote
      information response.";
    case TPM_ALG_ID {
      description
        "This references the indices of table 9 in the TPM 2.0
        structure specification.";
      leaf TPM_ALG_ID-value {
        type uint16;
        description
          "The TPM Algo ID.";
      }
    }
    case COSE_Algorithm {
      description
        "This references the IANA COSE Algorithms Registry indices.
        Every index of this registry to be used must be mapable to a
        TPM_ALG_ID value.";
      leaf COSE_Algorithm-value {
        type int32;
        description
          "The TPM Algo ID.";
      }
    }
  }
}

grouping tpm12-attestation-key-identifier {
```

```
description
  "A selector for a suitable key identifier for a TPM 1.2.";
choice key-identifier {
  description
    "Identifier for the attestation key to use for signing
    attestation evidence.";
  case public-key {
    leaf pub-key-id {
      type binary;
      description
        "The value of the identifier for the public key.";
    }
  }
  case TSS_UUID {
    description
      "Use a YANG agent generated (and maintained) attestation
      key UUID that complies with the TSS_UUID datatype of the TCG
      Software Stack (TSS) Specification, Version 1.10 Golden,
      August 20, 2003.";
    container TSS_UUID-value {
      description
        "A detailed structure that is used to create the
        TPM 1.2 native TSS_UUID as defined in the TCG Software
        Stack (TSS) Specification, Version 1.10 Golden,
        August 20, 2003.";
      leaf ulTimeLow {
        type uint32;
        description
          "The low field of the timestamp.";
      }
      leaf usTimeMid {
        type uint16;
        description
          "The middle field of the timestamp.";
      }
      leaf usTimeHigh {
        type uint16;
        description
          "The high field of the timestamp multiplexed with the
          version number.";
      }
      leaf bClockSeqHigh {
        type uint8;
        description
          "The high field of the clock sequence multiplexed with
          the variant.";
      }
      leaf bClockSeqLow {
```

```
        type uint8;
        description
            "The low field of the clock sequence.";
    }
    leaf-list rgbNode {
        type uint8;
        description
            "The spatially unique node identifier.";
    }
}
}
}

grouping tpm20-attestation-key-identifier {
    description
        "A selector for a suitable key identifier.";
    choice key-identifier {
        description
            "Identifier for the attestation key to use for signing
            attestation evidence.";
        case public-key {
            leaf pub-key-id {
                type binary;
                description
                    "The value of the identifier for the public key.";
            }
        }
        case uuid {
            description
                "Use a YANG agent generated (and maintained) attestation
                key UUID.";
            leaf uuid-value {
                type binary;
                description
                    "The UUID identifying the corresponding public key.";
            }
        }
    }
}

grouping tpm-name {
    description
        "In a system with multiple-TPMs get the data from a specific TPM
        identified by the name and physical-index.";
    leaf tpm_name {
        type string;
        description
```

```
        "Name of the TPM or All";
    }
    leaf tpm-physical-index {
        if-feature ietfhw:entity-mib;
        type int32 {
            range "1..2147483647";
        }
        config false;
        description
            "The entPhysicalIndex for the TPM.";
        reference
            "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
    }
}

grouping compute-node {
    description
        "In a distributed system with multiple compute nodes
        this is the node identified by name and physical-index.";
    leaf node-name {
        type string;
        description
            "Name of the compute node or All";
    }
    leaf node-physical-index {
        if-feature ietfhw:entity-mib;
        type int32 {
            range "1..2147483647";
        }
        config false;
        description
            "The entPhysicalIndex for the compute node.";
        reference
            "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
    }
}

grouping tpml2-pcr-info-short {
    description
        "This structure is for defining a digest at release when the only
        information that is necessary is the release configuration.";
    uses tpml2-pcr-selection;
    leaf locality-at-release {
        type uint8;
        description
            ".This SHALL be the locality modifier required to release the
            information (TPM 1.2 type TPM_LOCALITY_SELECTION)";
    }
    leaf digest-at-release {
```



```
    type binary;
    description
        "This SHALL be the digest of the PCR indices and PCR values
        to verify when revealing auth data (TPM 1.2 type
        TPM_COMPOSITE_HASH).";
}
}

grouping tpml2-version {
    description
        "This structure provides information relative the version of
        the TPM.";
    list version {
        description
            "This indicates the version of the structure
            (TPM 1.2 type TPM_STRUCT_VER). This MUST be 1.1.0.0.";
        leaf major {
            type uint8;
            description
                "Indicates the major version of the structure.
                MUST be 0x01.";
        }
        leaf minor {
            type uint8;
            description
                "Indicates the minor version of the structure.
                MUST be 0x01.";
        }
        leaf revMajor {
            type uint8;
            description
                "Indicates the rev major version of the structure.
                MUST be 0x00.";
        }
        leaf revMinor {
            type uint8;
            description
                "Indicates the rev minor version of the structure.
                MUST be 0x00.";
        }
    }
}

grouping tpml2-quote-info-common {
    description
        "These statements are used in bot quote variants of the TPM 1.2";
    leaf fixed {
        type binary;
    }
}
```

```
        description
            "This SHALL always be the string 'QUOT' or 'QUO2'
            (length is 4 bytes).";
    }
    leaf external-data {
        type binary;
        description
            "160 bits of externally supplied data, typically a nonce.";
    }
    leaf signature-size {
        type uint32;
        description
            "The size of TPM 1.2 'signature' value.";
    }
    leaf signature {
        type binary;
        description
            "Signature over SHA-1 hash of tpml2-quote-info2'.";
    }
}

grouping tpml2-quote-info {
    description
        "This structure provides the mechanism for the TPM to quote the
        current values of a list of PCRs (as used by the TPM_Quote2
        command).";
    uses tpml2-version;
    leaf digest-value {
        type binary;
        description
            "This SHALL be the result of the composite hash algorithm using
            the current values of the requested PCR indices
            (TPM 1.2 type TPM_COMPOSITE_HASH.)";
    }
}

grouping tpml2-quote-info2 {
    description
        "This structure provides the mechanism for the TPM to quote the
        current values of a list of PCRs
        (as used by the TPM_Quote2 command).";
    leaf tag {
        type uint8;
        description
            "This SHALL be TPM_TAG_QUOTE_INFO2.";
    }
    uses tpml2-pcr-info-short;
}
```

```
grouping tpm12-cap-version-info {
  description
    "TPM returns the current version and revision of the TPM 1.2 .";
  list TPM_PCR_COMPOSITE {
    description
      "The TPM 1.2 TPM_PCRVALUES for the pcr-indices.";
    uses tpm12-pcr-selection;
    leaf value-size {
      type uint32;
      description
        "This SHALL be the size of the 'tpm12-pcr-value' field
        (not the number of PCRs).";
    }
    leaf-list tpm12-pcr-value {
      type binary;
      description
        "The list of TPM_PCRVALUES from each PCR selected in sequence
        of tpm12-pcr-selection.";
    }
  }
  list version-info {
    description
      "An optional output parameter from a TPM 1.2 TPM_Quote2.";
    leaf tag {
      type uint16;
      description
        "The TPM 1.2 version and revision
        (TPM 1.2 type TPM_STRUCTURE_TAG).
        This MUST be TPM_CAP_VERSION_INFO (0x0030)";
    }
    uses tpm12-version;
    leaf spec-level {
      type uint16;
      description
        "A number indicating the level of ordinals supported.";
    }
    leaf errata-rev {
      type uint8;
      description
        "A number indicating the errata version of the
        specification.";
    }
    leaf tpm-vendor-id {
      type binary;
      description
        "The vendor ID unique to each TPM manufacturer.";
    }
    leaf vendor-specific-size {
      type uint16;
    }
  }
}
```

```
        description
            "The size of the vendor-specific area.";
    }
    leaf vendor-specific {
        type binary;
        description
            "Vendor specific information.";
    }
}

}

}

grouping tpm12-pcr-composite {
    description
        "The actual values of the selected PCRs (a list of TPM_PCRVALUES
        (binary) and associated metadata for TPM 1.2.";
    list TPM_PCR_COMPOSITE {
        description
            "The TPM 1.2 TPM_PCRVALUES for the pcr-indices.";
        uses tpm12-pcr-selection;
        leaf value-size {
            type uint32;
            description
                "This SHALL be the size of the 'tpm12-pcr-value' field
                (not the number of PCRs).";
        }
        leaf-list tpm12-pcr-value {
            type binary;
            description
                "The list of TPM_PCRVALUES from each PCR selected in sequence
                of tpm12-pcr-selection.";
        }
    }
}

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";
    }
}

identity log-type {
    description
        "The type of logs available.";
```

```
}

identity bios {
  base log-type;
  description
    "Measurement log created by the BIOS/UEFI.";
}

identity ima {
  base log-type;
  description
    "Measurement log created by IMA.";
}

grouping log-identifier {
  description
    "Identifier for type of log to be retrieved.";
  leaf log-type {
    type identityref {
      base log-type;
    }
    mandatory true;
    description
      "The corresponding measurement log type identity.";
  }
}

grouping boot-event-log {
  description
    "Defines an event log corresponding to the event that extended the
    PCR";
  leaf event-number {
    type uint32;
    description
      "Unique event number of this event";
  }
  leaf event-type {
    type uint32;
    description
      "log event type";
  }
  leaf pcr-index {
    type uint16;
    description
      "Defines the PCR index that this event extended";
  }
  list digest-list {
    description "Hash of event data";
  }
}
```

```
    uses hash-algo;
    leaf-list digest {
        type binary;
        description
            "The hash of the event data";
    }
}
leaf event-size {
    type uint32;
    description
        "Size of the event data";
}
leaf-list event-data {
    type uint8;
    description
        "the event data size determined by event-size";
}
}

grouping ima-event {
    description
        "Defines an hash log extend event for IMA measurements";
    leaf event-number {
        type uint64;
        description
            "Unique number for this event for sequencing";
    }
    leaf ima-template {
        type string;
        description
            "Name of the template used for event logs
            for e.g. ima, ima-ng";
    }
    leaf filename-hint {
        type string;
        description
            "File that was measured";
    }
    leaf filedata-hash {
        type binary;
        description
            "Hash of filedata";
    }
    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 uint16;
    description
        "Defines the PCR index that this event extended";
}
leaf signature {
    type binary;
    description
        "The file signature";
}
}

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-log {
    list ima-event-entry {
        key event-number;
        description
            "Ordered list of ima event logs by event-number";
        uses ima-event;
    }
    description
        "Measurement log created by IMA.";
}

grouping event-logs {
    description
        "A selector for the log and its type.";
    choice log-type {
        mandatory true;
        description
            "Event log type determines the event logs content.";
    }
}
```

```
    case bios {
      description
        "BIOS/UEFI event logs";
      container bios-event-logs {
        description
          "This is an index referencing the TCG Algorithm
          Registry based on TPM_ALG_ID.";
        uses bios-event-log;
      }
    }
  case ima {
    description
      "IMA event logs";
    container ima-event-logs {
      description
        "This is an index referencing the TCG Algorithm
        Registry based on TPM_ALG_ID.";
      uses ima-event-log;
    }
  }
}

rpc tpm12-challenge-response-attestation {
  description
    "This RPC accepts the input for TSS TPM 1.2 commands of the
    managed device. ComponentIndex from the hardware manager YANG
    module to refer to dedicated TPM in composite devices,
    e.g. smart NICs, is still a TODO.";
  input {
    container tpm1-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;
      uses tpm12-signature-scheme;
      uses tpm12-attestation-key-identifier;
      leaf add-version {
        type boolean;
        description
          "Whether or not to include TPM_CAP_VERSION_INFO; if true,
          then TPM_Quote2 must be used to create the response.";
      }
      uses tpm-name;
    }
  }
}
```



```

    }
    output {
      list tpm12-attestation-response {
        key tpm_name;
        description
          "The binary output of TPM 1.2 TPM_Quote/TPM_Quote2, including
            the PCR selection and other associated attestation evidence
            metadata";
        uses tpm-name;
        uses node-uptime;
        uses compute-node;
        uses tpm12-quote-info-common;
        choice tpm12-quote {
          mandatory true;
          description
            "Either a tpm12-quote-info or tpm12-quote-info2, depending
              on whether TPM_Quote or TPM_Quote2 was used
              (cf. input field add-version).";
          case tpm12-quotel {
            description
              "BIOS/UEFI event logs";
            uses tpm12-quote-info;
            uses tpm12-pcr-composite;
          }
          case tpm12-quote2 {
            description
              "BIOS/UEFI event logs";
            uses tpm12-quote-info2;
          }
        }
      }
    }
  }
}

rpc tpm20-challenge-response-attestation {
  description
    "This RPC accepts the input for TSS TPM 2.0 commands of the
      managed device. ComponentIndex from the hardware manager YANG
      module to refer to dedicated TPM in composite devices,
      e.g. smart NICs, is still a TODO.";
  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 tpm20-pcr-selection;
    }
  }
}

```

```
    uses nonce;
    uses tpm20-signature-scheme;
    uses tpm20-attestation-key-identifier;
  }
  list tpms {
    key tpm_name;
    description
      "TPMs to fetch the attestation information.";
    uses tpm-name;
  }
}
output {
  list tpm20-attestation-response {
    key tpm_name;
    description
      "The binary output of TPM2b_Quote. An TPMS_ATTEST structure
      including a length, encapsulated in a signature";
    uses tpm-name;
    uses node-uptime;
    uses compute-node;
    container tpms-attest {
      leaf pcrdigest {
        type binary;
        description
          "split out value of TPMS_QUOTE_INFO for convenience";
      }
      leaf tpms-attest-result {
        type binary;
        description
          "The complete TPM generate structure including
          signature.";
      }
      leaf tpms-attest-result-length {
        type uint32;
        description
          "Length of attest result provided by the TPM structure.";
      }
    }
    description
      "A composite of value and length and list of selected
      pcrcs (original name: [type]attested)";
  }
  leaf tpmt-signature {
    type binary;
    description
      "Split out value of the signature for convenience.
      TODO: check for length values that complement binary value
      data node leafs.";
  }
}
```

```
    }
  }
}

rpc basic-trust-establishment {
  description
    "This RPC creates a tpm-resident, non-migratable key to be used
    in TPM_Quote commands, an attestation certificate.";
  input {
    uses nonce;
    uses tpm20-signature-scheme;
    uses tpm-name;
    leaf certificate-name {
      type string;
      description
        "An arbitrary name for the identity certificate chain
        requested.";
    }
  }
  output {
    list attestation-certificates {
      key tpm_name;
      description
        "Attestation Certificate data from a TPM identified by the TPM
        name";
      uses tpm-name;
      uses node-uptime;
      uses compute-node;
      leaf certificate-name {
        type string;
        description
          "An arbitrary name for this identity certificate or
          certificate chain.";
      }
      leaf attestation-certificate {
        type ietfct:end-entity-cert-cms;
        description
          "The binary signed certificate chain data for this identity
          certificate.";
      }
      uses tpm20-attestation-key-identifier;
    }
  }
}

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 {
  list log-selector {
    key node-name;
    description
      "Selection of log entries to be reported.";
    uses compute-node;
    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 an log event which matches 1:1 with a
            unique event record contained within the log. Log
            entries subsequent to 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 index-number {
          type uint64;
          description
            "The numeric index number of a log entry. Zero means
            to start at the beginning of the log. Entries
            subsequent to 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 subsequent to this timestamp is to be sent.";
    }
    description
      "Timestamp from which to start the extraction.";
  }
}
```

```
    uses log-identifier;
    uses tpm20-pcr-selection;
    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 {
            key "node-name tpm_name";
            description
                "Event logs of a node in a distributed system
                 identified by the node name";
            uses compute-node;
            uses node-uptime;
            uses tpm-name;
            container log-result {
                description
                    "The requested entries of the corresponding log.";
                uses event-logs;
            }
        }
    }
}

container rats-support-structures {
    config false;
    description
        "The datastore definition enabling verifiers or relying
         parties to discover the information necessary to use the
         remote attestation RPCs appropriately.";
    leaf-list supported-algos {
        type uint16;
        description
            "Supported TPM_ALG_ID values for the TPM in question.
             Will include ComponentIndex soon.";
    }
    list tpms {
        key tpm_name;
        uses tpm-name;
        description
            "A list of TPMs in this composite
```

```
device that rats can be conducted with.";
list certificates {
  description
    "The TPM's endorsement-certificate.";
  container certificate {
    leaf certificate-name {
      type string;
      description
        "An arbitrary name for this identity certificate or
        certificate chain.";
    }
    leaf certificate-type {
      type enumeration {
        enum endorsement-cert {
          value 0;
          description
            "EK Cert type.";
        }
        enum attestation-cert {
          value 1;
          description
            "AK Cert type.";
        }
      }
      description "Type of this certificate";
    }
    leaf certificate-value {
      type ietfct:end-entity-cert-cms;
      description
        "The binary signed public endorsement key (EK),
        attestation key (AK) and corresponding assertions
        (EK, AK Certificate). In a TPM 2.0 the EK, AK Certificate
        resides in a well-defined NVRAM location by the TPM
        vendor.";
    }
    description
      "Two kinds of certificates can be accessed via this
      statement. An Attestation Key Certificate and a
      Endorsement Key Certificate.";
  }
}
list compute-nodes {
  key node-name;
  uses compute-node;
  description
    "A list names of hardware components in this composite
    device that rats can be conducted with.";
```

```
    }  
  }  
}  
<CODE ENDS>
```

### 3. IANA considerations

This document will include requests to IANA:

To be defined yet.

### 4. Security Considerations

There are always some.

### 5. Acknowledgements

Not yet.

### 6. Change Log

Changes from version 00 to version 01:

- o Addressed author's comments
- o Extended complementary details about attestation-certificates
- o Relabeled chunk-size to log-entry-quantity
- o Relabeled location with compute-node or tpm-name where appropriate
- o Added a valid entity-mib physical-index to compute-node and tpm-name to map it back to hardware inventory
- o Relabeled name to tpm\_name
- o Removed event-string in last-entry

### 7. References

#### 7.1. Normative References

[I-D.birkholz-rats-reference-interaction-model]  
Birkholz, H. and M. Eckel, "Reference Interaction Model for Challenge-Response-based Remote Attestation", draft-birkholz-rats-reference-interaction-model-00 (work in progress), March 2019.

- [I-D.ietf-netconf-crypto-types]  
Watsen, K. and H. Wang, "Common YANG Data Types for  
Cryptography", draft-ietf-netconf-crypto-types-10 (work in  
progress), July 2019.
- [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>>.

## 7.2. Informative References

- [I-D.birkholz-attestation-terminology]  
Birkholz, H., Wiseman, M., and H. Tschofenig, "Reference  
Terminology for Remote Attestation Procedures", draft-  
birkholz-attestation-terminology-02 (work in progress),  
July 2018.

### Authors' Addresses

Henk Birkholz  
Fraunhofer SIT  
Rheinstrasse 75  
Darmstadt 64295  
Germany

Email: [henk.birkholz@sit.fraunhofer.de](mailto:henk.birkholz@sit.fraunhofer.de)

Michael Eckel  
Fraunhofer SIT  
Rheinstrasse 75  
Darmstadt 64295  
Germany

Email: [michael.eckel@sit.fraunhofer.de](mailto:michael.eckel@sit.fraunhofer.de)

Shwetha Bhandari  
Cisco Systems

Email: [shwethab@cisco.com](mailto:shwethab@cisco.com)

Bill Sulzen  
Cisco Systems

Email: [bsulzen@cisco.com](mailto:bsulzen@cisco.com)



Eric Voit  
Cisco Systems  
Email: [evoit@cisco.com](mailto:evoit@cisco.com)

Liang Xia (Frank)  
Huawei Technologies  
101 Software Avenue, Yuhuatai District  
Nanjing, Jiangsu 210012  
China  
Email: [Frank.Xialiang@huawei.com](mailto:Frank.Xialiang@huawei.com)

Tom Laffey  
Hewlett Packard Enterprise  
Email: [tom.laffey@hpe.com](mailto:tom.laffey@hpe.com)

Guy C. Fedorkow  
Juniper Networks  
10 Technology Park Drive  
Westford, Massachusetts 01886  
Email: [gfedorkow@juniper.net](mailto:gfedorkow@juniper.net)

RATS Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 9, 2021

H. Birkholz  
M. Eckel  
Fraunhofer SIT  
C. Newton  
L. Chen  
University of Surrey  
July 08, 2020

Reference Interaction Models for Remote Attestation Procedures  
draft-birkholz-rats-reference-interaction-model-03

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. Privacy preserving conveyance of Evidence via Direct Anonymous Attestation is elaborated on for each interaction model, individually.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	3
3. Disambiguation . . . . .	4
4. Scope and Intent . . . . .	4
5. Direct Anonymous Attestation . . . . .	5
5.1. Endorsers . . . . .	5
5.2. Endorsers for Direct Anonymous Attestation . . . . .	6
6. Normative Prerequisites . . . . .	6
7. Generic Information Elements . . . . .	7
8. Interaction Models . . . . .	9
8.1. Challenge/Response Remote Attestation . . . . .	10
8.2. Uni-Directional Remote Attestation . . . . .	11
8.3. Streaming Remote Attestation . . . . .	13
9. Additional Application-Specific Requirements . . . . .	15
9.1. Confidentiality . . . . .	15
9.2. Mutual Authentication . . . . .	15
9.3. Hardware-Enforcement/Support . . . . .	15
10. Implementation Status . . . . .	15
10.1. Implementer . . . . .	16
10.2. Implementation Name . . . . .	16
10.3. Implementation URL . . . . .	16
10.4. Maturity . . . . .	16
10.5. Coverage and Version Compatibility . . . . .	16
10.6. License . . . . .	16
10.7. Implementation Dependencies . . . . .	16
10.8. Contact . . . . .	17
11. Security and Privacy Considerations . . . . .	17
12. Acknowledgments . . . . .	17
13. Change Log . . . . .	17
14. References . . . . .	19
14.1. Normative References . . . . .	19
14.2. Informative References . . . . .	20
Appendix A. CDDL Specification for a simple CoAP Challenge/Response Interaction . . . . .	20
Authors' Addresses . . . . .	21

## 1. Introduction

Remote Attestation procedures (RATS, [I-D.ietf-rats-architecture]) 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 - created by an Attester. The roles `_Attester_` and `_Verifier_`, as well as the Conceptual Messages `_Evidence_` and `_Attestation Results_` are terms defined by the RATS Architecture [I-D.ietf-rats-architecture]. This documents captures interaction models that can be used in specific RATS-related solution documents. The primary focus of this document is the conveyance of attestation Evidence. Specific goals of this document are to:

- o prevent inconsistencies in descriptions of these interaction models in other documents (due to text cloning over time),
- o 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, and to
- o illustrate the application of Direct Anonymous Attestation (DAA) for each of the interaction models described.

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 Endorsements or Attestation Results.

## 2. Terminology

This document uses the terms, roles, and concepts defined in [I-D.ietf-rats-architecture]:

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.

### 3. 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 a decoupled system or entities that exchange the payload of the Conceptual Message type called Evidence [I-D.ietf-rats-architecture]. This conveyance can also be "local", if the Verifier is part of the same entity as the Attester, e.g., separate system components of a Composite Device (a single RATS Entity). Examples of these types of co-located 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)).

### 4. 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 [I-D.ietf-rats-architecture] 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 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.

## 5. Direct Anonymous Attestation

DAA [DAA] is a signature scheme used in RATS that allows preservation of the privacy of users that are associated with an Attester (e.g. its owner). 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 a credential for an Attester and makes the public key (in the form of a public key certificate) available to the verifier to enable them to validate the DAA signature obtained as part of the Evidence.

In order to support these DAA signatures, the DAA Issuer MUST associate a single key pair with each 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 public key certificate for the group replaces the Attester Identity documents in the verification of the Evidence (instead of unique Attester Identity documents). This is in contrast to intuition that there has to be a unique Attester Identity per device.

This document extends the duties of the Endorser role as defined by the RATS architecture with respect to the provision of these Attester Identity documents to Attesters. The existing duties of the Endorser role and the duties of a DAA Issuer are quite similar as illustrated in the following subsections.

### 5.1. Endorsers

Via its Attesting Environments, an Attester can only create Evidence about its Target Environments. After being appraised to be trustworthy, a Target Environment may become a new Attesting Environment in charge of creating Evidence for further Target Environments. [I-D.ietf-rats-architecture] explains this as Layered Attestation. Layered Attestation has to start with an initial Attesting Environment (i.e., there cannot be turtles all the way down [turtles]). At this rock bottom of Layered Attestation, the Attesting Environments are called Roots of Trust (RoT). An Attester cannot create 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).

## 5.2. Endorsers for Direct Anonymous Attestation

In order to enable DAA to be used, an Endorser role takes on the duties of a DAA Issuer in addition to its already defined duties. DAA Issuers offer zero-knowledge proofs based on public key certificates used for a group of Attesters [DAA]. Effectively, these certificates share the semantics of Endorsements. The differences are:

- o The associated private keys are used by the DAA Issuer to provide an Attester with a credential that it can use to convince the Verifier that its Evidence is valid. To keep their anonymity the Attester randomises this credential each time that it is used.
- o The Verifier can use the DAA Issuer's public key certificate, together with the randomised credential from the Attester, to confirm that the Evidence comes from a valid Attester.
- o A credential is conveyed from an Endorser to an Attester together with the transfer of the public key certificates from Endorser to Verifier.

The zero-knowledge proofs required cannot be created by an Attester alone - like the Endorsements of RoTs - and have to be created by a trustable third entity - like an Endorser. Due to that vast semantic overlap (XXX-mcr:explain), an Endorser in this document can convey trustable third party statements both to a Verifier and an Attester.

## 6. Normative Prerequisites

In order to ensure an appropriate conveyance of Evidence, the following set of prerequisites MUST be in place to support the implementation of interaction models:

**Attester Identity:** The provenance of Evidence with respect to a distinguishable Attesting Environment MUST be correct and unambiguous.

An Attester Identity MAY be a unique identity, it MAY be included in a zero-knowledge proof (ZKP), or it MAY be part of a group signature, or it MAY be a randomised DAA credential.

**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 (e.g. an PKIX certificate or trusted key material, or a randomised

DAA credential), or SHOULD include a correct and unambiguous and stable reference to an accessible identity document.

**Authentication Secret:** An Authentication Secret MUST be available exclusively to an Attester's Attesting Environment.

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.

**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.

## 7. 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 or can be included in their 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.

**Attester Identity ('attesterIdentity'):** \_mandatory\_

A statement about a distinguishable Attester made by an Endorser without accompanying evidence about its validity - used as proof of identity.

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

**Authentication Secret IDs ('authSecID'):** \_mandatory\_

A statement representing an identifier list that MUST be associated with corresponding Authentication Secrets used to protect Evidence. In DAA, Authentication Secret IDs are



represented by the Endorser (DAA issuer)'s public key that MUST be used to create DAA credentials for the corresponding Authentication Secrets used to protect Evidence.

Each Authentication Secret is uniquely associated with a distinguishable Attesting Environment. Consequently, an Authentication Secret ID also identifies an Attesting Environment. In DAA an Authentication Secret ID does not identify a unique Attesting Environment but associated with a group of Attesting Environments. This is because an Attesting Environment should not be distinguishable and the DAA credential which represents the Attesting Environment is randomised each time it used.

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 Conceptual Message and are, for example, used to appraise the integrity of Attesters via a Verifiers. The other information elements in this section can be expressed as Claims in any type of Conceptual Messages.

Reference Claims ('refClaims') \_mandatory\_

Reference Claims are a specific subset of Appraisal Policies as defined in [I-D.ietf-rats-architecture].

Reference Claims are used to appraise the Claims received from an Attester via appraisal by direct comparison. For example, Reference Claims MAY be Reference Integrity Measurements (RIM) or assertions that are implicitly trusted because they are signed by a trusted authority (see Endorsements in [I-D.ietf-rats-architecture]). Reference Claims typically represent (trusted) Claim sets about an Attester's intended platform operational state.

Claim Selection ('claimSelection'): \_optional\_

A statement that represents a (sub-)set of Claims that can be created by an Attester.

Claim Selections can act as filters that can specify the exact set of Claims to be included in Evidence. An Attester MAY decide whether or not to provide all Claims as requested via a Claim Selection.

Evidence ('signedAttestationEvidence'): \_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. The Evidence MUST be protected via the 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.

## 8. Interaction Models

The following subsections introduce and illustrate the interaction models:

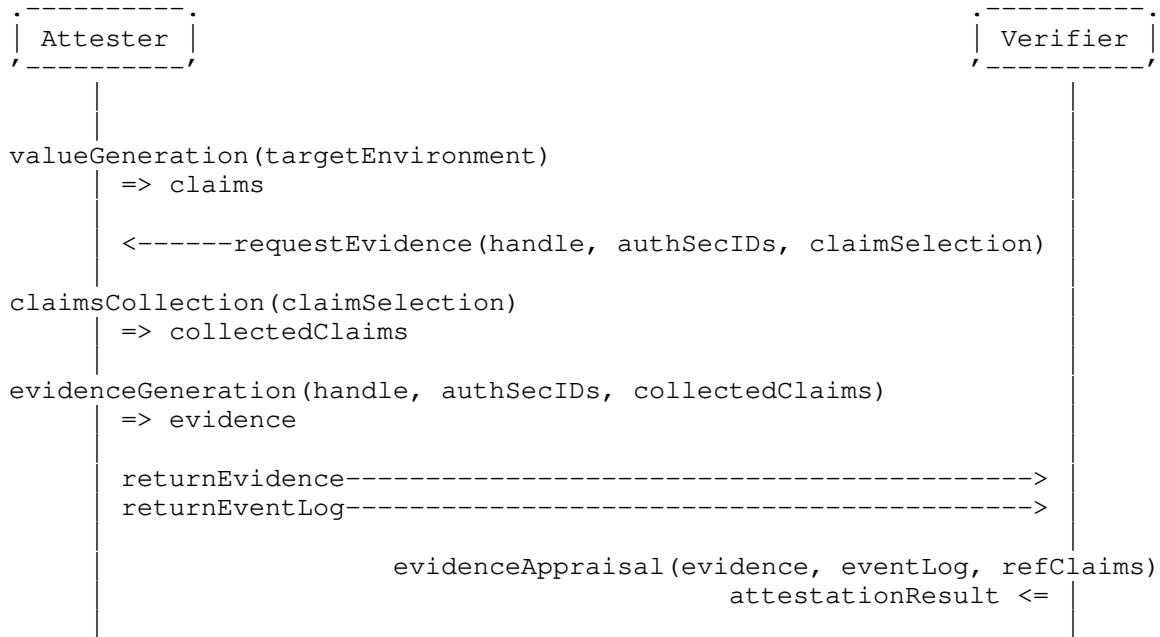
1. Challenge/Response Remote Attestation
2. Uni-Directional Remote Attestation
3. Streaming Remote Attestation

Each section starts with a sequence diagram illustrating the interactions between Attester and Verifier. The other roles RATS roles - mainly Relying Parties and Endorsers - are not relevant for this interaction model. While the interaction models presented focus on the conveyance of Evidence, future work could apply this to the conveyance of other Conceptual Messages, namely Attestation Results, Endorsements, or Appraisal Policies.

All interaction model have a strong focus on the use of a handle to incorporate a proof of freshness. The ways these handles are

processed is the most prominent difference between the three interaction models.

### 8.1. Challenge/Response Remote Attestation



This 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 cryptographically strongly randomly generated, and therefore unpredictable, nonce. The Verifier-generated nonce is intended to guarantee Evidence freshness.

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.

Analogously, a particular set of Evidence originating from a particular Attesting Environments 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.

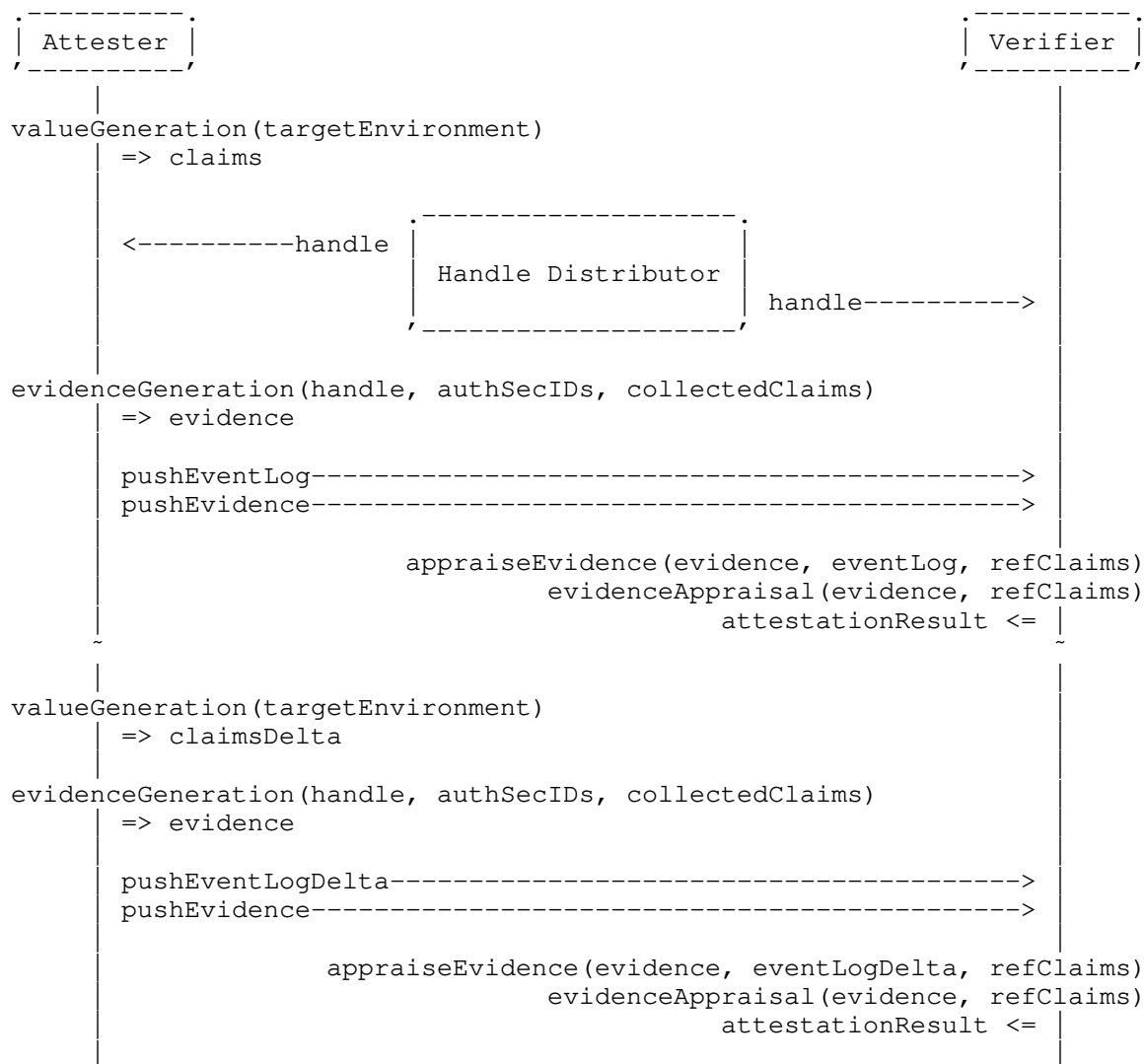
The Claim Selection narrows down the set of Claims collected and used to create Evidence to those that the Verifier requires. If the Claim Selection is omitted, then by default all Claims that are known and available on the Attester MUST be used to create corresponding Evidence. For example when performing a boot integrity evaluation, a Verifier may only be requesting a particular subset of claims about the Attester, such as Evidence about BIOS and firmware the Attester booted up, and not include information about all currently running software.

While it is crucial that Claims, the Handle, as well as the Attester Identity information MUST be cryptographically bound to the signature of Evidence, they may be presented in an encrypted form.

Cryptographic blinding MAY be used at this point. For further reference see section Section 11.

As soon as the Verifier receives signed Evidence, it validates the signature, the Attester Identity, as well as the Nonce, and appraises the Claims. Appraisal procedures are application-specific and can be conducted via comparison of the Claims with corresponding Reference Claims, such as Reference Integrity Measurements. The final output of the Verifier are Attestation Results. Attestation Results constitute new Claims Sets about an Attester's properties and characteristics that enables Relying Parties, for example, to assess an Attester's trustworthiness.

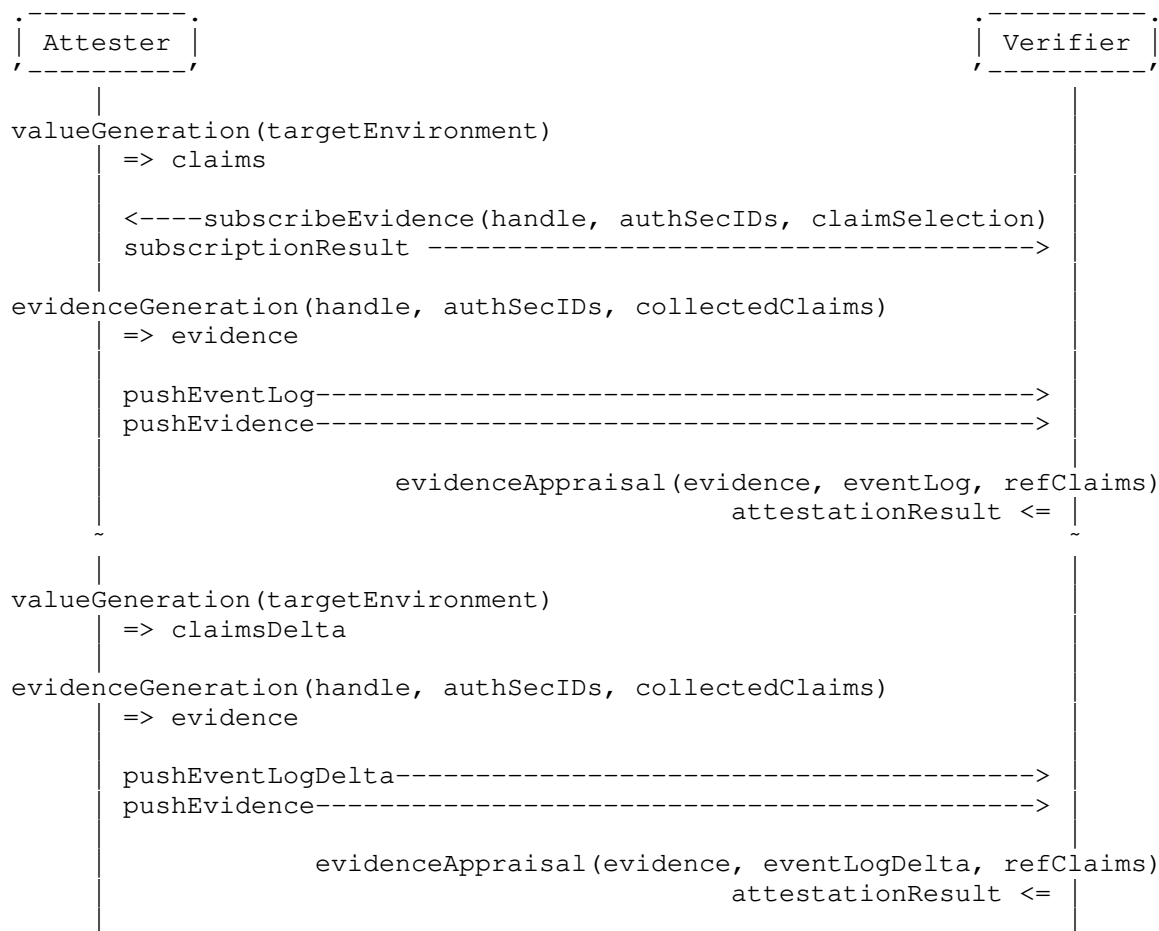
## 8.2. Uni-Directional Remote Attestation



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 trustable signed 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 - that 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 every evidence created. Correspondingly, evidence created for conveyance via this model provides a proof that it was fresh at a certain point in time. Effectively, this allows for 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.

### 8.3. Streaming Remote Attestation



Streaming Remote Attestation procedures require the setup of subscription state. Setting up subscription state between a Verifier and an Attester is conducted via a subscribe operation. This subscribe operation is used to convey the handles required for Evidence generation. Effectively, this allows for series of evidence to be pushed to a Verifier similar to the Uni-Directional model. While a Handle Distributor is not required in this model, it is also limited to bi-lateral subscription relationships, in which each Verifier has to create and provide its individual handle. Handles provided by a specific subscribing Verifier MUST be used in Evidence generation for that specific Verifier. The Streaming model uses the same information elements as the Challenge/Response and the Uni-Directional model. Methods to detect excessive time drift that would mandate a refreshed Handle to be conveyed via another subscribe operation are out-of-scope of this document.

## 9. Additional Application-Specific Requirements

Depending on the use cases covered, there can be additional requirements. An exemplary subset is illustrated in this section.

### 9.1. 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 that preserves confidentiality. In private networks, such as carrier management networks, it must be evaluated whether or not the transport medium is considered confidential.

### 9.2. 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.

### 9.3. 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).

## 10. 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".

#### 10.1. Implementer

The open-source implementation was initiated and is maintained by the Fraunhofer Institute for Secure Information Technology - SIT.

#### 10.2. Implementation Name

The open-source implementation is named "CHallenge-Response based Remote Attestation" or in short: CHARRA.

#### 10.3. Implementation URL

The open-source implementation project resource can be located via:  
<https://github.com/Fraunhofer-SIT/charra>

#### 10.4. Maturity

The code's level of maturity is considered to be "prototype".

#### 10.5. Coverage and Version Compatibility

The current version (commit '847bcde') is aligned with the exemplary specification of the CoAP FETCH bodies defined in section Appendix A of this document.

#### 10.6. License

The CHARRA project and all corresponding code and data maintained on github are provided under the BSD 3-Clause "New" or "Revised" license.

#### 10.7. 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 code and data is also maintained on github and the project resources can be located via:  
<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/>).

#### 10.8. Contact

Michael Eckel ([michael.eckel@sit.fraunhofer.de](mailto:michael.eckel@sit.fraunhofer.de))

#### 11. 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.

#### 12. Acknowledgments

Olaf Bergmann, Michael Richardson, and Ned Smith

#### 13. Change Log

- o Initial draft -00
- o Changes from version 00 to version 01:
  - \* Added details to the flow diagram
  - \* Integrated comments from Ned Smith (Intel)
  - \* Reorganized sections and
  - \* Updated interaction model
  - \* Replaced "claims" with "assertions"
  - \* Added proof-of-concept CDDL for CBOR via CoAP based on a TPM 2.0 quote operation
- o Changes from version 01 to version 02:

- \* Revised the relabeling of "claims" with "assertion" in alignment with the RATS Architecture I-D.
- \* Added Implementation Status section
- \* Updated interaction model
- \* Text revisions based on changes in [I-D.ietf-rats-architecture] and comments provided on rats@ietf.org.
- o Changes from version 02 to version 00 RATS related document
  - \* update of the challenge/response diagram
  - \* minor rephrasing of Prerequisites section
  - \* rephrasing to information elements and interaction model section
- o Changes from version 00 to version 01
  - \* added Attestation Authenticity, updated Identity and Secret
  - \* relabeled Secret ID to Authentication Secret ID + rephrasing
  - \* relabeled Claim Selection to Assertion Selection + rephrasing
  - \* relabeled Evidence to (Signed) Attestation Evidence
  - \* Added Attestation Result and Reference Assertions
  - \* update of the challenge/response diagram and expositional text
  - \* added CDDL spec for CoAP FETCH operation proof-of-concept
- o Changes from version 01 to version 02
  - \* prepared the inclusion of additional reference models
  - \* update to Introduction and Scope section
  - \* major update to (Normative) Prerequisites
  - \* relabeled Attestation Authenticity to Att. Evidence Authenticity
  - \* relabeled Assertion term back to Claim terms

- \* added BCP205 Implementation Status section related to Appendix CDDL
- o Changes from version 02 to version 03
  - \* major refactoring to now accommodate three interaction models
  - \* updated existing and added two new diagrams for models
  - \* major refactoring of existing and adding of new diagram description
  - \* incorporated content about Direct Anonymous Attestation
  - \* integrated comments from Michael Richardson
  - \* updated roster

## 14. References

### 14.1. 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://www.rfc-editor.org/info/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://www.rfc-editor.org/info/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://www.rfc-editor.org/info/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://www.rfc-editor.org/info/rfc5280>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

- [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>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

#### 14.2. Informative References

- [DAA] Brickell, E., Camenisch, J., and L. Chen, "Direct Anonymous Attestation", ACM Proceedings of the 11rd ACM conference on Computer and Communications Security , page 132-145, 2004.
- [I-D.birkholz-rats-tuda] Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-rats-tuda-02 (work in progress), March 2020.
- [I-D.ietf-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-04 (work in progress), May 2020.
- [turtles] Wikipedia, "Turtles all the way down", July 2020, <[https://en.wikipedia.org/wiki/Turtles\\_all\\_the\\_way\\_down](https://en.wikipedia.org/wiki/Turtles_all_the_way_down)>.

#### 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 transfer protocol used is CoAP using the FETCH operation. The actual resource operated on can be empty. Both the Challenge 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.

RAIM-Bodies = CoAP-FETCH-Body / CoAP-FETCH-Response-Body

```
CoAP-FETCH-Body = [ hello: bool, ; if true, the AK-Cert is conveyed
                    nonce: bytes,
                    pcr-selection: [ + [ tcg-hash-alg-id: uint .size 2, ; TPM2_AL
G_ID
                                [ + pcr: uint .size 1 ],
                                ],
                    ],
```

```
CoAP-FETCH-Response-Body = [ attestation-evidence: TPMS_ATTEST-quote,
                             tpm-native-signature: bytes,
                             ? ak-cert: bytes, ; attestation key certificate
                             ]
```

```
TPMS_ATTEST-quote = [ qualifiediSigner: uint .size 2, ;TPM2B_NAME
                     TPMS_CLOCK_INFO,
                     firmwareVersion: uint .size 8
                     quote-responses: [ * [ pcr: uint .size 1,
                                             + [ pcr-value: bytes,
                                                ? hash-alg-id: uint .size 2,
                                                ],
                                             ],
                                             ? pcr-digest: bytes,
                                             ],
                     ]
```

```
TPMS_CLOCK_INFO = [ clock: uint .size 8,
                    resetCounter: uint .size 4,
                    restartCounter: uint .size 4,
                    save: bool,
                    ]
```

#### Authors' Addresses

Henk Birkholz  
 Fraunhofer SIT  
 Rheinstrasse 75  
 Darmstadt 64295  
 Germany

Email: [henk.birkholz@sit.fraunhofer.de](mailto:henk.birkholz@sit.fraunhofer.de)

Michael Eckel  
Fraunhofer SIT  
Rheinstrasse 75  
Darmstadt 64295  
Germany

Email: michael.eckel@sit.fraunhofer.de

Christopher Newton  
University of Surrey

Email: cn0016@surrey.ac.uk

Liqun Chen  
University of Surrey

Email: liqun.chen@surrey.ac.uk

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

L. Lundblade  
Security Theory LLC  
G. Mandyam  
J. O'Donoghue  
Qualcomm Technologies Inc.  
February 23, 2022

The Entity Attestation Token (EAT)  
draft-ietf-rats-eat-12

Abstract

An Entity Attestation Token (EAT) provides an attested claims set that describes state and characteristics of an entity, a device like a phone, IoT device, network equipment or such. This claims set is used by a relying party, server or service to determine how much it wishes to trust the entity.

An EAT is either a CBOR Web Token (CWT) or JSON Web Token (JWT) with attestation-oriented claims. To a large degree, all this document does is extend CWT and JWT.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 27, 2022.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of



publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	5
1.1. Entity Overview . . . . .	6
1.2. CWT, JWT, UCCS, UJCS and DEB . . . . .	7
1.3. CDDL, CBOR and JSON . . . . .	8
1.4. Operating Model and RATS Architecture . . . . .	8
1.4.1. Relationship between Attestation Evidence and Attestation Results . . . . .	9
2. Terminology . . . . .	10
3. The Claims . . . . .	11
3.1. Token ID Claim (cti and jti) . . . . .	11
3.2. Timestamp claim (iat) . . . . .	11
3.3. Nonce Claim (nonce) . . . . .	12
3.4. Universal Entity ID Claim (ueid) . . . . .	12
3.5. Semi-permanent UEIDs (SUEIDs) . . . . .	15
3.6. Hardware OEM Identification (oemid) . . . . .	16
3.6.1. Random Number Based OEMID . . . . .	16
3.6.2. IEEE Based OEMID . . . . .	16
3.6.3. IANA Private Enterprise Number Based OEMID . . . . .	17
3.7. Hardware Model Claim (hardware-model) . . . . .	17
3.8. Hardware Version Claims (hardware-version-claims) . . . . .	18
3.9. Software Name Claim . . . . .	19
3.10. Software Version Claim . . . . .	19
3.11. The Security Level Claim (security-level) . . . . .	19
3.12. Secure Boot Claim (secure-boot) . . . . .	21
3.13. Debug Status Claim (debug-status) . . . . .	21
3.13.1. Enabled . . . . .	22
3.13.2. Disabled . . . . .	22
3.13.3. Disabled Since Boot . . . . .	22
3.13.4. Disabled Permanently . . . . .	22
3.13.5. Disabled Fully and Permanently . . . . .	22
3.14. Including Keys . . . . .	23
3.15. The Location Claim (location) . . . . .	24
3.16. The Uptime Claim (uptime) . . . . .	25
3.17. The Boot Odometer Claim (odometer) . . . . .	25
3.18. The Boot Seed Claim (boot-seed) . . . . .	25
3.19. The Intended Use Claim (intended-use) . . . . .	26
3.20. The Profile Claim (profile) . . . . .	27
3.21. The DLOA (Digital Letter or Approval) Claim (dloas) . . . . .	27
3.22. The Software Manifests Claim (manifests) . . . . .	28

3.23. The Software Evidence Claim (swevidence)	30
3.24. The SW Measurement Results Claim (swresults)	30
3.24.1. Scheme	31
3.24.2. Objective	31
3.24.3. Results	31
3.24.4. Objective Name	32
3.25. Submodules (submods)	34
3.25.1. Submodule Types	34
3.25.1.1. Submodule Claims-Set	34
3.25.1.2. Nested Token	35
3.25.1.3. Detached Submodule Digest	37
3.25.2. No Inheritance	38
3.25.3. Security Levels	38
3.25.4. Submodule Names	39
3.25.5. CDDL for submods	39
4. Unprotected JWT Claims-Sets	39
5. Detached EAT Bundles	39
6. Endorsements and Verification Keys	40
6.1. Identification Methods	41
6.1.1. COSE/JWS Key ID	41
6.1.2. JWS and COSE X.509 Header Parameters	42
6.1.3. CBOR Certificate COSE Header Parameters	42
6.1.4. Claim-Based Key Identification	42
6.2. Other Considerations	42
7. Profiles	43
7.1. Format of a Profile Document	43
7.2. List of Profile Issues	43
7.2.1. Use of JSON, CBOR or both	43
7.2.2. CBOR Map and Array Encoding	44
7.2.3. CBOR String Encoding	44
7.2.4. CBOR Preferred Serialization	44
7.2.5. COSE/JOSE Protection	44
7.2.6. COSE/JOSE Algorithms	45
7.2.7. DEB Support	45
7.2.8. Verification Key Identification	45
7.2.9. Endorsement Identification	45
7.2.10. Freshness	45
7.2.11. Required Claims	45
7.2.12. Prohibited Claims	45
7.2.13. Additional Claims	46
7.2.14. Refined Claim Definition	46
7.2.15. CBOR Tags	46
7.2.16. Manifests and Software Evidence Claims	46
8. Encoding and Collected CDDL	46
8.1. Claims-Set and CDDL for CWT and JWT	46
8.2. Encoding Data Types	47
8.2.1. Common Data Types	47
8.2.2. JSON Interoperability	47

8.2.3. Labels . . . . .	48
8.3. CBOR Interoperability . . . . .	48
8.3.1. EAT Constrained Device Serialization . . . . .	48
8.4. Collected Common CDDL . . . . .	49
8.5. Collected CDDL for CBOR . . . . .	54
8.6. Collected CDDL for JSON . . . . .	55
9. IANA Considerations . . . . .	56
9.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries . . . . .	56
9.2. Claim Characteristics . . . . .	57
9.2.1. Interoperability and Relying Party Orientation . . .	57
9.2.2. Operating System and Technology Neutral . . . . .	57
9.2.3. Security Level Neutral . . . . .	58
9.2.4. Reuse of Extant Data Formats . . . . .	58
9.2.5. Proprietary Claims . . . . .	58
9.3. Claims Registered by This Document . . . . .	58
9.3.1. Claims for Early Assignment . . . . .	59
9.3.2. To be Assigned Claims . . . . .	62
9.3.3. Version Schemes Registered by this Document . . . . .	65
9.3.4. UEID URN Registered by this Document . . . . .	66
9.3.5. Tag for Detached EAT Bundle . . . . .	66
10. Privacy Considerations . . . . .	66
10.1. UEID and SUEID Privacy Considerations . . . . .	67
10.2. Location Privacy Considerations . . . . .	67
10.3. Replay Protection and Privacy . . . . .	68
11. Security Considerations . . . . .	68
11.1. Key Provisioning . . . . .	68
11.1.1. Transmission of Key Material . . . . .	69
11.2. Transport Security . . . . .	69
11.3. Multiple EAT Consumers . . . . .	69
12. References . . . . .	70
12.1. Normative References . . . . .	70
12.2. Informative References . . . . .	73
Appendix A. Examples . . . . .	76
A.1. Simple TEE Attestation . . . . .	76
A.2. Submodules for Board and Device . . . . .	77
A.3. EAT Produced by Attestation Hardware Block . . . . .	79
A.4. Detached EAT Bundle . . . . .	79
A.5. Key / Key Store Attestation . . . . .	81
A.6. SW Measurements of an IoT Device . . . . .	83
A.7. Attestation Results in JSON format . . . . .	86
Appendix B. UEID Design Rationale . . . . .	87
B.1. Collision Probability . . . . .	87
B.2. No Use of UUID . . . . .	89
Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID) . . . . .	90
C.1. DevID Used With EAT . . . . .	90
C.2. How EAT Provides an Equivalent Secure Device Identity . .	91

C.3. An X.509 Format EAT . . . . .	91
C.4. Device Identifier Permanence . . . . .	92
Appendix D. Changes from Previous Drafts . . . . .	92
D.1. From draft-rats-eat-01 . . . . .	92
D.2. From draft-mandyam-rats-eat-00 . . . . .	92
D.3. From draft-ietf-rats-eat-01 . . . . .	92
D.4. From draft-ietf-rats-eat-02 . . . . .	93
D.5. From draft-ietf-rats-eat-03 . . . . .	93
D.6. From draft-ietf-rats-eat-04 . . . . .	93
D.7. From draft-ietf-rats-eat-05 . . . . .	94
D.8. From draft-ietf-rats-eat-06 . . . . .	94
D.9. From draft-ietf-rats-eat-07 . . . . .	94
D.10. From draft-ietf-rats-eat-08 . . . . .	94
D.11. From draft-ietf-rats-eat-09 . . . . .	94
D.12. From draft-ietf-rats-eat-10 . . . . .	95
D.13. From draft-ietf-rats-eat-11 . . . . .	96
Authors' Addresses . . . . .	96

## 1. Introduction

EAT provides the definition of a base set of claims that can be made about an entity, a device, some software and/or some hardware. This claims set is received by a relying party who uses it to decide if and how it will interact with the remote entity. It may choose to not trust the entity and not interact with it. It may choose to trust it. It may partially trust it, for example allowing monetary transactions only up to a limit.

EAT defines the encoding of the claims set in CBOR [RFC8949] and JSON [RFC7159]. EAT is an extension to CBOR Web Token (CWT) [RFC8392] and JSON Web Token (JWT) [RFC7519].

The claims set is secured in transit with the same mechanisms used by CWT and JWT, in particular CBOR Object Signing and Encryption (COSE) [RFC8152] and JSON Object Signing and Encryption (JOSE) [RFC7515] [RFC7516]. Authenticity and integrity protection must always be provided. Privacy (encryption) may additionally be provided. The key material used to sign and encrypt is specifically created and provisioned for the purpose of attestation. It is the use of this key material that make the claims set "attested" rather than just some parameters sent to the relying party by the device.

EAT is focused on authenticating, identifying and characterizing implementations where implementations are devices, chips, hardware, software and such. This is distinct from protocols like TLS [RFC8446] that authenticate and identify servers and services. It is equally distinct from protocols like SASL [RFC4422] that authenticate and identify persons.

The notion of attestation is large, ranging over a broad variety of use cases and security levels. Here are a few examples of claims:

- o Make and model of manufactured consumer device
- o Make and model of a chip or processor, particularly for a security-oriented chip
- o Identification and measurement of the software running on a device
- o Configuration and state of a device
- o Environmental characteristics of a device like its GPS location
- o Formal certifications received

EAT also supports nesting of sets of claims and EAT tokens for use with complex composite devices.

This document uses the terminology and main operational model defined in [RATS.Architecture]. In particular, it can be used for RATS Attestation Evidence and Attestation Results.

### 1.1. Entity Overview

The document uses the term "entity" to refer to the target of the attestation token. The claims defined in this document are claims about an entity.

An entity is an implementation in hardware, software or both.

An entity is the same as the Attester Target Environment defined in RATS Architecture.

An entity also corresponds to a "system component" as defined in the Internet Security Glossary [RFC4949]. That glossary also defines "entity" and "system entity" as something that may be a person or organization as well as a system component. Here "entity" never refers to a person or organization.

An entity is never a server or a service.

An entity may be the whole device or it may be a subsystem, a subsystem of a subsystem and so on. EAT allows claims to be organized into submodules, nested EATs and so on. See Section 3.25. The entity to which a claim applies is the submodule in which it appears, or to the top-level entity if it doesn't appear in a submodule.

Some examples of entities:

- o A Secure Element
- o A TEE
- o A card in a network router
- o A network router, perhaps with each card in the router a submodule
- o An IoT device
- o An individual process
- o An app on a smartphone
- o A smartphone with many submodules for its many subsystems
- o A subsystem in a smartphone like the modem or the camera

An entity may have strong security like defenses against hardware invasive attacks. It may also have low security, having no special security defenses. There is no minimum security requirement to be an entity.

#### 1.2. CWT, JWT, UCCS, UJCS and DEB

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

- o CBOR Web Token (CWT) [RFC8392]
- o Unprotected CWT Claims Sets (UCCS) [UCCS.Draft]
- o JSON Web Token (JWT) [RFC7519]

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

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

The identification of a protocol element as an EAT, whether CBOR or JSON encoded, follows the general conventions used by CWT, JWT and

UCCS. Largely this depends on the protocol carrying the EAT. In some cases it may be by content type (e.g., MIME type). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This specification adds two more top-level messages:

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

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

### 1.3. CDDL, CBOR and JSON

This document defines Concise Binary Object Representation (CBOR) [RFC8949] and Javascript Object Notation (JSON) [RFC7159] encoding for an EAT. All claims in an EAT MUST use the same encoding except where explicitly allowed. It is explicitly allowed for a nested token to be of a different encoding. Some claims explicitly contain objects and messages that may use a different encoding than the enclosing EAT.

This specification uses Concise Data Definition Language (CDDL) [RFC8610] for all definitions. The implementor interprets the CDDL to come to either the CBOR or JSON encoding. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 8.2.2 where Appendix E is insufficient.

The CWT and JWT specifications were authored before CDDL was available and did not use CDDL. This specification includes a CDDL definition of most of what is defined in [RFC8392]. Similarly, this specification includes CDDL for most of what is defined in [RFC7519].

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

### 1.4. Operating Model and RATS Architecture

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

To summarize, an Attester generates Attestation Evidence. Attestation Evidence is a claims set describing various characteristics of an entity. Attestation Evidence also is usually

signed by a key that proves the entity and the evidence it produces are authentic. The claims set includes a nonce or some other means to provide freshness. EAT is designed to carry Attestation Evidence. The Attestation Evidence goes to a Verifier where the signature is verified. Some of the claims may also be checked against Reference Values. The Verifier then produces Attestation Results which is also usually a claims set. EAT is also designed to carry Attestation Results. The Attestation Results go to the Relying Party which is the ultimate consumer of the Remote Attestation Procedure. The Relying Party uses the Attestation Results as needed for the use case, perhaps allowing an entity on the network, allowing a financial transaction or such.

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

#### 1.4.1. Relationship between Attestation Evidence and Attestation Results

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

Many claims in Attestation Evidence simply will pass through the Verifier to the Relying Party without modification. They will be verified as authentic from the entity by the Verifier just through normal verification of the Attester's signature. The UEID, Section 3.4, and Location, Section 3.15, are examples of claims that may be passed through.

Some claims in Attestation Evidence will be verified by the Verifier by comparison to Reference Values. These claims will not likely be conveyed to the Relying Party. Instead, some claim indicating they were checked may be added to the Attestation Results or it may be tacitly known that the Verifier always does this check. For example, the Verifier receives the Software Evidence claim, Section 3.23, compares it to Reference Values and conveys the results to the Relying Party in a Software Measurement Results Claim, Section 3.24.

In some cases the Verifier may provide privacy-preserving functionality by stripping or modifying claims that do not possess sufficient privacy-preserving characteristics. For example, the data in the Location claim, Section 3.15, may be modified to have a precision of a few kilometers rather than a few meters.



## 2. Terminology

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

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

**Claim:** A piece of information asserted about a subject. A claim is represented as pair with a value and either a name or key to identify it.

**Claim Name:** A unique text string that identifies the claim. It is used as the claim name for JSON encoding.

**Claim Key:** The CBOR map key used to identify a claim.

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

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

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

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

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

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

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

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

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

### 3. The Claims

This section describes new claims defined for attestation that are to be added to the CWT [IANA.CWT.Claims] and JWT [IANA.JWT.Claims] IANA registries.

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

CDDL, along with a text description, is used to define each claim independent of encoding. Each claim is defined as a CDDL group. In Section 8 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

Each claim described has a unique text string and integer that identifies it. CBOR encoded tokens MUST use only the integer for Claim Keys. JSON encoded tokens MUST use only the text string for Claim Names.

#### 3.1. Token ID Claim (cti and jti)

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

#### 3.2. Timestamp claim (iat)

The "iat" claim defined in CWT and JWT is used to indicate the date-of-creation of the token, the time at which the claims are collected and the token is composed and signed.

The data for some claims may be held or cached for some period of time before the token is created. This period may be long, even days. Examples are measurements taken at boot or a geographic

position fix taken the last time a satellite signal was received. There are individual timestamps associated with these claims to indicate their age is older than the "iat" timestamp.

CWT allows the use floating-point for this claim. EAT disallows the use of floating-point. An EAT token MUST NOT contain an iat claim in float-point format. Any recipient of a token with a floating-point format iat claim MUST consider it an error. A 64-bit integer representation of epoch time can represent a range of +/- 500 billion years, so the only point of a floating-point timestamp is to have precession greater than one second. This is not needed for EAT.

### 3.3. Nonce Claim (nonce)

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

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

The nonce must be at least 8 bytes (64 bits) long as fewer bytes are unlikely to be secure. A maximum of 64 bytes is set to limit the memory a constrained implementation uses. This size range is not set for the already-registered JWT nonce, but it should follow this size recommendation when used in an EAT.

Multiple nonces are allowed to accommodate multistage verification and consumption.

```
$$claims-set-claims //=  
    (nonce-label => nonce-type / [ 2* nonce-type ])  
  
nonce-type = bstr .size (8..64)
```

### 3.4. Universal Entity ID Claim (ueid)

A UEID identifies an individual manufactured entity like a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire entity or a submodule. It does not identify types, models or classes of entities. It is akin to a serial number, though it does not have to be sequential.

UEIDs MUST be universally and globally unique across manufacturers and countries. UEIDs MUST also be unique across protocols and systems, as tokens are intended to be embedded in many different

protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries should have the same UEID (if they are not global and universal in this way, then Relying Parties receiving them will have to track other characteristics of the entity to keep entities distinct between manufacturers).

There are privacy considerations for UEIDs. See Section 10.1.

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

A UEID is constructed of a single type byte followed by the bytes that are the identifier. Several types are allowed to accommodate different industries, different manufacturing processes and to have an alternative that doesn't require paying a registration fee.

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

UEIDs are variable length. All implementations MUST be able to receive UEIDs that are 33 bytes long (1 type byte and 256 bits). No UEID longer than 33 bytes SHOULD be sent.

Type Byte	Type Name	Specification
0x01	RAND	This is a 128, 192 or 256-bit random number generated once and stored in the entity. This may be constructed by concatenating enough identifiers to make up an equivalent number of random bits and then feeding the concatenation through a cryptographic hash function. It may also be a cryptographic quality random number generated once at the beginning of the life of the entity and stored. It MUST NOT be smaller than 128 bits. See the length analysis in Appendix B.
0x02	IEEE EUI	This uses the IEEE company identification registry. An EUI is either an EUI-48, EUI-60 or EUI-64 and made up of an OUI, OUI-36 or a CID, different registered company identifiers, and some unique per-entity identifier. EUIs are often the same as or similar to MAC addresses. This type includes MAC-48, an obsolete name for EUI-48. (Note that while entities with multiple network interfaces may have multiple MAC addresses, there is only one UEID for an entity) [IEEE.802-2001], [OUI.Guide].
0x03	IMEI	This is a 14-digit identifier consisting of an 8-digit Type Allocation Code and a 6-digit serial number allocated by the manufacturer, which SHALL be encoded as byte string of length 14 with each byte as the digit's value (not the ASCII encoding of the digit; the digit 3 encodes as 0x03, not 0x33). The IMEI value encoded SHALL NOT include Luhn checksum or SVN information. See [ThreeGPP.IMEI].

Table 1: UEID Composition Types

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

The consumer (the Relying Party) of a UEID MUST treat a UEID as a completely opaque string of bytes and not make any use of its internal structure. For example, they should not use the OUI part of a type 0x02 UEID to identify the manufacturer of the entity. Instead, they should use the OEMID claim. See Section 3.6. The reasons for this are:

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

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

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

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

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

### 3.5. Semi-permanent UEIDs (SUEIDs)

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

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

There are privacy considerations for SUEIDs. See Section 10.1.

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

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

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

### 3.6. Hardware OEM Identification (oemid)

This claim identifies the Original Equipment Manufacturer (OEM) of the hardware. Any of the three forms described below MAY be used at the convenience of the claim sender. The receiver of this claim MUST be able to handle all three forms.

#### 3.6.1. Random Number Based OEMID

The random number based OEMID MUST always 16 bytes (128 bits).

The OEM MAY create their own ID by using a cryptographic-quality random number generator. They would perform this only once in the life of the company to generate the single ID for said company. They would use that same ID in every entity they make. This uniquely identifies the OEM on a statistical basis and is large enough should there be ten billion companies.

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

In JSON format tokens this MUST be base64url encoded.

#### 3.6.2. IEEE Based OEMID

The IEEE operates a global registry for MAC addresses and company IDs. This claim uses that database to identify OEMs. The contents of the claim may be either an IEEE MA-L, MA-M, MA-S or an IEEE CID [IEEE.RA]. An MA-L, formerly known as an OUI, is a 24-bit value used as the first half of a MAC address. MA-M similarly is a 28-bit value uses as the first part of a MAC address, and MA-S, formerly known as OUI-36, a 36-bit value. Many companies already have purchased one of these. A CID is also a 24-bit value from the same space as an MA-L, but not for use as a MAC address. IEEE has published Guidelines for Use of EUI, OUI, and CID [OUI.Guide] and provides a lookup service [OUI.Lookup].

Companies that have more than one of these IDs or MAC address blocks SHOULD select one and prefer that for all their entities.

Commonly, these are expressed in Hexadecimal Representation as described in [IEEE.802-2001]. It is also called the Canonical format. When this claim is encoded the order of bytes in the bstr are the same as the order in the Hexadecimal Representation. For

example, an MA-L like "AC-DE-48" would be encoded in 3 bytes with values 0xAC, 0xDE, 0x48.

This format is always 3 bytes in size in CBOR.

In JSON format tokens, this MUST be base64url encoded and always 4 bytes.

### 3.6.3. IANA Private Enterprise Number Based OEMID

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

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

In CBOR this value MUST be encoded as a major type 0 integer and is typically 3 bytes. In JSON, this value MUST be encoded as a number.

```
oemid-pen = int
```

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

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

```
$$claims-set-claims //= (  
    oemid-label =>  
        oemid-random / oemid-ieee / oemid-pen  
)
```

### 3.7. Hardware Model Claim (hardware-model)

This claim differentiates hardware models, products and variants manufactured by a particular OEM, the one identified by OEM ID in Section 3.6.

This claim must be unique so as to differentiate the models and products for the OEM ID. This claim does not have to be globally unique, but it can be. A receiver of this claim MUST not assume it is globally unique. To globally identify a particular product, the receiver should concatenate the OEM ID and this claim.

The granularity of the model identification is for each OEM to decide. It may be very granular, perhaps including some version information. It may be very general, perhaps only indicating top-level products.



The purpose of this claim is to identify models within protocols, not for human-readable descriptions. The format and encoding of this claim should not be human-readable to discourage use other than in protocols. If this claim is to be derived from an already-in-use human-readable identifier, it can be run through a hash function.

There is no minimum length so that an OEM with a very small number of models can use a one-byte encoding. The maximum length is 32 bytes. All receivers of this claim MUST be able to receive this maximum size.

The receiver of this claim MUST treat it as a completely opaque string of bytes, even if there is some apparent naming or structure. The OEM is free to alter the internal structure of these bytes as long as the claim continues to uniquely identify its models.

```
hardware-model-type = bytes .size (1..32)
```

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

### 3.8. Hardware Version Claims (hardware-version-claims)

The hardware version is a text string the format of which is set by each manufacturer. The structure and sorting order of this text string can be specified using the version-scheme item from CoSWID [CoSWID]. It is useful to know how to sort versions so the newer can be distinguished from the older.

The hardware version can also be given by a 13-digit [EAN-13]. A new CoSWID version scheme is registered with IANA by this document in Section 9.3.3. An EAN-13 is also known as an International Article Number or most commonly as a bar code.

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

```
hardware-version-type = [
    version:  tstr,
    scheme:   $version-scheme
]
```

### 3.9. Software Name Claim

This is a free-form text claim for the name of the software for the entity or submodule. A CoSWID manifest or other type of manifest can be used instead if this claim is too limited to correctly characterize the SW for the entity or submodule.

```
$$claims-set-claims //= ( sw-name-label => tstr )
```

### 3.10. Software Version Claim

This makes use of the CoSWID version scheme data type to give a simple version for the software. A full CoSWID manifest or other type of manifest can be used instead if this is too simple.

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

```
sw-version-type = [  
    version:  tstr,  
    scheme:   $version-scheme ; As defined by CoSWID  
]
```

### 3.11. The Security Level Claim (security-level)

This claim characterizes the entity's ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is by defining four security levels.

This claim describes the security environment and countermeasures available on the entity where the attestation key resides and the claims originate.

- 1 - Unrestricted: There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise, the EAT provides no meaningful security assurances.
- 2 - Restricted: Entities at this level are not general-purpose operating environments that host features, such as app download systems, web browsers and complex applications. It is akin to the secure-restricted level (see below) without the security orientation. Examples include a Wi-Fi subsystem, an IoT camera, or sensor device. Often these can be considered more secure than unrestricted just because they are much simpler and a smaller attack surface, but this won't always be the case. Some unrestricted devices may be implemented in a way that provides poor protection of signing keys.

- 3 - Secure-Restricted: Entities at this level must meet the criteria defined in Section 4 of FIDO Allowed Restricted Operating Environments [FIDO.AROE]. Examples include TEE's and schemes using virtualization-based security. Security at this level is aimed at defending against large-scale network/remote attacks against the entity.
- 4 - Hardware: Entities at this level must include substantial defense against physical or electrical attacks against the entity itself. It is assumed the potential attacker has captured the entity and can disassemble it. Examples include TPMs and Secure Elements.

The entity should claim the highest security level it achieves and no higher. This set is not extensible so as to provide a common interoperable description of security level to the Relying Party. If a particular use case considers this claim to be inadequate, it can define its own proprietary claim. It may consider including both this claim as a coarse indication of security and its own proprietary claim as a refined indication.

This claim is not intended as a replacement for a formal security certification scheme, such as those based on FIPS 140 [FIPS-140] or those based on Common Criteria [Common.Criteria]. See Section 3.21.

```
$$claims-set-claims // = (
    security-level-label =>
        security-level-chor-type /
        security-level-json-type
)
```

```
security-level-chor-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)
```

```
security-level-json-type =
    "unrestricted" /
    "restricted" /
    "secure-restricted" /
    "hardware"
```

### 3.12. Secure Boot Claim (secure-boot)

The value of true indicates secure boot is enabled. Secure boot is considered enabled when the firmware and operating system, are under control of the manufacturer of the entity identified in the OEMID claim described in Section 3.6. Control by the manufacturer of the firmware and the operating system may be by it being in ROM, being cryptographically authenticated, a combination of the two or similar.

```
$$claims-set-claims // = (secure-boot-label => bool)
```

### 3.13. Debug Status Claim (debug-status)

This applies to entity-wide or submodule-wide debug facilities of the entity like JTAG and diagnostic hardware built into chips. It applies to any software debug facilities related to root, operating system or privileged software that allow system-wide memory inspection, tracing or modification of non-system software like user mode applications.

This characterization assumes that debug facilities can be enabled and disabled in a dynamic way or be disabled in some permanent way such that no enabling is possible. An example of dynamic enabling is one where some authentication is required to enable debugging. An example of permanent disabling is blowing a hardware fuse in a chip. The specific type of the mechanism is not taken into account. For example, it does not matter if authentication is by a global password or by per-entity public keys.

As with all claims, the absence of the debug level claim means it is not reported. A conservative interpretation might assume the enabled state.

This claim is not extensible so as to provide a common interoperable description of debug status. If a particular implementation considers this claim to be inadequate, it can define its own proprietary claim. It may consider including both this claim as a coarse indication of debug status and its own proprietary claim as a refined indication.

The higher levels of debug disabling requires that all debug disabling of the levels below it be in effect. Since the lowest level requires that all of the target's debug be currently disabled, all other levels require that too.

There is no inheritance of claims from a submodule to a superior module or vice versa. There is no assumption, requirement or guarantee that the target of a superior module encompasses the

targets of submodules. Thus, every submodule must explicitly describe its own debug state. The receiver of an EAT MUST not assume that debug is turned off in a submodule because there is a claim indicating it is turned off in a superior module.

An entity may have multiple debug facilities. The use of plural in the description of the states refers to that, not to any aggregation or inheritance.

The architecture of some chips or devices may be such that a debug facility operates for the whole chip or device. If the EAT for such a chip includes submodules, then each submodule should independently report the status of the whole-chip or whole-device debug facility. This is the only way the receiver can know the debug status of the submodules since there is no inheritance.

#### 3.13.1. Enabled

If any debug facility, even manufacturer hardware diagnostics, is currently enabled, then this level must be indicated.

#### 3.13.2. Disabled

This level indicates all debug facilities are currently disabled. It may be possible to enable them in the future. It may also be that they were enabled in the past, but they are currently disabled.

#### 3.13.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the entity booted/started.

#### 3.13.4. Disabled Permanently

This level indicates all non-manufacturer facilities are permanently disabled such that no end user or developer can enable them. Only the manufacturer indicated in the OEMID claim can enable them. This also indicates that all debug facilities are currently disabled and have been so since boot/start.

#### 3.13.5. Disabled Fully and Permanently

This level indicates that all debug facilities for the entity are permanently disabled.

```
$$claims-set-claims // = (
    debug-status-label =>
        debug-status-cbor-type / debug-status-json-type
)

debug-status-cbor-type = &(
    enabled: 0,
    disabled: 1,
    disabled-since-boot: 2,
    disabled-permanently: 3,
    disabled-fully-and-permanently: 4
)

debug-status-json-type =
    "enabled" /
    "disabled" /
    "disabled-since-boot" /
    "disabled-permanently" /
    "disabled-fully-and-permanently"
```

### 3.14. Including Keys

An EAT may include a cryptographic key such as a public key. The signing of the EAT binds the key to all the other claims in the token.

The purpose for inclusion of the key may vary by use case. For example, the key may be included as part of an IoT device onboarding protocol. When the FIDO protocol includes a public key in its attestation message, the key represents the binding of a user, device and Relying Party. This document describes how claims containing keys should be defined for the various use cases. It does not define specific claims for specific use cases.

Keys in CBOR format tokens SHOULD be the COSE\_Key format [RFC8152] and keys in JSON format tokens SHOULD be the JSON Web Key format [RFC7517]. These two formats support many common key types. Their use avoids the need to decode other serialization formats. These two formats can be extended to support further key types through their IANA registries.

The general confirmation claim format [RFC8747], [RFC7800] may also be used. It provides key encryption. It also allows for inclusion by reference through a key ID. The confirmation claim format may be employed in the definition of some new claim for a particular use case.

When the actual confirmation claim is included in an EAT, this document associates no use case semantics other than proof of possession. Different EAT use cases may choose to associate further semantics. The key in the confirmation claim **MUST** be protected in the same way as the key used to sign the EAT. That is, the same, equivalent or better hardware defenses, access controls, key generation and such must be used.

### 3.15. The Location Claim (location)

The location claim gives the location of the entity from which the attestation originates. It is derived from the W3C Geolocation API [W3C.GeoLoc]. The latitude, longitude, altitude and accuracy must conform to [WGS84]. The altitude is in meters above the [WGS84] ellipsoid. The two accuracy values are positive numbers in meters. The heading is in degrees relative to true north. If the entity is stationary, the heading is NaN (floating-point not-a-number). The speed is the horizontal component of the entity velocity in meters per second.

When encoding floating-point numbers half-precision **SHOULD NOT** be used. They usually do not provide enough precision for a geographic location.

The location may have been cached for a period of time before token creation. For example, it might have been minutes or hours or more since the last contact with a GPS satellite. Either the timestamp or age data item can be used to quantify the cached period. The timestamp data item is preferred as it a non-relative time.

The age data item can be used when the entity doesn't know what time it is either because it doesn't have a clock or it isn't set. The entity **MUST** still have a "ticker" that can measure a time interval. The age is the interval between acquisition of the location data and token creation.

See location-related privacy considerations in Section 10.2.

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

```
location-type = {  
    latitude => number,  
    longitude => number,  
    ? altitude => number,  
    ? accuracy => number,  
    ? altitude-accuracy => number,  
    ? heading => number,  
    ? speed => number,  
    ? timestamp => ~time-int,  
    ? age => uint  
}  
  
latitude = 1 / "latitude"  
longitude = 2 / "longitude"  
altitude = 3 / "altitude"  
accuracy = 4 / "accuracy"  
altitude-accuracy = 5 / "altitude-accuracy"  
heading = 6 / "heading"  
speed = 7 / "speed"  
timestamp = 8 / "timestamp"  
age = 9 / "age"
```

### 3.16. The Uptime Claim (uptime)

The "uptime" claim MUST contain a value that represents the number of seconds that have elapsed since the entity or submod was last booted.

```
$$claims-set-claims // = (uptime-label => uint)
```

### 3.17. The Boot Odometer Claim (odometer)

The "odometer" claim contains a value that represents the number of times the entity or submod has been booted. Support for this claim requires a persistent storage on the device.

```
$$claims-set-claims // = (odometer-label => uint)
```

### 3.18. The Boot Seed Claim (boot-seed)

The Boot Seed claim MUST contain a random value created at system boot time that will allow differentiation of reports from different boot sessions.

This value is usually public. It is not a secret and MUST NOT be used for any purpose that a secret seed is needed, such as seeding a random number generator.



`$$claims-set-claims // = (boot-seed-label => bytes)`

### 3.19. The Intended Use Claim (intended-use)

EAT's may be used in the context of several different applications. The intended-use claim provides an indication to an EAT consumer about the intended usage of the token. This claim can be used as a way for an application using EAT to internally distinguish between different ways it uses EAT.

- 1 - Generic: Generic attestation describes an application where the EAT consumer requires the most up-to-date security assessment of the attesting entity. It is expected that this is the most commonly-used application of EAT.
- 2- Registration: Entities that are registering for a new service may be expected to provide an attestation as part of the registration process. This intended-use setting indicates that the attestation is not intended for any use but registration.
- 3 - Provisioning: Entities may be provisioned with different values or settings by an EAT consumer. Examples include key material or device management trees. The consumer may require an EAT to assess entity security state of the entity prior to provisioning.
- 4 - Certificate Issuance Certification Authorities (CA's) may require attestations prior to the issuance of certificates related to keypairs hosted at the entity. An EAT may be used as part of the certificate signing request (CSR).
- 5 - Proof-of-Possession: An EAT consumer may require an attestation as part of an accompanying proof-of-possession (PoP) application. More precisely, a PoP transaction is intended to provide to the recipient cryptographically-verifiable proof that the sender has possession of a key. This kind of attestation may be necessary to verify the security state of the entity storing the private key used in a PoP application.

```
$$claims-set-claims //= (
  intended-use-label =>
    intended-use-cbor-type / intended-use-json-type
)

intended-use-cbor-type = &(
  generic: 1,
  registration: 2,
  provisioning: 3,
  csr: 4,
  pop: 5
)

intended-use-json-type =
  "generic" /
  "registration" /
  "provisioning" /
  "csr" /
  "pop"
```

### 3.20. The Profile Claim (profile)

See Section 7 for the detailed description of a profile.

A profile is identified by either a URL or an OID. Typically, the URI will reference a document describing the profile. An OID is just a unique identifier for the profile. It may exist anywhere in the OID tree. There is no requirement that the named document be publicly accessible. The primary purpose of the profile claim is to uniquely identify the profile even if it is a private profile.

The OID is always absolute and never relative. In CBOR tokens, the OID MUST be encoded according to [RFC9090] and the URI according to [RFC8949]. Both are unwrapped and thus not CBOR tags. In JSON tokens, the OID is a string of the form "X.X.X", and a URI is a normal URI string.

Note that this is named "eat\_profile" for JWT and is distinct from the already registered "profile" claim in the JWT claims registry.

```
$$claims-set-claims //= (profile-label => ~uri / ~oid)
```

### 3.21. The DLOA (Digital Letter or Approval) Claim (dloas)

A DLOA (Digital Letter of Approval) [DLOA] is an XML document that describes a certification that an entity has received. Examples of certifications represented by a DLOA include those issued by Global Platform and those based on Common Criteria. The DLOA is unspecific

to any particular certification type or those issued by any particular organization.

This claim is typically issued by a Verifier, not an Attester. When this claim is issued by a Verifier, it MUST be because the entity has received the certification in the DLOA.

This claim MAY contain more than one DLOA. If multiple DLOAs are present, it MUST be because the entity received all of the certifications.

DLOA XML documents are always fetched from a registrar that stores them. This claim contains several data items used to construct a URL for fetching the DLOA from the particular registrar.

This claim MUST be encoded as an array with either two or three elements. The first element MUST be the URI for the registrar. The second element MUST be a platform label indicating which platform was certified. If the DLOA applies to an application, then the third element is added which MUST be an application label. The method of constructing the registrar URI, platform label and possibly application label is specified in [DLOA].

```
$$claims-set-claims // = (  
    dloas-label => [ + dloa-type ]  
)
```

```
dloa-type = [  
    dloa_registrar: ~uri  
    dloa_platform_label: text  
    ? dloa_application_label: text  
]
```

### 3.22. The Software Manifests Claim (manifests)

This claim contains descriptions of software present on the entity. These manifests are installed on the entity when the software is installed or are created as part of the installation process. Installation is anything that adds software to the entity, possibly factory installation, the user installing elective applications and so on. The defining characteristic is they are created by the software manufacturer. The purpose of these claims in an EAT is to relay them without modification to the Verifier and possibly to the Relying Party.

Some manifests may be signed by their software manufacturer before they are put into this EAT claim. When such manifests are put into this claim, the manufacturer's signature SHOULD be included. For

example, the manifest might be a CoSWID signed by the software manufacturer, in which case the full signed CoSWID should be put in this claim.

This claim allows multiple formats for the manifest. For example, the manifest may be a CBOR-format CoSWID, an XML-format SWID or other. Identification of the type of manifest is always by a CBOR tag. In many cases, for examples CoSWID, a tag will already be registered with IANA. If not, a tag MUST be registered. It can be in the first-come-first-served space which has minimal requirements for registration.

The claim is an array of one or more manifests. To facilitate hand off of the manifest to a decoding library, each manifest is contained in a byte string. This occurs for CBOR-format manifests as well as non-CBOR format manifests.

If a particular manifest type uses CBOR encoding, then the item in the array for it MUST be a byte string that contains a CBOR tag. The EAT decoder must decode the byte string and then the CBOR within it to find the tag number to identify the type of manifest. The contents of the byte string is then handed to the particular manifest processor for that type of manifest. CoSWID and SUIT manifest are examples of this.

If a particular manifest type does not use CBOR encoding, then the item in the array for it MUST be a CBOR tag that contains a byte string. The EAT decoder uses the tag to identify the processor for that type of manifest. The contents of the tag, the byte string, are handed to the manifest processor. Note that a byte string is used to contain the manifest whether it is a text based format or not. An example of this is an XML format ISO/IEC 19770 SWID.

It is not possible to describe the above requirements in CDDL, so the type for an individual manifest is any in the CDDL below. The above text sets the encoding requirement.

This claim allows for multiple manifests in one token since multiple software packages are likely to be present. The multiple manifests MAY be of multiple formats. In some cases EAT submodules may be used instead of the array structure in this claim for multiple manifests.

When the [CoSWID] format is used, it MUST be a payload CoSWID, not an evidence CoSWID.

```
$$claims-set-claims //= (  
    manifests-label => manifests-type  
)  
  
manifests-type = [+ $$manifest-formats]  
  
coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>  
  
$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx
```

### 3.23. The Software Evidence Claim (swevidence)

This claim contains descriptions, lists, evidence or measurements of the software that exists on the entity. The defining characteristic of this claim is that its contents are created by processes on the entity that inventory, measure or otherwise characterize the software on the entity. The contents of this claim do not originate from the software manufacturer.

This claim uses the same mechanism for identification of the type of the swevidence as is used for the type of the manifest in the manifests claim. It also uses the same byte string based mechanism for containing the claim and easing the hand off to a processing library. See the discussion above in the manifests claim.

When the [CoSWID] format is used, it MUST be evidence CoSWIDs, not payload CoSWIDS.

```
$$claims-set-claims //= (  
    swevidence-label => swevidence-type  
)  
  
swevidence-type = [+ $$swevidence-formats]  
  
coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>  
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag
```

### 3.24. The SW Measurement Results Claim (swresults)

This claims reports the outcome of the comparison of a measurement on some software to the expected Reference Values. It may report a successful comparison, failed comparison or other.

This claim MAY be generated by the Verifier and sent to the Relying Party. For example, it could be the results of the Verifier comparing the contents of the swevidence claim to Reference Values.

This claim MAY also be generated on the entity if the entity has the ability for one subsystem to measure another subsystem. For example, a TEE might have the ability to measure the software of the rich OS and may have the Reference Values for the rich OS.

Within an attestation target or submodule, multiple results can be reported. For example, it may be desirable to report the results for the kernel and each individual application separately.

For each software objective, the following can be reported. TODO: defined objective

#### 3.24.1. Scheme

This is the free-form text name of the verification system or scheme that performed the verification. There is no official registry of schemes or systems. It may be the name of a commercial product or such.

#### 3.24.2. Objective

This roughly characterizes the coverage of the software measurement software. This corresponds to the attestation target or the submodule. If all of the indicated target is not covered, the measurement must indicate partial.

- 1 - all: Indicates all the software has been verified, for example, all the software in the attestation target or the submodule
- 2 - firmware: Indicates all of and only the firmware
- 3 - kernel: Refers to all of the most-privileged software, for example the Linux kernel
- 4 - privileged: Refers to all of the software used by the root, system or administrative account
- 5 - system-libs: Refers to all of the system libraries that are broadly shared and used by applications and such
- 6 - partial: Some other partial set of the software

#### 3.24.3. Results

This describes the result of the measurement and also the comparison to Reference Values.

- 1 - verification-not-run: Indicates that no attempt was made to run the verification
- 2 - verification-indeterminate: The verification was attempted, but it did not produce a result; perhaps it ran out of memory, the battery died or such
- 3 - verification-failed: The verification ran to completion, the comparison was completed and did not compare correctly to the Reference Values
- 4 - fully-verified: The verification ran to completion and all measurements compared correctly to Reference Values
- 5 - partially-verified: The verification ran to completion and some, but not all, measurements compared correctly to Reference Values

#### 3.24.4. Objective Name

This is a free-form text string that describes the objective. For example, "Linux kernel" or "Facebook App"

```
$$claims-set-claims // = (swresults-label => [ + swresult-type ])  
  
verification-result-cbor-type = &(  
    verification-not-run: 1,  
    verification-indeterminate: 2,  
    verification-failed: 3,  
    fully-verified: 4,  
    partially-verified: 5,  
)  
  
verification-result-json-type =  
    "verification-not-run" /  
    "verification-indeterminate" /  
    "verification-failed" /  
    "fully-verified" /  
    "partially-verified"  
  
verification-objective-cbor-type = &(  
    all: 1,  
    firmware: 2,  
    kernel: 3,  
    privileged: 4,  
    system-libs: 5,  
    partial: 6,  
)  
  
verification-objective-json-type =  
    "all" /  
    "firmware" /  
    "kernel" /  
    "privileged" /  
    "system-libs" /  
    "partial"  
  
swresult-type = [  
    verification-system: tstr,  
    objective: verification-objective-cbor-type /  
        verification-objective-json-type,  
    result: verification-result-cbor-type /  
        verification-result-json-type,  
    ? objective-name: tstr  
]
```



### 3.25. Submodules (submods)

Some devices are complex, having many subsystems. A mobile phone is a good example. It may have several connectivity subsystems for communications (e.g., Wi-Fi and cellular). It may have subsystems for low-power audio and video playback. It may have multiple security-oriented subsystems like a TEE and a Secure Element.

The claims for a subsystem can be grouped together in a submodule or submod.

The submods are in a single map/object, one entry per submodule. There is only one submods map/object in a token. It is identified by its specific label. It is a peer to other claims, but it is not called a claim because it is a container for a claims set rather than an individual claim. This submods part of a token allows what might be called recursion. It allows claims sets inside of claims sets inside of claims sets...

#### 3.25.1. Submodule Types

The following sections define the three types of submodules:

- o A submodule Claims-Set
- o A nested token, which can be any valid EAT token, CBOR or JSON
- o The digest of a detached Claims-Set

##### 3.25.1.1. Submodule Claims-Set

This is a subordinate Claims-Set containing claims about the submodule.

The submodule claims-set is produced by the same Attester as the surrounding token. It is secured using the same mechanism as the enclosing token (e.g., it is signed by the same attestation key). It roughly corresponds to an Attester Target Environment, as described in the RATS architecture.

It may contain claims that are the same as its surrounding token or superior submodules. For example, the top-level of the token may have a UEID, a submod may have a different UEID and a further subordinate submodule may also have a UEID.

The encoding of a submodule Claims-Set MUST be the same as the encoding as the token it is part of.

This data type for this type of submodule is a map/object. It is identified when decoding by it's type being a map/object.

#### 3.25.1.2. Nested Token

This type of submodule is a fully formed complete token. It is typically produced by a separate Attester. It is typically used by a Composite Device as described in RATS Architecture [RATS.Architecture] In being a submodule of the surrounding token, it is cryptographically bound to the surrounding token. If it was conveyed in parallel with the surrounding token, there would be no such binding and attackers could substitute a good attestation from another device for the attestation of an errant subsystem.

A nested token does not need to use the same encoding as the enclosing token. This is to allow Composite Devices to be built without regards to the encoding supported by their Attesters. Thus a CBOR-encoded token like a CWT or UCCS can have a JWT as a nested token submodule and a JSON-encoded token can have a CWT or UCCS as a nested token submodule.

The following two sections describe how to encode and decode a nested token.

##### 3.25.1.2.1. Surrounding EAT is CBOR-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a CBOR-encoded token.

If the nested token is CBOR-encoded, then it MUST be a CBOR tag and MUST be wrapped in a byte string. The tag identifies whether the nested token is a CWT, a UCCS, a CBOR-encoded DEB, or some other CBOR-format token defined in the future. A nested CBOR-encoded token that is not a CBOR tag is NOT allowed.

If the nested token is JSON-encoded, then the data item MUST be a text string. The text string MUST contain a JSON-encoded array of two items. The first item is a string identifying the type of the token. The second item is the JSON-encoded token.

The string identifying the JSON-encoded token MUST be one of the following:

"JWT": The second item MUST be a JWT formatted according to [RFC7519]

"UJCS": The second item MUST be a UJCS-Message as defined in this document.

"DEB": The second item MUST be a JSON-encoded Detached EAT Bundle as defined in this document.

The definition of additional types requires a standards action.

When decoding, if a byte string is encountered, it is known to be a nested CBOR-encoded token. The byte string wrapping is removed. The type of the token is determined by the CBOR tag.

When decoding, if a text string is encountered, it is known to be a JSON-encoded token. The two-item array is decoded and tells the type of the JSON-encoded token.

```
Nested-Token =  
  tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)  
  bstr .cbor Tagged-CBOR-Token
```

#### 3.25.1.2.2. Surrounding EAT is JSON-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a JSON-encoded token.

The nested token MUST be an array of two in the same format as described in the section above.

A CBOR-encoded token nested inside a JSON-encoded MUST use the same array of two, but with the type as follows:

"CBOR": Some base64url-encoded CBOR that is a tag, typically a CWT, UCCS or CBOR-encoded DEB

When decoding, the array of two is decoded. The first item indicates the type and encoding of the nested token. If the type string is not "CBOR", then the token is JSON-encoded and of the type indicated by the string.

If the type string is "CBOR", then the token is CBOR-encoded. The base64url encoding is removed. The CBOR-encoded data is then decoded. The type of nested token is determined by the CBOR-tag. It is an error if the CBOR is not a tag.

```
Nested-Token = [  
  type : "JWT" / "CBOR" / "UJCS" / "DEB",  
  nested-token : JWT-Message /  
                  B64URL-Tagged-CBOR-Token /  
                  DEB-JSON-Message /  
                  UJCS-Message  
]
```

```
B64URL-Tagged-CBOR-Token = tstr .regexp "[A-Za-z0-9_=-]+"
```

### 3.25.1.3. Detached Submodule Digest

This is type of submodule equivalent to a Claims-Set submodule, except the Claims-Set is conveyed separately outside of the token.

This type of submodule consists of a digest made using a cryptographic hash of a Claims-Set. The Claims-Set is not included in the token. It is conveyed to the Verifier outside of the token. The submodule containing the digest is called a detached digest. The separately conveyed Claims-Set is called a detached claims set.

The input to the digest is exactly the byte-string wrapped encoded form of the Claims-Set for the submodule. That Claims-Set can include other submodules including nested tokens and detached digests.

The primary use for this is to facilitate the implementation of a small and secure attester, perhaps purely in hardware. This small, secure attester implements COSE signing and only a few claims, perhaps just UEID and hardware identification. It has inputs for digests of submodules, perhaps 32-byte hardware registers. Software running on the device constructs larger claim sets, perhaps very large, encodes them and digests them. The digests are written into the small secure attesters registers. The EAT produced by the small secure attester only contains the UEID, hardware identification and digests and is thus simple enough to be implemented in hardware. Probably, every data item in it is of fixed length.

The integrity protection for the larger Claims Sets will not be as secure as those originating in hardware block, but the key material and hardware-based claims will be. It is possible for the hardware to enforce hardware access control (memory protection) on the digest registers so that some of the larger claims can be more secure. For example, one register may be writable only by the TEE, so the detached claims from the TEE will have TEE-level security.

The data type for this type of submodule MUST be an array. It contains two data items, an algorithm identifier and a byte string containing the digest.

When decoding a CBOR format token the detached digest type is distinguished from the other types by it being an array. In CBOR the none of other submodule types are arrays.

When decoding a JSON format token, a little more work is required because both the nested token and detached digest types are an array. To distinguish the nested token from the detached digest, the first element in the array is examined. If it is "JWT", "UJCS" or "DEB", the the submodule is a nested token. Otherwise it will contain an algorithm identifier and is a detached digest.

A DEB, described in Section 5, may be used to convey detached claims sets and the token with their detached digests. EAT, however, doesn't require use of a DEB. Any other protocols may be used to convey detached claims sets and the token with their detached digests. Note that since detached Claims-Sets are usually signed, protocols conveying them must make sure they are not modified in transit.

### 3.25.2. No Inheritance

The subordinate modules do not inherit anything from the containing token. The subordinate modules must explicitly include all of their claims. This is the case even for claims like the nonce.

This rule is in place for simplicity. It avoids complex inheritance rules that might vary from one type of claim to another.

### 3.25.3. Security Levels

The security level of the non-token subordinate modules should always be less than or equal to that of the containing modules in the case of non-token submodules. It makes no sense for a module of lesser security to be signing claims of a module of higher security. An example of this is a TEE signing claims made by the non-TEE parts (e.g. the high-level OS) of the device.

The opposite may be true for the nested tokens. They usually have their own more secure key material. An example of this is an embedded secure element.

#### 3.25.4. Submodule Names

The label or name for each submodule in the submods map is a text string naming the submodule. No submodules may have the same name.

#### 3.25.5. CDDL for submods

The submodule type is distinguished in the encoded bytes by its data type, map/object for a Claims-Set, string for nested token and array for a detached submodule. Nested tokens are byte-string wrapped when encoded in CBOR and base64 encoded for JSON.

```
$$claims-set-claims // = (submods-label => { + text => Submodule })
```

```
Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest
```

```
Detached-Submodule-Digest = [  
    algorithm : int / text,  
    digest : bstr  
]
```

### 4. Unprotected JWT Claims-Sets

This is simply the JSON equivalent of an Unprotected CWT Claims-Set [UCCS.Draft].

It has no protection of its own so protections must be provided by the protocol carrying it. These are extensively discussed in [UCCS.Draft]. All the security discussion and security considerations in [UCCS.Draft] apply to UJCS.

(Note: The EAT author is open to this definition being moved into the UCCS draft, perhaps along with the related CDDL. It is place here for now so that the current UCCS draft plus this document are complete. UJCS is needed for the same use cases that a UCCS is needed. Further, JSON will commonly be used to convey Attestation Results since JSON is common for server to server communications. Server to server communications will often have established security (e.g., TLS) therefore the signing and encryption from JWS and JWE are unnecessary and burdensome).

### 5. Detached EAT Bundles

A detached EAT bundle is a structure to convey a fully-formed and signed token plus detached claims set that relate to that token. It is a top-level EAT message like a CWT, JWT, UCCS and UJCS. It can be used any place that CWT, JWT, UCCS or UJCS messages are used. It may also be sent as a submodule.

A DEB has two main parts.

The first part is a full top-level token. This top-level token must have at least one submodule that is a detached digest. This top-level token may be either CBOR or JSON-encoded. It may be a CWT, JWT, UCCS or UJCS, but not a DEB. The same mechanism for distinguishing the type for nested token submodules is used here.

The second part is a map/object containing the detached Claims-Sets corresponding to the detached digests in the full token. When the DEB is CBOR-encoded, each Claims-Set is wrapped in a byte string. When the DEB is JSON-encoded, each Claims-Set is base64url encoded. All the detached Claims-Sets MUST be encoded in the same format as the DEB. No mixing of encoding formats is allowed for the Claims-Sets in a DEB.

For CBOR-encoded DEBs, tag TBD602 can be used to identify it. The normal rules apply for use or non-use of a tag. When it is sent as a submodule, it is always sent as a tag to distinguish it from the other types of nested tokens.

The digests of the detached claims sets are associated with detached claims-sets by label/name. It is up to the constructor of the detached EAT bundle to ensure the names uniquely identify the detached claims sets. Since the names are used only in the detached EAT bundle, they can be very short, perhaps one byte.

```
Detached-EAT-Bundle = [
  main-token : Nested-Token,
  detached-claims-sets: {
    + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set
  }
]
```

```
json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"
```

```
cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

## 6. Endorsements and Verification Keys

The Verifier must possess the correct key when it performs the cryptographic part of an EAT verification (e.g., verifying the COSE/JOSE signature). This section describes several ways to identify the verification key. There is not one standard method.

The verification key itself may be a public key, a symmetric key or something complicated in the case of a scheme like Direct Anonymous Attestation (DAA).

RATS Architecture [RATS.Architecture] describes what is called an Endorsement. This is an input to the Verifier that is usually the basis of the trust placed in an EAT and the Attester that generated it. It may contain the public key for verification of the signature on the EAT. It may contain Reference Values to which EAT claims are compared as part of the verification process. It may contain implied claims, those that are passed on to the Relying Party in Attestation Results.

There is not yet any standard format(s) for an Endorsement. One format that may be used for an Endorsement is an X.509 certificate. Endorsement data like Reference Values and implied claims can be carried in X.509 v3 extensions. In this use, the public key in the X.509 certificate becomes the verification key, so identification of the Endorsement is also identification of the verification key.

The verification key identification and establishment of trust in the EAT and the attester may also be by some other means than an Endorsement.

For the components (Attester, Verifier, Relying Party,...) of a particular end-end attestation system to reliably interoperate, its definition should specify how the verification key is identified. Usually, this will be in the profile document for a particular attestation system.

## 6.1. Identification Methods

Following is a list of possible methods of key identification. A specific attestation system may employ any one of these or one not listed here.

The following assumes Endorsements are X.509 certificates or equivalent and thus does not mention or define any identifier for Endorsements in other formats. If such an Endorsement format is created, new identifiers for them will also need to be created.

### 6.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC8152] and [RFC7515]

COSE leaves the semantics of the key ID open-ended. It could be a record locator in a database, a hash of a public key, an input to a



KDF, an authority key identifier (AKI) for an X.509 certificate or other. The profile document should specify what the key ID's semantics are.

#### 6.1.2. JWS and COSE X.509 Header Parameters

COSE X.509 [COSE.X509.Draft] and JSON Web Signature [RFC7515] define several header parameters (x5t, x5u,...) for referencing or carrying X.509 certificates any of which may be used.

The X.509 certificate may be an Endorsement and thus carrying additional input to the Verifier. It may be just an X.509 certificate, not an Endorsement. The same header parameters are used in both cases. It is up to the attestation system design and the Verifier to determine which.

#### 6.1.3. CBOR Certificate COSE Header Parameters

Compressed X.509 and CBOR Native certificates are defined by CBOR Certificates [CBOR.Cert.Draft]. These are semantically compatible with X.509 and therefore can be used as an equivalent to X.509 as described above.

These are identified by their own header parameters (c5t, c5u,...).

#### 6.1.4. Claim-Based Key Identification

For some attestation systems, a claim may be re-used as a key identifier. For example, the UEID uniquely identifies the entity and therefore can work well as a key identifier or Endorsement identifier.

This has the advantage that key identification requires no additional bytes in the EAT and makes the EAT smaller.

This has the disadvantage that the unverified EAT must be substantially decoded to obtain the identifier since the identifier is in the COSE/JOSE payload, not in the headers.

#### 6.2. Other Considerations

In all cases there must be some way that the verification key is itself verified or determined to be trustworthy. The key identification itself is never enough. This will always be by some out-of-band mechanism that is not described here. For example, the Verifier may be configured with a root certificate or a master key by the Verifier system administrator.

Often an X.509 certificate or an Endorsement carries more than just the verification key. For example, an X.509 certificate might have key usage constraints and an Endorsement might have Reference Values. When this is the case, the key identifier must be either a protected header or in the payload such that it is cryptographically bound to the EAT. This is in line with the requirements in section 6 on Key Identification in JSON Web Signature [RFC7515].

## 7. Profiles

This EAT specification does not guarantee that implementations of it will interoperate. The variability in this specification is necessary to accommodate the widely varying use cases. An EAT profile narrows the specification for a specific use case. An ideal EAT profile will guarantee interoperability.

The profile can be named in the token using the profile claim described in Section 3.20.

A profile can apply to Attestation Evidence or to Attestation Results or both.

### 7.1. Format of a Profile Document

A profile document doesn't have to be in any particular format. It may be simple text, something more formal or a combination.

In some cases CDDL may be created that replaces CDDL in this or other document to express some profile requirements. For example, to require the altitude data item in the location claim, CDDL can be written that replicates the location claim with the altitude no longer optional.

### 7.2. List of Profile Issues

The following is a list of EAT, CWT, UCCS, JWS, UJCS, COSE, JOSE and CBOR options that a profile should address.

#### 7.2.1. Use of JSON, CBOR or both

The profile should indicate whether the token format should be CBOR, JSON, both or even some other encoding. If some other encoding, a specification for how the CDDL described here is serialized in that encoding is necessary.

This should be addressed for the top-level token and for any nested tokens. For example, a profile might require all nested tokens to be of the same encoding of the top level token.

#### 7.2.2. CBOR Map and Array Encoding

The profile should indicate whether definite-length arrays/maps, indefinite-length arrays/maps or both are allowed. A good default is to allow only definite-length arrays/maps.

An alternate is to allow both definite and indefinite-length arrays/maps. The decoder should accept either. Encoders that need to fit on very small hardware or be actually implement in hardware can use indefinite-length encoding.

This applies to individual EAT claims, CWT and COSE parts of the implementation.

#### 7.2.3. CBOR String Encoding

The profile should indicate whether definite-length strings, indefinite-length strings or both are allowed. A good default is to allow only definite-length strings. As with map and array encoding, allowing indefinite-length strings can be beneficial for some smaller implementations.

#### 7.2.4. CBOR Preferred Serialization

The profile should indicate whether encoders must use preferred serialization. The profile should indicate whether decoders must accept non-preferred serialization.

#### 7.2.5. COSE/JOSE Protection

COSE and JOSE have several options for signed, MACed and encrypted messages. EAT/CWT has the option to have no protection using UCCS and JOSE has a NULL protection option. It is possible to implement no protection, sign only, MAC only, sign then encrypt and so on. All combinations allowed by COSE, JOSE, JWT, CWT, UCCS and UJCS are allowed by EAT.

The profile should list the protections that must be supported by all decoders implementing the profile. The encoders then must implement a subset of what is listed for the decoders, perhaps only one.

Implementations may choose to sign or MAC before encryption so that the implementation layer doing the signing or MACing can be the smallest. It is often easier to make smaller implementations more secure, perhaps even implementing in solely in hardware. The key material for a signature or MAC is a private key, while for encryption it is likely to be a public key. The key for encryption requires less protection.

#### 7.2.6. COSE/JOSE Algorithms

The profile document should list the COSE algorithms that a Verifier must implement. The Attester will select one of them. Since there is no negotiation, the Verifier should implement all algorithms listed in the profile. If detached submodules are used, the COSE algorithms allowed for their digests should also be in the profile.

#### 7.2.7. DEB Support

A Detached EAT Bundle Section 5 is a special case message that will not often be used. A profile may prohibit its use.

#### 7.2.8. Verification Key Identification

Section Section 6 describes a number of methods for identifying a verification key. The profile document should specify one of these or one that is not described. The ones described in this document are only roughly described. The profile document should go into the full detail.

#### 7.2.9. Endorsement Identification

Similar to, or perhaps the same as Verification Key Identification, the profile may wish to specify how Endorsements are to be identified. However note that Endorsement Identification is optional, where as key identification is not.

#### 7.2.10. Freshness

Just about every use case will require some means of knowing the EAT is recent enough and not a replay of an old token. The profile should describe how freshness is achieved. The section on Freshness in [RATS.Architecture] describes some of the possible solutions to achieve this.

#### 7.2.11. Required Claims

The profile can list claims whose absence results in Verification failure.

#### 7.2.12. Prohibited Claims

The profile can list claims whose presence results in Verification failure.

#### 7.2.13. Additional Claims

The profile may describe entirely new claims. These claims can be required or optional.

#### 7.2.14. Refined Claim Definition

The profile may lock down optional aspects of individual claims. For example, it may require altitude in the location claim, or it may require that HW Versions always be described using EAN-13.

#### 7.2.15. CBOR Tags

The profile should specify whether the token should be a CWT Tag or not. Similarly, the profile should specify whether the token should be a UCCS tag or not.

When COSE protection is used, the profile should specify whether COSE tags are used or not. Note that RFC 8392 requires COSE tags be used in a CWT tag.

Often a tag is unnecessary because the surrounding or carrying protocol identifies the object as an EAT.

#### 7.2.16. Manifests and Software Evidence Claims

The profile should specify which formats are allowed for the manifests and software evidence claims. The profile may also go on to say which parts and options of these formats are used, allowed and prohibited.

### 8. Encoding and Collected CDDL

An EAT is fundamentally defined using CDDL. This document specifies how to encode the CDDL in CBOR or JSON. Since CBOR can express some things that JSON can't (e.g., tags) or that are expressed differently (e.g., labels) there is some CDDL that is specific to the encoding format.

#### 8.1. Claims-Set and CDDL for CWT and JWT

CDDL was not used to define CWT or JWT. It was not available at the time.

This document defines CDDL for both CWT and JWT as well as UCCS. This document does not change the encoding or semantics of anything in a CWT or JWT.

A Claims-Set is the central data structure for EAT, CWT, JWT and UCCS. It holds all the claims and is the structure that is secured by signing or other means. It is not possible to define EAT, CWT, JWT or UCCS in CDDL without it. The CDDL definition of Claims-Set here is applicable to EAT, CWT, JWT and UCCS.

This document specifies how to encode a Claims-Set in CBOR or JSON.

With the exception of nested tokens and some other externally defined structures (e.g., SWIDs) an entire Claims-Set must be encoded in either CBOR or JSON, never a mixture.

CDDL for the seven claims defined by [RFC8392] and [RFC7519] is included here.

## 8.2. Encoding Data Types

This makes use of the types defined in [RFC8610] Appendix D, Standard Prelude.

### 8.2.1. Common Data Types

time-int is identical to the epoch-based time, but disallows floating-point representation.

Unless explicitly indicated, URIs are not the URI tag defined in [RFC8949]. They are just text strings that contain a URI.

string-or-uri = tstr

time-int = #6.1(int)

### 8.2.2. JSON Interoperability

JSON should be encoded per [RFC8610] Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

- o bstr - must be base64url encoded
- o time - must be encoded as NumericDate as described section 2 of [RFC7519].
- o string-or-uri - must be encoded as StringOrURI as described section 2 of [RFC7519].
- o uri - must be a URI [RFC3986].

- o oid - encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1").

### 8.2.3. Labels

Map labels, including Claims-Keys and Claim-Names, and enumerated-type values are always integers when encoding in CBOR and strings when encoding in JSON. There is an exception to this for naming submodules and detached claims sets in a DEB. These are strings in CBOR.

The CDDL in most cases gives both the integer label and the string label as it is not convenient to have conditional CDDL for such.

### 8.3. CBOR Interoperability

CBOR allows data items to be serialized in more than one form. If the sender uses a form that the receiver can't decode, there will not be interoperability.

This specification gives no blanket requirements to narrow CBOR serialization for all uses of EAT. This allows individual uses to tailor serialization to the environment. It also may result in EAT implementations that don't interoperate.

One way to guarantee interoperability is to clearly specify CBOR serialization in a profile document. See Section 7 for a list of serialization issues that should be addressed.

EAT will be commonly used where the entity generating the attestation is constrained and the receiver/Verifier of the attestation is a capacious server. Following is a set of serialization requirements that work well for that use case and are guaranteed to interoperate. Use of this serialization is recommended where possible, but not required. An EAT profile may just reference the following section rather than spell out serialization details.

#### 8.3.1. EAT Constrained Device Serialization

- o Preferred serialization described in section 4.1 of [RFC8949] is not required. The EAT decoder must accept all forms of number serialization. The EAT encoder may use any form it wishes.
- o The EAT decoder must accept indefinite length arrays and maps as described in section 3.2.2 of [RFC8949]. The EAT encoder may use indefinite length arrays and maps if it wishes.

- o The EAT decoder must accept indefinite length strings as described in section 3.2.3 of [RFC8949]. The EAT encoder may use indefinite length strings if it wishes.
- o Sorting of maps by key is not required. The EAT decoder must not rely on sorting.
- o Deterministic encoding described in Section 4.2 of [RFC8949] is not required.
- o Basic validity described in section 5.3.1 of [RFC8949] must be followed. The EAT encoder must not send duplicate map keys/labels or invalid UTF-8 strings.

#### 8.4. Collected Common CDDL

```
Claims-Set = {  
  * $$claims-set-claims,  
  * Claim-Label .feature "extended-label" => any  
}  
  
Claim-Label = int / text  
string-or-uri = tstr  
  
time-int = #6.1(int)  
$$claims-set-claims /= (iss-label => text)  
$$claims-set-claims /= (sub-label => text)  
$$claims-set-claims /= (aud-label => text)  
$$claims-set-claims /= (exp-label => ~time)  
$$claims-set-claims /= (nbf-label => ~time)  
$$claims-set-claims /= (iat-label => ~time)  
  
$$claims-set-claims /=  
  (nonce-label => nonce-type / [ 2* nonce-type ])  
  
nonce-type = bstr .size (8..64)  
$$claims-set-claims /= (ueid-label => ueid-type)  
  
ueid-type = bstr .size (7..33)  
$$claims-set-claims /= (sueids-label => sueids-type)  
  
sueids-type = {  
  + tstr => ueid-type  
}  
oemid-pen = int  
  
oemid-ieee = bstr .size 3
```



```
oemid-random = bstr .size 16

$$claims-set-claims //= (
    oemid-label =>
        oemid-random / oemid-ieee / oemid-pen
)
$$claims-set-claims //= (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version: tstr,
    scheme: $version-scheme
]
hardware-model-type = bytes .size (1..32)

$$claims-set-claims //= (
    hardware-model-label => hardware-model-type
)
$$claims-set-claims //= ( sw-name-label => tstr )
$$claims-set-claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version: tstr,
    scheme: $version-scheme ; As defined by CoSWID
]
$$claims-set-claims //= (
    security-level-label =>
        security-level-cbor-type /
        security-level-json-type
)

security-level-cbor-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)

security-level-json-type =
    "unrestricted" /
    "restricted" /
    "secure-restricted" /
    "hardware"
$$claims-set-claims //= (secure-boot-label => bool)
$$claims-set-claims //= (
    debug-status-label =>
```

```

        debug-status-cbor-type / debug-status-json-type
    )

    debug-status-cbor-type = &(
        enabled: 0,
        disabled: 1,
        disabled-since-boot: 2,
        disabled-permanently: 3,
        disabled-fully-and-permanently: 4
    )

    debug-status-json-type =
        "enabled" /
        "disabled" /
        "disabled-since-boot" /
        "disabled-permanently" /
        "disabled-fully-and-permanently"
    $$claims-set-claims // = (location-label => location-type)

    location-type = {
        latitude => number,
        longitude => number,
        ? altitude => number,
        ? accuracy => number,
        ? altitude-accuracy => number,
        ? heading => number,
        ? speed => number,
        ? timestamp => ~time-int,
        ? age => uint
    }

    latitude = 1 / "latitude"
    longitude = 2 / "longitude"
    altitude = 3 / "altitude"
    accuracy = 4 / "accuracy"
    altitude-accuracy = 5 / "altitude-accuracy"
    heading = 6 / "heading"
    speed = 7 / "speed"
    timestamp = 8 / "timestamp"
    age = 9 / "age"
    $$claims-set-claims // = (uptime-label => uint)
    $$claims-set-claims // = (boot-seed-label => bytes)
    $$claims-set-claims // = (odometer-label => uint)
    $$claims-set-claims // = (
        intended-use-label =>
            intended-use-cbor-type / intended-use-json-type
    )

```

```
intended-use-cbor-type = &(
    generic: 1,
    registration: 2,
    provisioning: 3,
    csr: 4,
    pop: 5
)

intended-use-json-type =
    "generic" /
    "registration" /
    "provisioning" /
    "csr" /
    "pop"
$$claims-set-claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: ~uri
    dloa_platform_label: text
    ? dloa_application_label: text
]
$$claims-set-claims //= (profile-label => ~uri / ~oid)
$$claims-set-claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ $$manifest-formats]

coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>

$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx$$claims-set-claims
// = (
    swevidence-label => swevidence-type
)

swevidence-type = [+ $$swevidence-formats]

coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag
$$claims-set-claims //= (swresults-label => [ + swresult-type ])

verification-result-cbor-type = &(
    verification-not-run: 1,
    verification-indeterminate: 2,
    verification-failed: 3,
    fully-verified: 4,
    partially-verified: 5,
```

```
)

verification-result-json-type =
    "verification-not-run" /
    "verification-indeterminate" /
    "verification-failed" /
    "fully-verified" /
    "partially-verified"

verification-objective-cbor-type = &(
    all: 1,
    firmware: 2,
    kernel: 3,
    privileged: 4,
    system-libs: 5,
    partial: 6,
)

verification-objective-json-type =
    "all" /
    "firmware" /
    "kernel" /
    "privileged" /
    "system-libs" /
    "partial"

swresult-type = [
    verification-system: tstr,
    objective: verification-objective-cbor-type /
        verification-objective-json-type,
    result: verification-result-cbor-type /
        verification-result-json-type,
    ? objective-name: tstr
]
$$claims-set-claims // = (submods-label => { + text => Submodule })

Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest

Detached-Submodule-Digest = [
    algorithm : int / text,
    digest : bstr
]
Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set
    }
]
```

```
    }  
  ]
```

```
json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"
```

```
cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

#### 8.5. Collected CDDL for CBOR

```
CBOR-Token = Tagged-CBOR-Token / Untagged-CBOR-Token
```

```
Tagged-CBOR-Token  = CWT-Tagged-Message  
Tagged-CBOR-Token /= UCCS-Tagged-Message  
Tagged-CBOR-Token /= DEB-Tagged-Message
```

```
Untagged-CBOR-Token  = CWT-Untagged-Message  
Untagged-CBOR-Token /= UCCS-Untagged-Message  
Untagged-CBOR-Token /= DEB-Untagged-Message
```

```
CWT-Tagged-Message = COSE_Tagged_Message  
CWT-Untagged-Message = COSE_Untagged_Message
```

```
UCCS-Message = UCCS-Tagged-Message / UCCS-Untagged-Message
```

```
UCCS-Tagged-Message = #6.601(UCCS-Untagged-Message)
```

```
UCCS-Untagged-Message = Claims-Set
```

```
DEB-Tagged-Message = #6.602(DEB-Untagged-Message)
```

```
DEB-Untagged-Message = Detached-EAT-Bundle
```

```
Nested-Token =  
  tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)  
  bstr .cbor Tagged-CBOR-Token
```

```
iss-label = 1  
sub-label = 2  
aud-label = 3
```

```

exp-label = 4
nbf-label = 5
iat-label = 6
cti-label = 7nonce-label = 10
ueid-label = 256
sueids-label = 257
oemid-label = 258
hardware-model-label = 259
hardware-version-label = 260
secure-boot-label = 262
debug-status-label = 263
location-label = 264
profile-label = 265
submods-label = 266
security-level-label = <TBD>
uptime-label = <TBD>
boot-seed-label = <TB>
odometer-label = <TBD>
intended-use-label = <TBD>
dloas-label = <TBD>
sw-name-label = <TBD>
sw-version-label = <TBD>
manifests-label = <TBD>
swevidence-label = <TBD>
swresults-label = <TBD>

```

#### 8.6. Collected CDDL for JSON

```
JWT-Message = text .regexp [A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]+
```

```
UJCS-Message = Claims-Set
```

```

Nested-Token = [
  type : "JWT" / "CBOR" / "UJCS" / "DEB",
  nested-token : JWT-Message /
                  B64URL-Tagged-CBOR-Token /
                  DEB-JSON-Message /
                  UJCS-Message
]

```

```

B64URL-Tagged-CBOR-Token = tstr .regexp "[A-Za-z0-9_=-]+"
iss-label = "iss"
sub-label = "sub"
aud-label = "aud"

```

```
exp-label = "exp"
nbf-label = "nbf"
iat-label = "iat"
cti-label = "cti"nonce-label /= "nonce"
```

```
ueid-label /= "ueid"
sueids-label /= "sueids"
oemid-label /= "oemid"
hardware-model-label /= "hwmodel"
hardware-version-label /= "hwversion"
security-level-label /= "seclevel"
secure-boot-label /= "secboot"
debug-status-label /= "dbgstat"
location-label /= "location"
profile-label /= "eat-profile"
uptime-label /= "uptime"
boot-seed-label /= "bootseed"
odometer-label /= "odometer"
intended-use-label /= "intuse"
dloas-label /= "dloas"
sw-name-label /= "swname"
sw-version-label /= "swversion"
manifests-label /= "manifests"
swevidence-label /= "swevidence"
swresults-label /= "swresults"
submods-label /= "submods"
```

```
latitude /= "lat"
longitude /= "long"
altitude /= "alt"
accuracy /= "accry"
altitude-accuracy /= "alt-accry"
heading /= "heading"
speed /= "speed"
```

## 9. IANA Considerations

### 9.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries

Claims defined for EAT are compatible with those of CWT and JWT so the CWT and JWT Claims Registries, [IANA.CWT.Claims] and [IANA.JWT.Claims], are re used. No new IANA registry is created.

All EAT claims defined in this document are placed in both registries. All new EAT claims defined subsequently should be placed in both registries.

## 9.2. Claim Characteristics

The following is design guidance for creating new EAT claims, particularly those to be registered with IANA.

Much of this guidance is generic and could also be considered when designing new CWT or JWT claims.

### 9.2.1. Interoperability and Relying Party Orientation

It is a broad goal that EATs can be processed by Relying Parties in a general way regardless of the type, manufacturer or technology of the device from which they originate. It is a goal that there be general-purpose verification implementations that can verify tokens for large numbers of use cases with special cases and configurations for different device types. This is a goal of interoperability of the semantics of claims themselves, not just of the signing, encoding and serialization formats.

This is a lofty goal and difficult to achieve broadly requiring careful definition of claims in a technology neutral way. Sometimes it will be difficult to design a claim that can represent the semantics of data from very different device types. However, the goal remains even when difficult.

### 9.2.2. Operating System and Technology Neutral

Claims should be defined such that they are not specific to an operating system. They should be applicable to multiple large high-level operating systems from different vendors. They should also be applicable to multiple small embedded operating systems from multiple vendors and everything in between.

Claims should not be defined such that they are specific to a SW environment or programming language.

Claims should not be defined such that they are specific to a chip or particular hardware. For example, they should not just be the contents of some HW status register as it is unlikely that the same HW status register with the same bits exists on a chip of a different manufacturer.

The boot and debug state claims in this document are an example of a claim that has been defined in this neutral way.



### 9.2.3. Security Level Neutral

Many use cases will have EATs generated by some of the most secure hardware and software that exists. Secure Elements and smart cards are examples of this. However, EAT is intended for use in low-security use cases the same as high-security use case. For example, an app on a mobile device may generate EATs on its own.

Claims should be defined and registered on the basis of whether they are useful and interoperable, not based on security level. In particular, there should be no exclusion of claims because they are just used only in low-security environments.

### 9.2.4. Reuse of Extant Data Formats

Where possible, claims should use already standardized data items, identifiers and formats. This takes advantage of the expertise put into creating those formats and improves interoperability.

Often extant claims will not be defined in an encoding or serialization format used by EAT. It is preferred to define a CBOR and JSON format for them so that EAT implementations do not require a plethora of encoders and decoders for serialization formats.

In some cases, it may be better to use the encoding and serialization as is. For example, signed X.509 certificates and CRLs can be carried as-is in a byte string. This retains interoperability with the extensive infrastructure for creating and processing X.509 certificates and CRLs.

### 9.2.5. Proprietary Claims

EAT allows the definition and use of proprietary claims.

For example, a device manufacturer may generate a token with proprietary claims intended only for verification by a service offered by that device manufacturer. This is a supported use case.

In many cases proprietary claims will be the easiest and most obvious way to proceed, however for better interoperability, use of general standardized claims is preferred.

## 9.3. Claims Registered by This Document

This specification adds the following values to the "JSON Web Token Claims" registry established by [RFC7519] and the "CBOR Web Token Claims Registry" established by [RFC8392]. Each entry below is an

addition to both registries (except for the nonce claim which is already registered for JWT, but not registered for CWT).

The "Claim Description", "Change Controller" and "Specification Documents" are common and equivalent for the JWT and CWT registries. The "Claim Key" and "Claim Value Types(s)" are for the CWT registry only. The "Claim Name" is as defined for the CWT registry, not the JWT registry. The "JWT Claim Name" is equivalent to the "Claim Name" in the JWT registry.

#### 9.3.1. Claims for Early Assignment

RFC Editor: in the final publication this section should be combined with the following section as it will no longer be necessary to distinguish claims with early assignment. Also, the following paragraph should be removed.

The claims in this section have been (requested for / given) early assignment according to [RFC7120]. They have been assigned values and registered before final publication of this document. While their semantics is not expected to change in final publication, it is possible that they will. The JWT Claim Names and CWT Claim Keys are not expected to change.

In draft -06 an early allocation was described. The processing of that early allocation was never correctly completed. This early allocation assigns different numbers for the CBOR claim labels. This early allocation will presumably complete correctly

- o Claim Name: Nonce
- o Claim Description: Nonce
- o JWT Claim Name: "nonce" (already registered for JWT)
- o Claim Key: TBD (requested value 10)
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): [OpenIDConnectCore], \*this document\*
- o Claim Name: UEID
- o Claim Description: The Universal Entity ID
- o JWT Claim Name: "ueid"

- o CWT Claim Key: TBD (requested value 256)
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: SUEIDs
- o Claim Description: Semi-permanent UEIDs
- o JWT Claim Name: "sueids"
- o CWT Claim Key: TBD (requested value 257)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Hardware OEMID
- o Claim Description: Hardware OEM ID
- o JWT Claim Name: "oemid"
- o Claim Key: TBD (requested value 258)
- o Claim Value Type(s): byte string or integer
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Hardware Model
- o Claim Description: Model identifier for hardware
- o JWT Claim Name: "hwmodel"
- o Claim Key: TBD (requested value 259)
- o Claim Value Type(s): byte string
- o Change Controller: IESG

- o Specification Document(s): \*this document\*
- o Claim Name: Hardware Version
- o Claim Description: Hardware Version Identifier
- o JWT Claim Name: "hwversion"
- o Claim Key: TBD (requested value 260)
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Secure Boot
- o Claim Description: Indicate whether the boot was secure
- o JWT Claim Name: "secboot"
- o Claim Key: TBD (requested value 262)
- o Claim Value Type(s): Boolean
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Debug Status
- o Claim Description: Indicate status of debug facilities
- o JWT Claim Name: "dbgstat"
- o Claim Key: TBD (requested value 263)
- o Claim Value Type(s): integer or string
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Location
- o Claim Description: The geographic location

- o JWT Claim Name: "location"
- o Claim Key: TBD (requested value 264)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Profile
- o Claim Description: Indicates the EAT profile followed
- o JWT Claim Name: "eat\_profile"
- o Claim Key: TBD (requested value 265)
- o Claim Value Type(s): URI or OID
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Submodules Section
- o Claim Description: The section containing submodules
- o JWT Claim Name: "submods"
- o Claim Key: TBD (requested value 266)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): \*this document\*

#### 9.3.2. To be Assigned Claims

(Early assignment is NOT requested for these claims. Implementers should be aware they may change)

- o Claim Name: Security Level
- o Claim Description: Characterization of the security of an Attester or submodule

- o JWT Claim Name: "secclevel"
- o Claim Key: TBD
- o Claim Value Type(s): integer or string
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Uptime
- o Claim Description: Uptime
- o JWT Claim Name: "uptime"
- o Claim Key: TBD
- o Claim Value Type(s): unsigned integer
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Boot Seed
- o Claim Description: Identifies a boot cycle
- o JWT Claim Name: "bootseed"
- o Claim Key: TBD
- o Claim Value Type(s): bytes
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: Intended Use
- o Claim Description: Indicates intended use of the EAT
- o JWT Claim Name: "intuse"
- o Claim Key: TBD
- o Claim Value Type(s): integer or string

- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: DLOAs
- o Claim Description: Certifications received as Digital Letters of Approval
- o JWT Claim Name: "dloas"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: SW Name
- o Claim Description: The name of the SW running in the entity
- o JWT Claim Name: "swname"
- o Claim Key: TBD
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: SW Version
- o Claim Description: The version of SW running in the entity
- o JWT Claim Name: "swversion"
- o Claim Key: TBD
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: SW Manifests

- o Claim Description: Manifests describing the SW installed on the entity
- o JWT Claim Name: "manifests"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: SW Evidence
- o Claim Description: Measurements of the SW, memory configuration and such on the entity
- o JWT Claim Name: "swevidence"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): \*this document\*
- o Claim Name: SW Measurment Results
- o Claim Description: The results of comparing SW measurements to reference values
- o JWT Claim Name: "swresults"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): \*this document\*

#### 9.3.3. Version Schemes Registered by this Document

IANA is requested to register a new value in the "Software Tag Version Scheme Values" established by [CoSWID].



The new value is a version scheme a 13-digit European Article Number [EAN-13]. An EAN-13 is also known as an International Article Number or most commonly as a bar code. This version scheme is the ASCII text representation of EAN-13 digits, the same ones often printed with a bar code. This version scheme must comply with the EAN allocation and assignment rules. For example, this requires the manufacturer to obtain a manufacture code from GS1.

Index	Version Scheme Name	Specification
5	ean-13	This document

#### 9.3.4. UEID URN Registered by this Document

IANA is requested to register the following new subtypes in the "DEV URN Subtypes" registry under "Device Identification". See [RFC9039].

Subtype	Description	Reference
ueid	Universal Entity Identifier	This document
sueid	Semi-permanent Universal Entity Identifier	This document

#### 9.3.5. Tag for Detached EAT Bundle

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

Tag	Data Items	Semantics
TBD602	array	Detached EAT Bundle Section 5

### 10. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity and therefore, implementations should consider providing privacy-preserving options dependent on the intended usage of the EAT. Examples would include suppression of location claims for EAT's provided to unauthenticated consumers.

### 10.1. UEID and SUEID Privacy Considerations

A UEID is usually not privacy-preserving. Any set of Relying Parties that receives tokens that happen to be from a particular entity will be able to know the tokens are all from the same entity and be able to track it.

Thus, in many usage situations UEID violates governmental privacy regulation. In other usage situations a UEID will not be allowed for certain products like browsers that give privacy for the end user. It will often be the case that tokens will not have a UEID for these reasons.

An SUEID is also usually not privacy-preserving. In some cases it may have fewer privacy issues than a UEID depending on when and how and when it is generated.

There are several strategies that can be used to still be able to put UEIDs and SUEIDs in tokens:

- o The entity obtains explicit permission from the user of the entity to use the UEID/SUEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID/SUEID.
- o The UEID/SUEID is used only in a particular context or particular use case. It is used only by one Relying Party.
- o The entity authenticates the Relying Party and generates a derived UEID/SUEID just for that particular Relying Party. For example, the Relying Party could prove their identity cryptographically to the entity, then the entity generates a UEID just for that Relying Party by hashing a proofed Relying Party ID with the main entity UEID/SUEID.

Note that some of these privacy preservation strategies result in multiple UEIDs and SUEIDs per entity. Each UEID/SUEID is used in a different context, use case or system on the entity. However, from the view of the Relying Party, there is just one UEID and it is still globally universal across manufacturers.

### 10.2. Location Privacy Considerations

Geographic location is most always considered personally identifiable information. Implementers should consider laws and regulations governing the transmission of location data from end user devices to servers and services. Implementers should consider using location

management facilities offered by the operating system on the entity generating the attestation. For example, many mobile phones prompt the user for permission when before sending location data.

### 10.3. Replay Protection and Privacy

EAT offers 2 primary mechanisms for token replay protection (also sometimes known as token "freshness"): the cti/jti claim and the nonce claim. The cti/jti claim in a CWT/JWT is a field that may be optionally included in the EAT and is in general derived on the same device in which the entity is instantiated. The nonce claim is based on a value that is usually derived remotely (outside of the entity). These claims can be used to extract and convey personally-identifying information either inadvertently or by intention. For instance, an implementor may choose a cti that is equivalent to a username associated with the device (e.g., account login). If the token is inspected by a 3rd-party then this information could be used to identify the source of the token or an account associated with the token (e.g., if the account name is used to derive the nonce). In order to avoid the conveyance of privacy-related information in either the cti/jti or nonce claims, these fields should be derived using a salt that originates from a true and reliable random number generator or any other source of randomness that would still meet the target system requirements for replay protection.

## 11. Security Considerations

The security considerations provided in Section 8 of [RFC8392] and Section 11 of [RFC7519] apply to EAT in its CWT and JWT form, respectively. In addition, implementors should consider the following.

### 11.1. Key Provisioning

Private key material can be used to sign and/or encrypt the EAT, or can be used to derive the keys used for signing and/or encryption. In some instances, the manufacturer of the entity may create the key material separately and provision the key material in the entity itself. The manufacturer of any entity that is capable of producing an EAT should take care to ensure that any private key material be suitably protected prior to provisioning the key material in the entity itself. This can require creation of key material in an enclave (see [RFC4949] for definition of "enclave"), secure transmission of the key material from the enclave to the entity using an appropriate protocol, and persistence of the private key material in some form of secure storage to which (preferably) only the entity has access.

#### 11.1.1. Transmission of Key Material

Regarding transmission of key material from the enclave to the entity, the key material may pass through one or more intermediaries. Therefore some form of protection ("key wrapping") may be necessary. The transmission itself may be performed electronically, but can also be done by human courier. In the latter case, there should be minimal to no exposure of the key material to the human (e.g. encrypted portable memory). Moreover, the human should transport the key material directly from the secure enclave where it was created to a destination secure enclave where it can be provisioned.

#### 11.2. Transport Security

As stated in Section 8 of [RFC8392], "The security of the CWT relies upon on the protections offered by COSE". Similar considerations apply to EAT when sent as a CWT. However, EAT introduces the concept of a nonce to protect against replay. Since an EAT may be created by an entity that may not support the same type of transport security as the consumer of the EAT, intermediaries may be required to bridge communications between the entity and consumer. As a result, it is RECOMMENDED that both the consumer create a nonce, and the entity leverage the nonce along with COSE mechanisms for encryption and/or signing to create the EAT.

Similar considerations apply to the use of EAT as a JWT. Although the security of a JWT leverages the JSON Web Encryption (JWE) and JSON Web Signature (JWS) specifications, it is still recommended to make use of the EAT nonce.

#### 11.3. Multiple EAT Consumers

In many cases, more than one EAT consumer may be required to fully verify the entity attestation. Examples include individual consumers for nested EATs, or consumers for individual claims with an EAT. When multiple consumers are required for verification of an EAT, it is important to minimize information exposure to each consumer. In addition, the communication between multiple consumers should be secure.

For instance, consider the example of an encrypted and signed EAT with multiple claims. A consumer may receive the EAT (denoted as the "receiving consumer"), decrypt its payload, verify its signature, but then pass specific subsets of claims to other consumers for evaluation ("downstream consumers"). Since any COSE encryption will be removed by the receiving consumer, the communication of claim subsets to any downstream consumer should leverage a secure protocol (e.g. one that uses transport-layer security, i.e. TLS),

However, assume the EAT of the previous example is hierarchical and each claim subset for a downstream consumer is created in the form of a nested EAT. Then transport security between the receiving and downstream consumers is not strictly required. Nevertheless, downstream consumers of a nested EAT should provide a nonce unique to the EAT they are consuming.

## 12. References

### 12.1. Normative References

- [CoSWID] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-20 (work in progress), January 2022.
- [DLOA] "Digital Letter of Approval", November 2015, <[https://globalplatform.org/wp-content/uploads/2015/12/GPC\\_DigitalLetterOfApproval\\_v1.0.pdf](https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf)>.
- [EAN-13] GS1, "International Article Number - EAN/UPC barcodes", 2019, <<https://www.gs1.org/standards/barcodes/ean-upc>>.
- [FIDO.AROE] The FIDO Alliance, "FIDO Authenticator Allowed Restricted Operating Environments List", November 2020, <<https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-allowed-restricted-operating-environments-list-v1.2-fd-20201102.html>>.
- [IANA.cbor-tags] "IANA CBOR Tags Registry", n.d., <<https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>>.
- [IANA.CWT.Claims] IANA, "CBOR Web Token (CWT) Claims", <<http://www.iana.org/assignments/cwt>>.
- [IANA.JWT.Claims] IANA, "JSON Web Token (JWT) Claims", <<https://www.iana.org/assignments/jwt>>.
- [OpenIDConnectCore] Sakimura, N., Bradley, J., Jones, M., Medeiros, B. D., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <[https://openid.net/specs/openid-connect-core-1\\_0.html](https://openid.net/specs/openid-connect-core-1_0.html)>.

- [PEN] "Private Enterprise Number (PEN) Request", n.d.,  
<<https://pen.iana.org/pen/PenApplication.page>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/info/rfc7800>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.
- [RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", RFC 9090, DOI 10.17487/RFC9090, July 2021, <<https://www.rfc-editor.org/info/rfc9090>>.
- [ThreeGPP.IMEI]  
3GPP, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification", 2019, <<https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729>>.
- [UCCS.Draft]  
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", draft-ietf-rats-uccs-02 (work in progress), January 2022.
- [WGS84] National Geospatial-Intelligence Agency (NGA), "WORLD GEODETIC SYSTEM 1984, NGA.STND.0036\_1.0.0\_WGS84", July 2014, <<https://earth-info.nga.mil/php/download.php?file=coord-wgs84>>.

## 12.2. Informative References

- [BirthdayAttack]  
"Birthday attack",  
<[https://en.wikipedia.org/wiki/Birthday\\_attack](https://en.wikipedia.org/wiki/Birthday_attack)>.
- [CBOR.Cert.Draft]  
Mattsson, J. P., Selander, G., Raza, S., Hoeglund, J., and  
M. Furuheid, "CBOR Encoded X.509 Certificates (C509  
Certificates)", draft-ietf-cose-chor-encoded-cert-03 (work  
in progress), January 2022.
- [Common.Criteria]  
"Common Criteria for Information Technology Security  
Evaluation", April 2017,  
<<https://www.commoncriteriaportal.org/cc/>>.
- [COSE.X509.Draft]  
Schaad, J., "CBOR Object Signing and Encryption (COSE):  
Header parameters for carrying and referencing X.509  
certificates", draft-ietf-cose-x509-08 (work in progress),  
December 2020.
- [FIPS-140]  
National Institute of Standards, "Security Requirements for  
Cryptographic Modules", May 2001,  
<<https://csrc.nist.gov/publications/detail/fips/140/2/final>>.
- [IEEE.802-2001]  
"IEEE Standard For Local And Metropolitan Area Networks  
Overview And Architecture", 2007,  
<<https://webstore.ansi.org/standards/ieee/ieee8022001r2007>>.
- [IEEE.802.1AR]  
"IEEE Standard, "IEEE 802.1AR Secure Device Identifier",  
December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.
- [IEEE.RA]  
"IEEE Registration Authority",  
<<https://standards.ieee.org/products-services/regauth/index.html>>.



## [OUI.Guide]

"Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", August 2017,  
<<https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf>>.

## [OUI.Lookup]

"IEEE Registration Authority Assignments",  
<<https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries>>.

## [RATS.Architecture]

Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-15 (work in progress), February 2022.

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

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006,  
<<https://www.rfc-editor.org/info/rfc4422>>.

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

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <<https://www.rfc-editor.org/info/rfc7120>>.

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

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021,  
<<https://www.rfc-editor.org/info/rfc9039>>.

[W3C.GeoLoc]

Worldwide Web Consortium, "Geolocation API Specification  
2nd Edition", January 2018, <[https://www.w3.org/TR/  
geolocation-API/#coordinates\\_interface](https://www.w3.org/TR/geolocation-API/#coordinates_interface)>.

## Appendix A. Examples

These examples are either UCCS, shown as CBOR diagnostic, or UJCS messages. Full CWT and JWT examples with signing and encryption are not given.

All UCCS examples can be the payload of a CWT. To do so, they must be converted from the UCCS message to a Claims-Set, which is achieved by "removing" the tag.

UJCS messages can be directly used as the payload of a JWT.

WARNING: These examples use tag and label numbers not yet assigned by IANA.

### A.1. Simple TEE Attestation

This is a simple attestation of a TEE that includes a manifest that is a payload CoSWID to describe the TEE's software.

/ This is a UCCS EAT that describes a simple TEE. /

```
601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 261: 3, / secure-restricted /
  / secure-boot /    262: true,
  / debug-status /   263: 2, / disabled-since-boot /
  / manifests /      273: [
                                / This is byte-string wrapped /
                                / payload CoSWID. It gives the TEE /
                                / software name, the version and /
                                / the name of the file it is in. /
                                h' da53574944a60064336132340c01016b
                                41636d6520544545204f530d65332e31
                                2e340282a2181f6b41636d6520544545
                                204f53182101a2181f6b41636d652054
                                4545204f5318210206a111a118186e61
                                636d655f7465655f332e657865'
                                ]
  })
```

/ A payload CoSWID created by the SW vendor. All this really does /  
 / is name the TEE SW, its version and lists the one file that /  
 / makes up the TEE. /

```
1398229316({
  / Unique CoSWID ID /      0: "3a24",
  / tag-version /          12: 1,
  / software-name /        1: "Acme TEE OS",
  / software-version /     13: "3.1.4",
  / entity /               2: [
                                {
                                  / entity-name /      31: "Acme TEE OS",
                                  / role /              33: 1 / tag-creator /
                                },
                                {
                                  / entity-name /      31: "Acme TEE OS",
                                  / role /              33: 2 / software-creator /
                                }
                              ],
  / payload /              6: {
    / ...file /            17: {
      / ...fs-name /       24: "acme_tee_3.exe"
    }
  }
})
```

## A.2. Submodules for Board and Device

```

/ This example shows use of submodules to give information /
/ about the chip, board and overall device. /
/
/ The main attestation is associated with the chip with the /
/ CPU and running the main OS. It is what has the keys and /
/ produces the token. /
/
/ The board is made by a different vendor than the chip. /
/ Perhaps it is some generic IoT board. /
/
/ The device is some specific appliance that is made by a /
/ different vendor than either the chip or the board. /
/
/ Here the board and device submodules aren't the typical /
/ target environments as described by the RATS architecture /
/ document, but they are a valid use of submodules. /

{
  / nonce /          10: h'948f8860d13a463e8e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / HW OEM ID /      258: h'894823', / IEEE OUI format OEM ID /
  / HW Model ID /    259: h'549dcecc8b987c737b44e40f7c635ce8'
                        / Hash of chip model name /,
  / HW Version /     260: ["1.3.4", 1], / Multipartnumeric version /
  / SW Name /        271: "Acme OS",
  / SW Version /     272: ["3.5.5", 1],
  / secure-boot /    262: true,
  / debug-status /   263: 3, / permanent-disable /
  / timestamp (iat) / 6: 1526542894,
  / security-level / 261: 3, / secure restricted OS /
  / submods / 266: {
    / A submodule to hold some claims about the circuit board /
    "board" : {
      / HW OEM ID /    258: h'9bef8787ebal3e2c8f6e7cb4blf4619a',
      / HW Model ID / 259: h'ee80f5a66c1fb9742999a8fdab930893'
                        / Hash of board module name /,
      / HW Version /   260: ["2.0a", 2] / multipartnumeric+suffix /
    },

    / A submodule to hold claims about the overall device /
    "device" : {
      / HW OEM ID /    258: 61234, / PEN Format OEM ID /
      / HW Version /   260: ["4012345123456", 5] / EAN-13 format (barcode) /
    }
  }
}

```

## A.3. EAT Produced by Attestation Hardware Block

```

/ This is an example of a token produced by a HW block      /
/ purpose-built for attestation. Only the nonce claim changes /
/ from one attestation to the next as the rest either come   /
/ directly from the hardware or from one-time-programmable memory /
/ (e.g. a fuse). 47 bytes encoded in CBOR (8 byte nonce, 16 byte /
/ UEID). /

601({
  / nonce /          10: h'948f8860d13a463e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID /          258: 64242, / Private Enterprise Number /
  / security-level / 261: 4, / hardware level security /
  / secure-boot /    262: true,
  / debug-status /   263: 3, / disabled-permanently /
  / HW version /     260: [ "3.1", 1 ] / Type is multipartnumeric /
})

```

## A.4. Detached EAT Bundle

In this DEB main token is produced by a HW attestation block. The detached Claims-Set is produced by a TEE and is largely identical to the Simple TEE examples above. The TEE digests its Claims-Set and feeds that digest to the HW block.

In a better example the attestation produced by the HW block would be a CWT and thus signed and secured by the HW block. Since the signature covers the digest from the TEE that Claims-Set is also secured.

The DEB itself can be assembled by untrusted SW.

```

/ This is a detached EAT bundle (DEB) tag.  /

602([

  / First part is a full EAT token with claims like nonce and /
  / UEID. Most importantly, it includes a submodule that is a /
  / detached digest which is the hash of the "TEE" claims set /
  / in the next section.                                     /
  /                                                         /
  / This token here is in UCCS format (unsigned). In a more /
  / realistic example, it would be a signed CWT.           /
  h'd90259a80a48948f8860d13a463e190100500198
  f50a4ff6c05861c8860d13a638ea19010219faf2
  19010504190106f5190107031901048263332e31
  0119010aa163544545822f5820e5cf95fd24fab7
  1446742dd58d43dae178e55fe2b94291a9291082
  ffc2635a0b',
  {
    / A CBOR-encoded byte-string wrapped EAT claims-set. It /
    / contains claims suitable for a TEE                       /
    "TEE" : h'a50a48948f8860d13a463e19010503190106f519
           01070219011181585dda53574944a60064336132
           340c01016b41636d6520544545204f530d65332e
           312e340282a2181f6b41636d6520544545204f53
           182101a2181f6b41636d6520544545204f531821
           0206a111a118186e61636d655f7465655f332e65
           7865'
  }
])

```

```

/ This example contains submodule that is a detached digest, /
/ which is the hash of a Claims-Set convey outside this token. /
/ Other than that is is the other example of a token from an /
/ attestation HW block /

601({
  / nonce /          10: h'948f8860d13a463e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID /          258: 64242, / Private Enterprise Number /
  / security-level / 261: 4, / hardware level security /
  / secure-boot /    262: true,
  / debug-status /   263: 3, / disabled-permanently /
  / hw version /     260: [ "3.1", 1 ], / multipartnumeric /
  / submods/         266: {
                                "TEE": [ / detached digest submod /
                                          -16, / SHA-256 /
                                          h'e5cf95fd24fab7144674
                                          2dd58d43dae178e55fe2
                                          b94291a9291082ffc2635
                                          a0b'
                                ]
  }
})

```

#### A.5. Key / Key Store Attestation

```

/ This is an attestation of a public key and the key store /
/ implementation that protects and manages it. The key store /
/ implementation is in a security-oriented execution /
/ environment separate from the high-level OS, for example a /
/ TEE. The key store is the Attester. /
/ /
/ There is some attestation of the high-level OS, just version /
/ and boot & debug status. It is a Claims-Set submodule because /
/ it has lower security level than the key store. The key /
/ store's implementation has access to info about the HLOS, so /
/ it is able to include it. /
/ /
/ A key and an indication of the user authentication given to /
/ allow access to the key is given. The labels for these are /
/ in the private space since this is just a hypothetical /
/ example, not part of a standard protocol. /
/ /
/ This is similar to Android Key Attestation. /

```

```
601({
```



```

/ nonce /          10: h'948f8860d13a463e',
/ security-level / 261: 3, / secure-restricted /
/ secure-boot /    262: true,
/ debug-status /   263: 2, / disabled-since-boot /
/ manifests /      273: [
                                h'da53574944a600683762623334383766
                                0c000169436172626f6e6974650d6331
                                2e320e0102a2181f75496e6475737472
                                69616c204175746f6d6174696f6e1821
                                02'
                                / Above is an encoded CoSWID      /
                                / with the following data          /
                                /   SW Name: "Carbonite"            /
                                /   SW Vers: "1.2"                  /
                                /   SW Creator:                      /
                                /     "Industrial Automation"       /
                                ],
/ expiration /      4: 1634324274, / 2021-10-15T18:57:54Z /
/ creation time /   6: 1634317080, / 2021-10-15T16:58:00Z /
-80000 : "fingerprint",
-80001 : { / The key -- A COSE_Key /
/ kty /          1: 2, / EC2, elliptic curve with x & y /
/ kid /          2: h'36675c206f96236c3f51f54637b94ced',
/ curve /        -1: 2, / curve is P-256 /
/ x-coord /      -2: h'65eda5a12577c2bae829437fe338701a
                    10aaa375e1bb5b5de108de439c08551d',
/ y-coord /      -3: h'1e52ed75701163f7f9e40ddf9f341b3d
                    c9ba860af7e0ca7ca7e9eecd0084d19c'
},

/ submods /        266 : {
                        "HLOS" : { / submod for high-level OS /
/ nonce /          10: h'948f8860d13a463e',
/ security-level / 261: 1, / unrestricted /
/ secure-boot /    262: true,
/ manifests /      273: [
                                h'da53574944a600687337
                                6537346b78380c000168
                                44726f6964204f530d65
                                52322e44320e0302a218
                                1f75496e647573747269
                                616c204175746f6d6174
                                696f6e182102'
                                / Above is an encoded CoSWID /
                                / with the following data:      /
                                /   SW Name: "Droid OS"          /
                                /   SW Vers: "R2.D2"              /
                                /   SW Creator:                    /

```

```
        /      "Industrial Automation"/  
      ]  
    }  
  }  
))
```

#### A.6. SW Measurements of an IoT Device

This is a simple token that might be for an IoT device. It includes CoSWID format measurements of the SW. The CoSWID is in byte-string wrapped in the token and also shown in diagnostic form.

```

/ This EAT UCCS is for an IoT device with a TEE. The attestation /
/ is produced by the TEE. There is a submodule for the IoT OS (the /
/ main OS of the IoT device that is not as secure as the TEE). The /
/ submodule contains claims for the IoT OS. The TEE also measures /
/ the IoT OS and puts the measurements in the submodule. /

```

```

601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 261: 3, / secure-restricted /
  / secure-boot /    262: true,
  / debug-status /   263: 2, / disabled-since-boot /
  / OEMID /          258: h'8945ad', / IEEE CID based /
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / sumods /         266: {
    "OS" : {
      / security-level / 261: 2, / restricted /
      / secure-boot /    262: true,
      / debug-status /   263: 2, / disabled-since-boot /
      / swevidence /     274: [
        / This is a byte-string wrapped /
        / evidence CoSWID. It has /
        / hashes of the main files of /
        / the IoT OS. /
        h'da53574944a600663463613234350c
        17016d41636d6520522d496f542d4f
        530d65332e312e3402a2181f724163
        6d6520426173652041747465737465
        7218210103a11183a318187161636d
        655f725f696f745f6f732e65786514
        1a0044b349078201582005f6b327c1
        73b4192bd2c3ec248a292215eab456
        611bf7a783e25c1782479905a31818
        6d7265736f75726365732e72736314
        1a000c38b10782015820c142b9aba4
        280c4bb8c75f716a43c99526694caa
        be529571f5569bb7dc542f98a31818
        6a636f6d6d6f6e2e6c6962141a0023
        3d3b0782015820a6a9dcdfb3884da5
        f884e4e1e8e8629958c2dbc7027414
        43a913e34de9333be6'
      ]
    }
  }
})

```

```

/ An evidence CoSWID created for the "Acme R-IoT-OS" created by /
/ the "Acme Base Attester" (both fictitious names). It provides /

```

```

/ measurements of the SW (other than the attester SW) on the /
/ device. /

1398229316({
  / Unique CoSWID ID /      0: "4ca245",
  / tag-version /          12: 23, / Attester-maintained counter /
  / software-name /        1: "Acme R-IoT-OS",
  / software-version /     13: "3.1.4",
  / entity /               2: {
    / entity-name /        31: "Acme Base Attester",
    / role /               33: 1 / tag-creator /
  },
  / evidence /             3: {
    / ...file /            17: [
      {
        / ...fs-name /      24: "acme_r_iot_os.exe",
        / ...size /         20: 4502345,
        / ...hash /         7: [
          1, / SHA-256 /
          h'05f6b327c173b419
          2bd2c3ec248a2922
          15eab456611bf7a7
          83e25c1782479905'
        ]
      },
      {
        / ...fs-name /      24: "resources.rsc",
        / ...size /         20: 800945,
        / ...hash /         7: [
          1, / SHA-256 /
          h'c142b9aba4280c4b
          b8c75f716a43c995
          26694caabe529571
          f5569bb7dc542f98'
        ]
      },
      {
        / ...fs-name /      24: "common.lib",
        / ...size /         20: 2309435,
        / ...hash /         7: [
          1, / SHA-256 /
          h'a6a9dcdfb3884da5
          f884e4e1e8e86299
          58c2dbc702741443
          a913e34de9333be6'
        ]
      }
    ]
  }
}
]

```

```
    }  
  })
```

#### A.7. Attestation Results in JSON format

This is a UJCS format token that might be the output of a Verifier that evaluated the IoT Attestation example immediately above.

This particular Verifier knows enough about the TEE Attester to be able to pass claims like security level directly through to the Relying Party. The Verifier also knows the Reference Values for the measured SW components and is able to check them. It informs the Relying Party that they were correct in the swresults claim. "Trustus Verifications" is the name of the services that verifies the SW component measurements.

This UJCS is identical to JSON-encoded Claims-Set that could be a JWT payload.

```
{  
  "nonce" : "lI+IYNE6Rj4=",  
  "secllevel" : "secure-restricted",  
  "secboot" : true,  
  "dbgstat" : "disabled-since-boot",  
  "OEMID" : "iUWt",  
  "UEID" : "AZjlCk/2wFhhyIYNE6Y4",  
  "submods" : {  
    "secllevel" : "restricted",  
    "secboot" : true,  
    "dbgstat" : "disabled-since-boot",  
    "swname" : "Acme R-IoT-OS",  
    "sw-version" : [  
      "3.1.4"  
    ],  
    "swresults" : [  
      [  
        "Trustus Verifications",  
        "all",  
        "fully-verified"  
      ]  
    ]  
  }  
}
```

## Appendix B. UEID Design Rationale

## B.1. Collision Probability

This calculation is to determine the probability of a collision of UEIDs given the total possible entity population and the number of entities in a particular entity management database.

Three different sized databases are considered. The number of devices per person roughly models non-personal devices such as traffic lights, devices in stores they shop in, facilities they work in and so on, even considering individual light bulbs. A device may have individually attested subsystems, for example parts of a car or a mobile phone. It is assumed that the largest database will have at most 10% of the world's population of devices. Note that databases that handle more than a trillion records exist today.

The trillion-record database size models an easy-to-imagine reality over the next decades. The quadrillion-record database is roughly at the limit of what is imaginable and should probably be accommodated. The 100 quadrillion database is highly speculative perhaps involving nanorobots for every person, livestock animal and domesticated bird. It is included to round out the analysis.

Note that the items counted here certainly do not have IP address and are not individually connected to the network. They may be connected to internal buses, via serial links, Bluetooth and so on. This is not the same problem as sizing IP addresses.

People	Devices / Person	Subsystems / Device	Database Portion	Database Size
10 billion	100	10	10%	trillion ( $10^{12}$ )
10 billion	100,000	10	10%	quadrillion ( $10^{15}$ )
100 billion	1,000,000	10	10%	100 quadrillion ( $10^{17}$ )

This is conceptually similar to the Birthday Problem where  $m$  is the number of possible birthdays, always 365, and  $k$  is the number of people. It is also conceptually similar to the Birthday Attack where collisions of the output of hash functions are considered.

The proper formula for the collision calculation is

$$p = 1 - e^{\{-k^2/(2n)\}}$$

p Collision Probability  
 n Total possible population  
 k Actual population

However, for the very large values involved here, this formula requires floating point precision higher than commonly available in calculators and SW so this simple approximation is used. See [BirthdayAttack].

$$p = k^2 / 2n$$

For this calculation:

p Collision Probability  
 n Total population based on number of bits in UEID  
 k Population in a database

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10 <sup>12</sup> )	2 * 10 <sup>-15</sup>	8 * 10 <sup>-35</sup>	5 * 10 <sup>-55</sup>
quadrillion (10 <sup>15</sup> )	2 * 10 <sup>-09</sup>	8 * 10 <sup>-29</sup>	5 * 10 <sup>-49</sup>
100 quadrillion (10 <sup>17</sup> )	2 * 10 <sup>-05</sup>	8 * 10 <sup>-25</sup>	5 * 10 <sup>-45</sup>

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 <sup>12</sup> )	60,000 years	10 <sup>24</sup> years	10 <sup>44</sup> years
quadrillion (10 <sup>15</sup> )	8 seconds	10 <sup>14</sup> years	10 <sup>34</sup> years
100 quadrillion (10 <sup>17</sup> )	8 microseconds	10 <sup>11</sup> years	10 <sup>31</sup> years

Clearly, 128 bits is enough for the near future thus the requirement that UEIDs be a minimum of 128 bits.

There is no requirement for 256 bits today as quadrillion-record databases are not expected in the near future and because this time-to-collision calculation is a very worst case. A future update of the standard may increase the requirement to 256 bits, so there is a requirement that implementations be able to receive 256-bit UEIDs.

## B.2. No Use of UUID

A UEID is not a UUID [RFC4122] by conscious choice for the following reasons.

UUIDs are limited to 128 bits which may not be enough for some future use cases.

Today, cryptographic-quality random numbers are available from common CPUs and hardware. This hardware was introduced between 2010 and 2015. Operating systems and cryptographic libraries give access to this hardware. Consequently, there is little need for implementations to construct such random values from multiple sources on their own.

Version 4 UUIDs do allow for use of such cryptographic-quality random numbers, but do so by mapping into the overall UUID structure of time and clock values. This structure is of no value here yet adds complexity. It also slightly reduces the number of actual bits with entropy.

UUIDs seem to have been designed for scenarios where the implementor does not have full control over the environment and uniqueness has to be constructed from identifiers at hand. UEID takes the view that



hardware, software and/or manufacturing process directly implement UEID in a simple and direct way. It takes the view that cryptographic quality random number generators are readily available as they are implemented in commonly used CPU hardware.

#### Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)

This section describes several distinct ways in which an IEEE IDevID [IEEE.802.1AR] relates to EAT, particularly to UEID and SUEID.

[IEEE.802.1AR] orients around the definition of an implementation called a "DevID Module." It describes how IDevIDs and LDevIDs are stored, protected and accessed using a DevID Module. A particular level of defense against attack that should be achieved to be a DevID is defined. The intent is that IDevIDs and LDevIDs are used with an open set of network protocols for authentication and such. In these protocols the DevID secret is used to sign a nonce or similar to proof the association of the DevID certificates with the device.

By contrast, EAT defines network protocol for proving trustworthiness to a Relying Party, the very thing that is not defined in [IEEE.802.1AR]. Nor does not give details on how keys, data and such are stored protected and accessed. EAT is intended to work with a variety of different on-device implementations ranging from minimal protection of assets to the highest levels of asset protection. It does not define any particular level of defense against attack, instead providing a set of security considerations.

EAT and DevID can be viewed as complimentary when used together or as competing to provide a device identity service.

##### C.1. DevID Used With EAT

As just described, EAT defines a network protocol and [IEEE.802.1AR] doesn't. Vice versa, EAT doesn't define a device implementation and DevID does.

Hence, EAT can be the network protocol that a DevID is used with. The DevID secret becomes the attestation key used to sign EATs. The DevID and its certificate chain become the Endorsement sent to the Verifier.

In this case the EAT and the DevID are likely to both provide a device identifier (e.g. a serial number). In the EAT it is the UEID (or SUEID). In the DevID (used as an endorsement), it is a device serial number included in the subject field of the DevID certificate. It is probably a good idea in this use for them to be the same serial number or for the UEID to be a hash of the DevID serial number.

### C.2. How EAT Provides an Equivalent Secure Device Identity

The UEID, SUEID and other claims like OEM ID are equivalent to the secure device identity put into the subject field of a DevID certificate. These EAT claims can represent all the same fields and values that can be put in a DevID certificate subject. EAT explicitly and carefully defines a variety of useful claims.

EAT secures the conveyance of these claims by having them signed on the device by the attestation key when the EAT is generated. EAT also signs the nonce that gives freshness at this time. Since these claims are signed for every EAT generated, they can include things that vary over time like GPS location.

DevID secures the device identity fields by having them signed by the manufacturer of the device sign them into a certificate. That certificate is created once during the manufacturing of the device and never changes so the fields cannot change.

So in one case the signing of the identity happens on the device and the other in a manufacturing facility, but in both cases the signing of the nonce that proves the binding to the actual device happens on the device.

While EAT does not specify how the signing keys, signature process and storage of the identity values should be secured against attack, an EAT implementation may have equal defenses against attack. One reason EAT uses CBOR is because it is simple enough that a basic EAT implementation can be constructed entirely in hardware. This allows EAT to be implemented with the strongest defenses possible.

### C.3. An X.509 Format EAT

It is possible to define a way to encode EAT claims in an X.509 certificate. For example, the EAT claims might be mapped to X.509 v3 extensions. It is even possible to stuff a whole CBOR-encoded unsigned EAT token into a X.509 certificate.

If that X.509 certificate is an IDevID or LDevID, this becomes another way to use EAT and DevID together.

Note that the DevID must still be used with an authentication protocol that has a nonce or equivalent. The EAT here is not being used as the protocol to interact with the rely party.

#### C.4. Device Identifier Permanence

In terms of permanence, an IDevID is similar to a UEID in that they do not change over the life of the device. They cease to exist only when the device is destroyed.

An SUEID is similar to an LDevID. They change on device life-cycle events.

[IEEE.802.1AR] describes much of this permanence as resistant to attacks that seek to change the ID. IDevID permanence can be described this way because [IEEE.802.1AR] is oriented around the definition of an implementation with a particular level of defense against attack.

EAT is not defined around a particular implementation and must work on a range of devices that have a range of defenses against attack. EAT thus can't be defined permanence in terms of defense against attack. EAT's definition of permanence is in terms of operations and device lifecycle.

#### Appendix D. Changes from Previous Drafts

The following is a list of known changes from the previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences.

##### D.1. From draft-rats-eat-01

- o Added UEID design rationale appendix

##### D.2. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but no new claims have been added.

- o Separate information and data model using CDDL.
- o Say an EAT is a CWT or JWT
- o Use a map to structure the boot\_state and location claims

##### D.3. From draft-ietf-rats-eat-01

- o Clarifications and corrections for OEMID claim
- o Minor spelling and other fixes

- o Add the nonce claim, clarify jti claim
- D.4. From draft-ietf-rats-eat-02
- o Roll all EUIs back into one UEID type
  - o UEIDs can be one of three lengths, 128, 192 and 256.
  - o Added appendix justifying UEID design and size.
  - o Submods part now includes nested eat tokens so they can be named and there can be more than one of them
  - o Lots of fixes to the CDDL
  - o Added security considerations
- D.5. From draft-ietf-rats-eat-03
- o Split boot\_state into secure-boot and debug-disable claims
  - o Debug disable is an enumerated type rather than Booleans
- D.6. From draft-ietf-rats-eat-04
- o Change IMEI-based UEIDs to be encoded as a 14-byte string
  - o CDDL cleaned up some more
  - o CDDL allows for JWTs and UCCSs
  - o CWT format submodules are byte string wrapped
  - o Allows for JWT nested in CWT and vice versa
  - o Allows UCCS (unsigned CWTs) and JWT unsecured tokens
  - o Clarify tag usage when nesting tokens
  - o Add section on key inclusion
  - o Add hardware version claims
  - o Collected CDDL is now filled in. Other CDDL corrections.
  - o Rename debug-disable to debug-status; clarify that it is not extensible

- o Security level claim is not extensible
  - o Improve specification of location claim and added a location privacy section
  - o Add intended use claim
- D.7. From draft-ietf-rats-eat-05
- o CDDL format issues resolved
  - o Corrected reference to Location Privacy section
- D.8. From draft-ietf-rats-eat-06
- o Added boot-seed claim
  - o Rework CBOR interoperability section
  - o Added profiles claim and section
- D.9. From draft-ietf-rats-eat-07
- o Filled in IANA and other sections for possible preassignment of Claim Keys for well understood claims
- D.10. From draft-ietf-rats-eat-08
- o Change profile claim to be either a URL or an OID rather than a test string
- D.11. From draft-ietf-rats-eat-09
- o Add SUEIDs
  - o Add appendix comparing IDevID to EAT
  - o Added section on use for Evidence and Attestation Results
  - o Fill in the key ID and endorsements identificaiton section
  - o Remove origination claim as it is replaced by key IDs and endorsements
  - o Added manifests and software evidence claims
  - o Add string labels non-claim labels for use with JSON (e.g. labels for members of location claim)

- o EAN-13 HW versions are no longer a separate claim. Now they are folded in as a CoSWID version scheme.

D.12. From draft-ietf-rats-eat-10

- o Hardware version is made into an array of two rather than two claims
- o Corrections and wording improvements for security levels claim
- o Add swresults claim
- o Add dloas claim - Digital Letter of Approvals, a list of certifications
- o CDDL for each claim no longer in a separate sub section
- o Consistent use of terminology from RATS architecture document
- o Consistent use of terminology from CWT and JWT documents
- o Remove operating model and procedures; refer to CWT, JWT and RATS architecture instead
- o Some reorganization of Section 1
- o Moved a few references, including RATS Architecture, to informative.
- o Add detached submodule digests and detached eat bundles (DEBs)
- o New simpler and more universal scheme for identifying the encoding of a nested token
- o Made clear that CBOR and JSON are only mixed when nesting a token in another token
- o Clearly separate CDDL for JSON and CBOR-specific data items
- o Define UJCS (unsigned JWTs)
- o Add CDDL for a general Claims-Set used by UCCS, UJCS, CWT, JWT and EAT
- o Top level CDDL for CWT correctly refers to COSE
- o OEM ID is specifically for HW, not for SW

- o HW OEM ID can now be a PEN
- o HW OEM ID can now be a 128-bit random number
- o Expand the examples section
- o Add software and version claims as easy / JSON alternative to CoSWID

D.13. From draft-ietf-rats-eat-11

- o Add HW model claim
- o Change reference for CBOR OID draft to RFC 9090
- o Correct the iat claim in some examples
- o Make HW Version just one claim rather than 3 (device, board and chip)
- o Remove CDDL comments from CDDL blocks
- o More clearly define "entity" and use it more broadly, particularly instead of "device"
- o Re do early allocation of CBOR labels since last one didn't complete correctly
- o Lots of rewording and tightening up of section 1
- o Lots of wording improvements in section 3, particularly better use of normative language
- o Improve wording in submodules section, particularly how to distinguish types when decoding
- o Remove security-level from early allocation
- o Add boot odometer claim
- o Add privacy considerations for replay protection

Authors' Addresses

Laurence Lundblade  
Security Theory LLC

EMail: [lg1@securitytheory.com](mailto:lg1@securitytheory.com)

Giridhar Mandyam  
Qualcomm Technologies Inc.  
5775 Morehouse Drive  
San Diego, California  
USA

Phone: +1 858 651 7200  
EMail: mandyam@qti.qualcomm.com

Jeremy O'Donoghue  
Qualcomm Technologies Inc.  
279 Farnborough Road  
Farnborough GU14 7LS  
United Kingdom

Phone: +44 1252 363189  
EMail: jodonogh@qti.qualcomm.com



Rats Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 3, 2020

G. Mandyam  
V. Sekhar  
S. Mohammed  
Qualcomm Technologies Inc.  
October 31, 2019

The Qualcomm Wireless Edge Services (QWES) Attestation Token  
draft-mandyam-rats-qwestoken-00

Abstract

An attestation format based on the Entity Attestation Token (EAT) is described. The Qualcomm Wireless Edge Services (QWES) token is used in the context of device onboarding and authentication. It is verified in the same manner as any CBOR Web Token (CWT).

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	2
2. EAT Compliant Claims in QWES Token . . . . .	2
2.1. OEM ID . . . . .	2
2.2. nonce . . . . .	2
3. QWES Token Augmentation to EAT . . . . .	3
3.1. DevID . . . . .	3
3.2. HWVer . . . . .	3
3.3. Context . . . . .	3
3.4. PKHash . . . . .	3
3.5. SPID . . . . .	3
3.6. QSEESVersion . . . . .	3
3.7. FWVersion . . . . .	3
3.8. Security State . . . . .	4
3.9. CSR . . . . .	4
3.10. AppData . . . . .	4
4. Example . . . . .	4
5. IANA Considerations . . . . .	4
6. Normative References . . . . .	5
Authors' Addresses . . . . .	5

## 1. Introduction

A description of the Qualcomm Wireless Edge Services (QWES) attestation token is provided. QWES allows for service providers to manage devices that implement Qualcomm semiconductor solutions. Based on the EAT-compliant attestation token (see [I-D.ietf-rats-eat]) produced in a trusted execution environment (TEE), a service provider can verify the device identity in addition to several other security-impacting characteristics.

## 2. EAT Compliant Claims in QWES Token

Certain claims defined for EAT are leveraged in the QWES token.

## 2.1. OEM ID

The QWES token makes use of an OEM ID. However, the type is a uint (not a bstr as per the EAT specification).

## 2.2. nonce

The QWES token can carry a nonce. The nonce is a bstr.

### 3. QWES Token Augmentation to EAT

Several claims have been defined that are not currently present in the EAT base set to complete the QWES token.

#### 3.1. DevID

The DevID is a 256-bit bstr that serves as a device identifier. It differs from the ueid (Universal Entity ID) defined in the EAT specification. A specific device ID can be created for ISV's based on their identifier combined with a salt. This allows for a certain level of privacy preservation. Note that the RAND option for ueid may be a suitable substitute for this claim.

#### 3.2. HWVer

This is a bstr claim that distinguishes different system-on-chip (SoC) models.

#### 3.3. Context

This is a numerical (uint) index that denotes the context of the attestation. The current values defined (0-4) are: on-demand attestation, registration, provisioning, certificate issuance and proof-of-possession.

#### 3.4. PKHash

This is a bstr containing a SHA-256 hash of a public key provisioned by the OEM. It can be used optionally in place of an OEM ID.

#### 3.5. SPID

This is a numerical (uint) identifier of the service provider associated with the QWES token. The EAT specification's origination claim can be a suitable substitute.

#### 3.6. QSEVersion

This is tstr that designates the TEE version ("QSEE" = Qualcomm Secure Execution Environment).

#### 3.7. FWVersion

This is tstr that designates the firmware version specifically dedicated to bootstrapping.

### 3.8. Security State

This is tstr that contains the state of one-time programmable (OTP) memory. Bits in this field will be set as per the security-impacting section of the OTP memory. The relevant bits in OTP would normally be used to control secure boot enablement, debug disablement, debug enablement parameters, and the state of device keys (i.e. whether they are locked for reading or writing).

### 3.9. CSR

A Certificate Signing Request (CSR) may be carried in a QWES token as a bstr. This allows for a CA (certifying authority) to verify an attestation and provide a certificate without an extra round trip. This is an optional claim.

### 3.10. AppData

This is a bstr containing a hash of the associated application data. Since the QWES token can be service provider-specific, the hash that is returned can correspond to the corresponding user space application that invoked the generation of the attestation token. This is an optional claim.

## 4. Example

A sample QWES token payload is shown. It would be signed and/or encrypted as per COSE guidelines.

```
{
  / DevID /                h'0a',
  / OEMID /                32,
  / HWVer /                h'0e',
  / Context /              2, / provisioning /
  / SPID /                 10,
  / Nonce /                h'da378321bb'
}
```

Sample QWES Token Payload

## 5. IANA Considerations

This memo includes no request to IANA.

## 6. Normative References

- [I-D.ietf-rats-eat]  
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-00 (work in progress), June 2019.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [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>>.

## Authors' Addresses

Giridhar Mandyam  
Qualcomm Technologies Inc.  
5775 Morehouse Drive  
San Diego, California 92121  
USA

Phone: +1 858 651 7200  
Email: [mandyam@qti.qualcomm.com](mailto:mandyam@qti.qualcomm.com)

Vivek Sekhar  
Qualcomm Technologies Inc.  
5775 Morehouse Drive  
San Diego, California 92121  
USA

Phone: +1 858 651 3557  
Email: [vsekhar@qti.qualcomm.com](mailto:vsekhar@qti.qualcomm.com)

Shahid Mohammed  
Qualcomm Technologies Inc.  
5775 Morehouse Drive  
San Diego, California 92121  
USA

Phone: +1 858 651 7975  
Email: shahidm@qti.qualcomm.com

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 7, 2020

D. Thaler  
Microsoft  
November 04, 2019

Remote Attestation Architecture  
draft-thaler-rats-architecture-01

Abstract

In network protocol exchanges, it is often the case that one entity (a relying party) requires evidence about the remote peer (and system components [RFC4949] thereof), in order to assess the trustworthiness of the peer. This document describes an architecture for such remote attestation procedures (RATS), which enable relying parties to decide whether to consider a remote system component trustworthy or not.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 7, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. Use Cases . . . . .	4
3.1. Network Endpoint Assessment . . . . .	4
3.2. Confidential Machine Learning (ML) Model Protection . . . . .	5
3.3. Confidential Data Retrieval . . . . .	5
3.4. Critical Infrastructure Control . . . . .	5
3.5. Trusted Execution Environment (TEE) Provisioning . . . . .	6
3.6. Hardware Watchdog . . . . .	6
4. Serialization Formats . . . . .	7
5. Architectural Models . . . . .	8
5.1. Passport Model . . . . .	8
5.2. Background-Check Model . . . . .	9
5.2.1. Variation: Verifying Relying Party . . . . .	10
5.2.2. Variation: Out-of-Band Evidence Conveyance . . . . .	10
5.3. Combinations . . . . .	11
6. Trust Model . . . . .	12
7. Conceptual Messages . . . . .	12
7.1. Evidence . . . . .	12
7.2. Endorsements . . . . .	13
7.3. Attestation Results . . . . .	13
8. Security Considerations . . . . .	14
9. IANA Considerations . . . . .	14
10. Acknowledgements . . . . .	14
11. Informative References . . . . .	14
Author's Address . . . . .	15

## 1. Introduction

In network protocol exchanges, it is often the case that one entity (a relying party) requires evidence about the remote peer (and system components [RFC4949] thereof), in order to assess the trustworthiness of the peer. Remote attestation procedures (RATS) enable relying parties to establish a level of confidence in the trustworthiness of remote system components through the creation of attestation evidence by remote system components and a processing chain towards the relying party. A relying party can then decide whether to consider a remote system component trustworthy or not.

To improve the confidence in a system component's trustworthiness, a relying party may require evidence about:

- system component identity,



- composition of system components, including nested components,
- roots of trust,
- assertion/claim origination or provenance,
- manufacturing origin,
- system component integrity,
- system component configuration,
- operational state and measurements of steps which led to the operational state, or
- other factors that could influence trust decisions.

This document discusses an architecture for describing solutions for this problem.

## 2. Terminology

This document uses the following terms:

- **Attestation:** A process by which one entity (the "Attester") provides evidence about its identity and health to another entity, which then assesses its trustworthiness.
- **Attestation Result:** The evaluation results generated by a Verifier, typically including information about an Attester, where the Verifier vouches for the validity of the results.
- **Attester:** An entity whose attributes must be evaluated in order to determine whether the entity is considered healthy or authorized to access a resource.
- **Endorsement:** A secure statement that some entity (typically a manufacturer) vouches for the integrity of an Attester's signing capability. (Note: in some discussions the entity providing an Endorsement has been called an Asserter, but some believe that term is confusing and the term Endorser would be more correct. For now, this document avoids using a specific term until consensus is reached.)
- **Evidence:** A set of information about an Attester that is to be evaluated by a Verifier.

- Relying Party: An entity that depends on the validity of information about another entity, typically for purposes of authorization. Compare /relying party/ in [RFC4949].
- Security policy: A set of rules that direct how a system evaluates the validity of information about another entity. For example, the security policy might involve an equality comparison against known-good values (called Reference Integrity Measurements in some contexts), or might involve more complex logic. Compare /security policy/ in [RFC4949].
- Verifier: An entity that evaluates the validity of information about an Attester.

### 3. Use Cases

This section covers a number of representative use cases for attestation, independent of solution. The purpose is to provide motivation for various aspects of the architecture presented in this draft. Many other use cases exist, and this document does not intend to have a complete list, only to have a set of use cases that collectively cover all the functionality required in the architecture. The use cases are covered prior to discussion of architectural models in Section 5, since each use case might be addressed via different solutions that have different architectural models.

Each use case includes a description, and a summary of what an Attester and a Relying Party refer to in the use case. (Since solutions to a use case may greatly vary in architectural model, the role of a Verifier is considered part of a specific solution, not a solution-independent property of a use case, and so is not covered in this section.)

#### 3.1. Network Endpoint Assessment

Network operators want a trustworthy report of identity and version of information of the hardware and software on the machines attached to their network, for purposes such as inventory, auditing, and/or logging. The network operator may also want a policy by which full access is only granted to devices that meet some definition of health, and so wants to get claims about such information and verify their validity. Attestation is desired to prevent vulnerable or compromised devices from getting access to the network and potentially harming others.

Typically, solutions start with some component (called a "Root of Trust") that provides device identity and protected storage for

measurements. They then perform a series of measurements, and express this with Evidence as to the hardware and firmware/software that is running.

- Attester: A device desiring access to a network
- Relying Party: A network infrastructure device such as a router, switch, or access point.

### 3.2. Confidential Machine Learning (ML) Model Protection

A device manufacturer wants to protect its intellectual property in terms of the ML model it developed and that runs in the devices that its customers purchased, and it wants to prevent attackers, potentially including the customer themselves, from seeing the details of the model.

This typically works by having some protected environment in the device attest to some manufacturer service. If attestation succeeds, then the manufacturer service releases either the model, or a key to decrypt a model the Attester already has in encrypted form, to the requester.

- Attester: A device desiring to run an ML model to do inferencing
- Relying Party: A server or service holding ML models it desires to protect

### 3.3. Confidential Data Retrieval

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. Attestation is desired to prevent leaking data to compromised devices.

- Attester: An entity desiring to retrieve confidential data
- Relying Party: An entity that holds confidential data for retrieval by other entities

### 3.4. Critical Infrastructure Control

In this use case, potentially dangerous physical equipment (e.g., power grid, traffic control, hazardous chemical processing, etc.) is connected to a network. The organization managing such infrastructure needs to ensure that only authorized code and users can control such processes, and they are protected from malware or

other adversaries. When a protocol operation can affect some critical system, the device attached to the critical equipment thus wants some assurance that the requester has not been compromised. As such, 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).

### 3.5. Trusted Execution Environment (TEE) Provisioning

A "Trusted Application Manager (TAM)" server is responsible for managing the applications running in the TEE of a client device. To do this, the TAM wants to verify the state of a TEE, or of applications in the TEE, of a client device. The TEE attests to the TAM, which can then decide whether the TEE is already in compliance with the TAM's latest policy, or if the TAM needs 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 trusted execution environment capable of running trusted applications that can be updated.
- Relying Party: A Trusted Application Manager.

### 3.6. Hardware Watchdog

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

A hardware watchdog can be implemented by forcing a reboot unless attestation to a remote server succeeds within a periodic interval, and having the reboot do remediation by bringing a device into compliance, including installation of patches as needed.

- Attester: The device that is desired to keep from being held hostage for a long period of time.
- Relying Party: A remote server that will securely grant the Attester permission to continue operating (i.e., not reboot) for a period of time.

#### 4. Serialization Formats

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

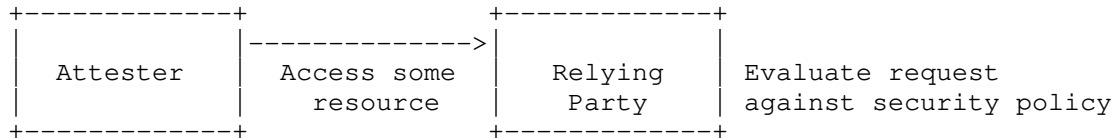


Figure 1: Typical Resource Access

In this diagram, the protocol between Attester and a Relying Party can be any new or existing protocol (e.g., HTTP(S), COAP(S), 802.1x, OPC UA, etc.), depending on the use case. Such protocols typically already have mechanisms for passing security information for purposes of authentication and authorization. Common formats include JWTs [RFC7519], CWTs [RFC8392], and X.509 certificates.

In many cases, it is desirable to add attestation to existing protocols, enabling a higher level of assurance against malware for example. To enable such integration, it is important that information needed for evaluating the Attester be usable with existing protocols that have constraints around what formats they can transport. For example, OPC UA [OPCUA] (probably the most common protocol in industrial IoT environments) is defined to carry X.509 certificates and so security information must be embedded into an X.509 certificate to be passed in the protocol. Thus, attestation-related information could be natively encoded in X.509 certificate extensions, or could be natively encoded in some other format (e.g., a CWT) which in turn is then encoded in an X.509 certificate extension.

Especially for constrained nodes, however, 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 area. So 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 the format that is natural for the Relying Party.

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

## 5. Architectural Models

There are multiple possible models for communication between an Attester, a Verifier, and a Relying Party.

### 5.1. Passport Model

In this model, an Attester sends Evidence to a Verifier, which compares the Evidence against its security policy. The Verifier then gives back an Attestation Result. If the Attestation Result was a successful one, the Attester can then present the Attestation Result to a Relying Party, which then compares the Attestation Result against its own security policy.

Since the resource access protocol between the Attester and Relying Party includes an Attestation Result, in this model the details of that protocol constrain the serialization format of the Attestation Result. The format of the Evidence on the other hand is only constrained by the Attester-Verifier attestation protocol.

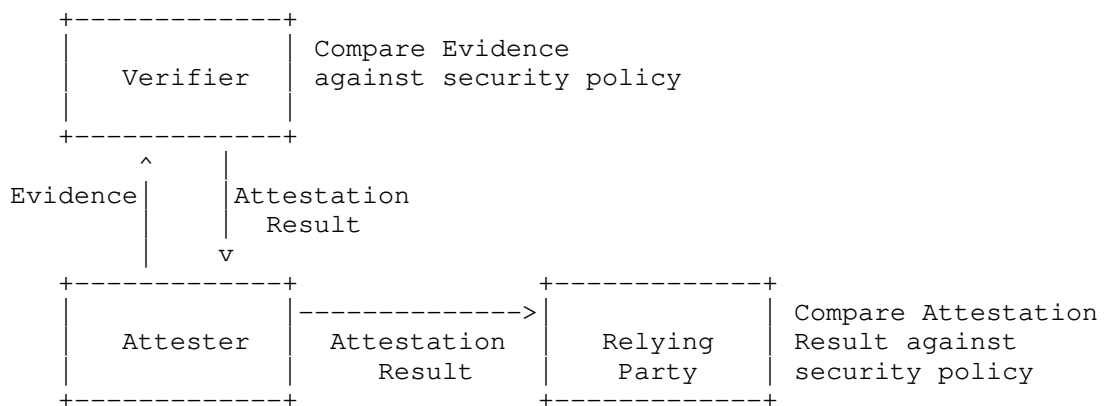


Figure 2: Passport Model

The passport model is so named because it resembles the process typically used for passports and drivers licenses, where a person applies and gets a passport or license that is issued by a government and shows information such as the person's name and birthdate. The passport or license can then be supplied to other entities to gain entrance to an airport boarding area, or age-restricted section of a bar, where the passport or license is considered sufficient because it vouches for that piece of information and is issued by a trusted authority. Thus, in this analogy, the passport issuing agency is a Verifier, the passport is an Attestation Result, and the airport security is a Relying Party.

## 5.2. Background-Check Model

In this model, an Attester sends Evidence to a Relying Party, which simply passes it on to a Verifier. The Verifier then compares the Evidence against its security policy, and returns an Attestation Result to the Relying Party. The Relying Party then compares the Attestation Result against its own security 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 and so the serialization format of the Attestation Result is still important. If the Relying Party is a constrained node whose purpose is to serve a given type resource using a standard resource access protocol, it already needs the parser(s) required by that existing protocol. Hence, the ability to let the Relying Party obtain an Attestation Result in the same serialization format allows minimizing the code footprint and attack surface area of the Relying Party, especially if the Relying Party is a constrained node.

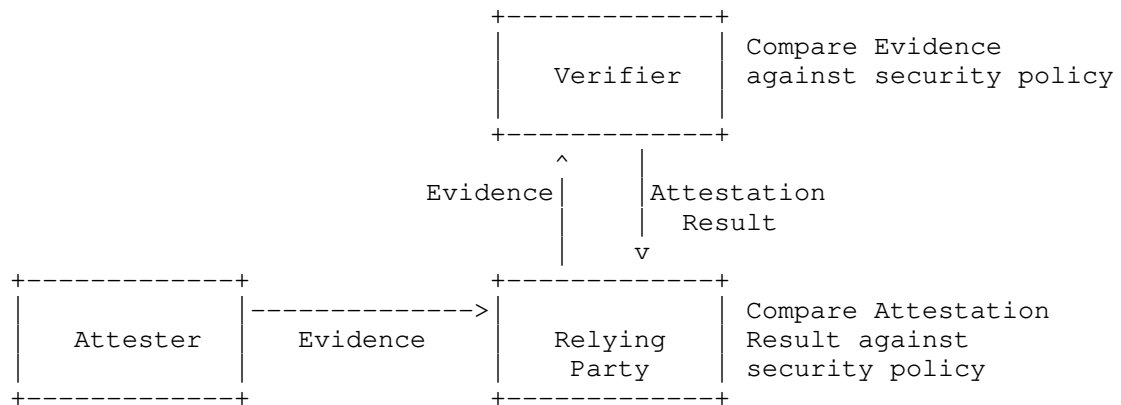


Figure 3: Background-Check Model

The background-check model is so named because it resembles the process typically used for job and loan applications, where a person fills out an application to get a job or a loan from a company, and

the company then does a background check with some other agency that checks credit history, arrest records, etc. and gives back a report on the application that is then used to help determine whether to actually offer the job or loan. Thus, in this analogy, a person asking for a loan is an Attester, the bank is the Relying Party, and a credit report agency is a Verifier.

#### 5.2.1. Variation: Verifying Relying Party

One variation of the background-check model is a "Verifying Relying party", where the Relying Party and the Verifier on the same machine, and so there is no need for a protocol between the two.

#### 5.2.2. Variation: Out-of-Band Evidence Conveyance

Another variation of the background-check model is shown in Figure 4, where the Verifier is still chosen by (and trusted by) the Relying Party, but the Evidence must be passed out-of-band. For example, in step 1, the Attester communicates with the Relying Party, which refers the matter to a Verifier chosen by the Relying Party in step 2. Evidence is then passed to that Verifier in step 3, e.g., either by the Relying Party providing the Attester with information about the Verifier to send Evidence to, or by the Verifier querying the Attester directly, although the latter has the problem that it only works if devices allow unsolicited inbound queries, which may be a security problem in some contexts.

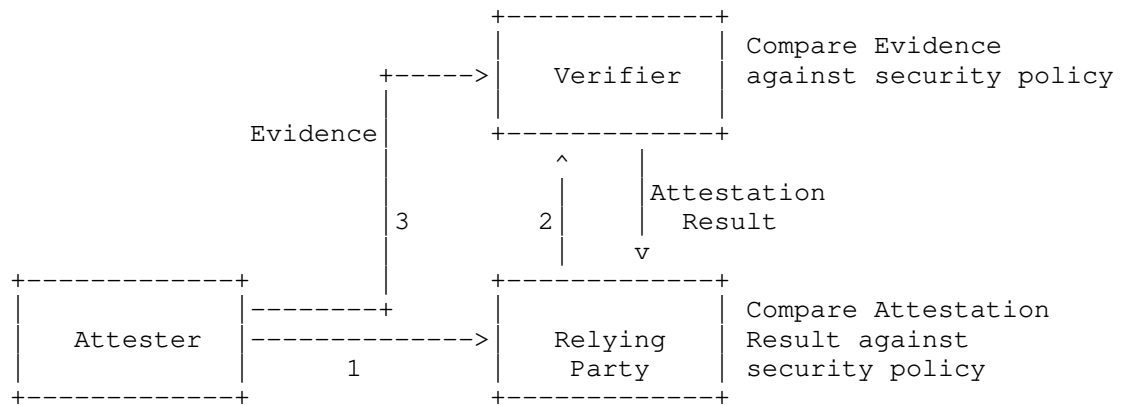


Figure 4: Out-of-Band Evidence Conveyance



### 5.3. Combinations

The choice of model is generally up to the Relying Party, and the same device may need to attest to different relying parties for different use cases (e.g., a network infrastructure device to gain access to the network, and then a server holding confidential data to get access to that data). As such, both models may simultaneously be in use by the same device.

Figure 5 shows an 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 3, Relying Party 2 actually provides the Attestation Result back to the Attester, allowing the Attester to use it with other Relying Parties. This is the model that the Trusted Application Manager plans to support in the TEEP architecture [I-D.ietf-teep-architecture].

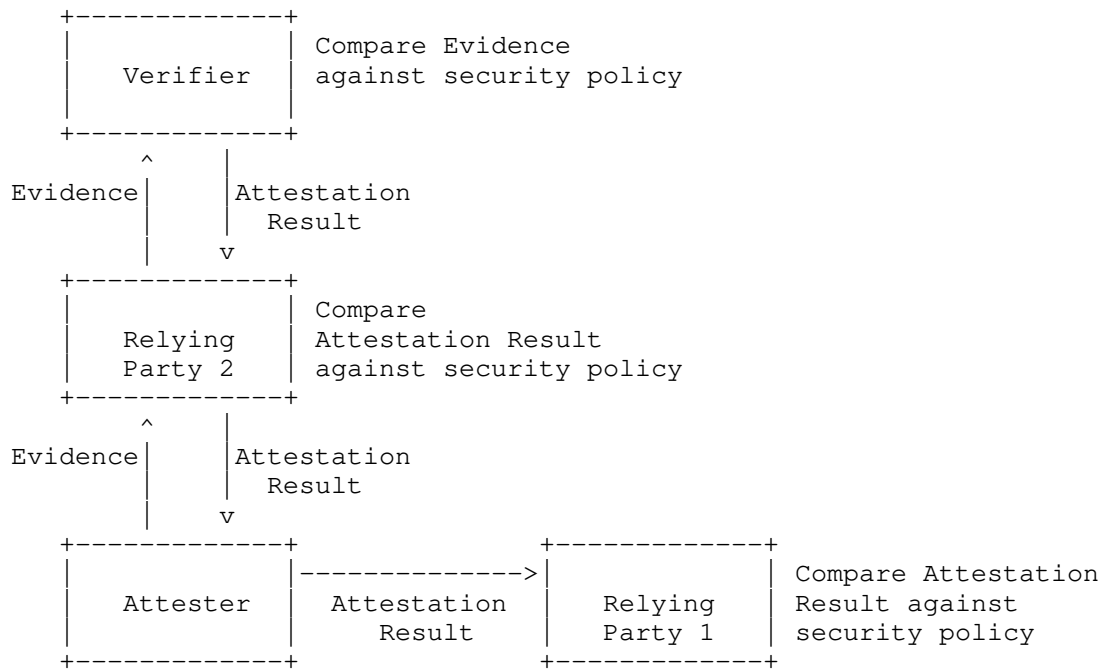


Figure 5: Example Combination

## 6. Trust Model

The scope of this document is scenarios for which a Relying Party trusts a Verifier that can evaluate the trustworthiness of information about an Attester. Such trust might come by the Relying Party trusting the Verifier (or its public key) directly, or might come by trusting an entity (e.g., a Certificate Authority) that the Verifier has a certificate that chains up to. The Relying Party might implicitly trust a Verifier (such as in the Verifying Relying Party combination). Or, for a stronger level of security, the Relying Party might require that the Verifier itself provide information about itself that the Relying Party can use to evaluate the health of the Verifier before accepting its Attestation Results.

In solutions following the background-check model, the Attester is assumed to trust the Verifier (again, whether directly or indirectly via a Certificate Authority that it trusts), since the Attester relies on an Attestation Result it obtains from the Verifier, in order to access resources.

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 evaluate the health of that manufacturer's devices. In solutions with weaker security, a Verifier might be configured to implicitly trust firmware or even software (e.g., a hypervisor). That is, it might evaluate the health of an application component, or operating system component or service, under the assumption that information provided about it by the lower-layer hypervisor or firmware is true. A stronger level 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. The component that is implicitly trusted is often referred to as a Root of Trust.

## 7. Conceptual Messages

### 7.1. Evidence

Today, Evidence tends to be highly device-specific, since the information in the evidence often includes vendor-specific information that is necessary to fully describe the manufacturer and model of the device including its security properties, the health of the device, and the level of confidence in the correctness of the information. Evidence is typically signed by the device (whether by hardware, firmware, or software on the device), and evaluating it in isolation would require security policy to be based on device-specific details (e.g., a device public key).

## 7.2. Endorsements

An Endorsement is a secure statement that some entity (typically a manufacturer) vouches for the integrity of the device's signing capability. For example, if the signing capability is in hardware, then an Endorsement might be a manufacturer certificate that signs a public key whose corresponding private key is only known inside the device's hardware. Thus, when Evidence and such an Endorsement are used together, evaluating them can be done against security policy that may not be specific to the device instance, but merely specific to the manufacturer providing the Endorsement. For example, a security policy might simply check that devices from a given manufacturer have information matching a set of known-good reference values, or a security policy might have a set of more complex logic on how to evaluate the validity of information.

However, while a security 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 a security 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 attestation for Network Endpoint Assessment 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.

## 7.3. Attestation Results

Attestation Results may indicate compliance or non-compliance with a Verifier's security policy. A 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.

An Attestation Result that indicates compliance can be used by a Relying Party to make authorization decisions based on the Relying Party's security policy. The simplest such policy might be to simply authorize any party supplying a compliant Attestation Result signed by a trusted Verifier. A more complex policy might also entail comparing information provided in the result against known-good reference values, or applying more complex logic using such information.

Thus, Attestation Results often need to include detailed information about the Attester, for use by Relying Parties, much like physical passports and drivers licenses include personal information such as name and date of birth. Unlike Evidence, which is often very device- and vendor-specific, Attestation Results can be vendor-neutral if the Verifier has a way to generate vendor-agnostic information based on evaluating vendor-specific information in Evidence. This allows a Relying Party's security 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 security policy.

## 8. Security Considerations

To evaluate the security provided by a particular security 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 security policy was itself obtained securely. As such, if security policy in a Relying Party or Verifier can be configured via a network protocol, the ability to attest to the health of the client providing the security policy needs to be considered.

## 9. IANA Considerations

This document does not require any actions by IANA.

## 10. Acknowledgements

Some content in this document came from drafts by Michael Richardson, Henk Birkholz, and Ned Smith, and from the IETF RATS Working Group Charter.

## 11. Informative References

[I-D.ietf-teep-architecture]

Pei, M., Tschofenig, H., Wheeler, D., Atyeo, A., and D. Liu, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-03 (work in progress), July 2019.

- [OPCUA] OPC Foundation, "OPC Unified Architecture Specification, Part 2: Security Model, Release 1.03", Global Platform GPD\_SPE\_009, November 2015, <<https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.

## Author's Address

Dave Thaler  
Microsoft

EMail: [dthaler@microsoft.com](mailto:dthaler@microsoft.com)

Remote ATtestation Procedures  
Internet-Draft  
Intended status: Standards Track  
Expires: April 23, 2020

L. Xia  
W. Pan  
Huawei  
October 21, 2019

Using Netconf Pub/Sub Model for RATS Interaction Procedure  
draft-xia-rats-pubsub-model-01

Abstract

This draft defines the a new method of using the netconf pub/sub model in the RATS interaction procedure, to increse its flexibility, efficiency and scalability.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 23, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	2
2. Conventions Used in This Document . . . . .	3
3. Pub/sub Model for Remote Attestation Procedure . . . . .	4
3.1. Solution Overview . . . . .	4
3.2. Remote Attestation Event Stream Definition . . . . .	7
3.3. Remote Attestation Subscription Definition . . . . .	8
3.4. Remote Attestation Selection Filters Definition . . . . .	8
3.5. Remote Attestation Subscription Parameters Handling . . . . .	9
3.6. Remote Attestation Notification Distribution . . . . .	9
3.7. Summary . . . . .	9
4. The YANG Module for Sub/pub Model Remote Attestation Procedures . . . . .	12
4.1. Tree Format . . . . .	12
4.2. Raw Format . . . . .	12
5. Security Considerations . . . . .	12
6. Acknowledgements . . . . .	12
7. IANA Considerations . . . . .	12
8. References . . . . .	12
8.1. Normative References . . . . .	12
8.2. Informative References . . . . .	13
Authors' Addresses . . . . .	14

## 1. Introduction

Remote attestation is for acquiring the evidence about various integrity information from remote endpoints to assess its trustworthiness (aka, behave in the expected manner). These evidence should be about: system component identity, composition of system components, roots of trust, system component integrity, system component configuration, operational state and so on.

[I-D.richardson-rats-usecases] describes possible use cases which remote attestation are using for different industries, like: network devices, FIDO authentication for online transaction, Cryptographic Key Attestation for mobile devices, and so on.

[I-D.birkholz-rats-architecture] lays a foundation of RATS architecture about the key RATS roles (i.e., Relying Party, Verifier, Attester and asserter) and the messages they exchange, as well as some key concepts. Based on it,

[I-D.birkholz-rats-reference-interaction-model] specifies a basic challenge-response-based interaction model for the remote attestation procedure, which a complete remote attestation procedure is triggered by a challenge message originated from the verifier, and finished when the attester sends its response message back. This is a very generic interaction model with wide adoption. This document proposes an alternative interaction model for Remote attestation procedure, by

customizing the IETF NETCONF pub/sub model [RFC8639][RFC8640][RFC8641]. With its nature of asynchronous communication, the new pub/sub model for remote attestation procedure is optimal for large-scale and loosely coupled distributed systems, especially for the network devices, which has the advantages as: loose coupling, scalability, time delivery sensitivity, supporting filtering capability, and so on. The pub/sub model can be used independently, or together with the challenge-response model to complement each other as a whole. Note that in which way these models are combined together are currently out of the scope of this draft.

In summary, by utilizing the pub/sub model in remote attestation procedure, the gained benefits are as below:

- o Flexibility: the verifier does not need to send the challenge message every time. The whole thing of the verifier is to subscribe a topic to the attester and then to anticipate the period or timely on-change notification from the attester about its integrity evidence.
- o Efficiency: once the verifier has subscribed its interested topics related with its triggering condition to the attester, it will get all the condition triggered notifications on time, which are the integrity related evidence for remote attestation in fact. It will ensure any integrity change/deviation of the remote endpoint to be detected with the minimum latency.
- o Scalability: it will save a lot of challenge messages by replacing with single subscription message for one topic stream, and decrease the total number of stateful connection between the verifier and attester, especially for a very large scale network. Thus, the scalability of the solution will increase.

This document is organized as follows. Section 2 defines conventions and acronyms used. Section 3 discusses pub/sub model of remote attestation procedure.

## 2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses terminology defined in[I-D.birkholz-rats-architecture][I-D.birkholz-rats-reference-interaction-model] for security related and RATS scoped terminology.



### 3. Pub/sub Model for Remote Attestation Procedure

#### 3.1. Solution Overview

The following sequence diagram illustrates the reference remote attestation procedure by utilizing the netconf pub/sub model defined by this document.

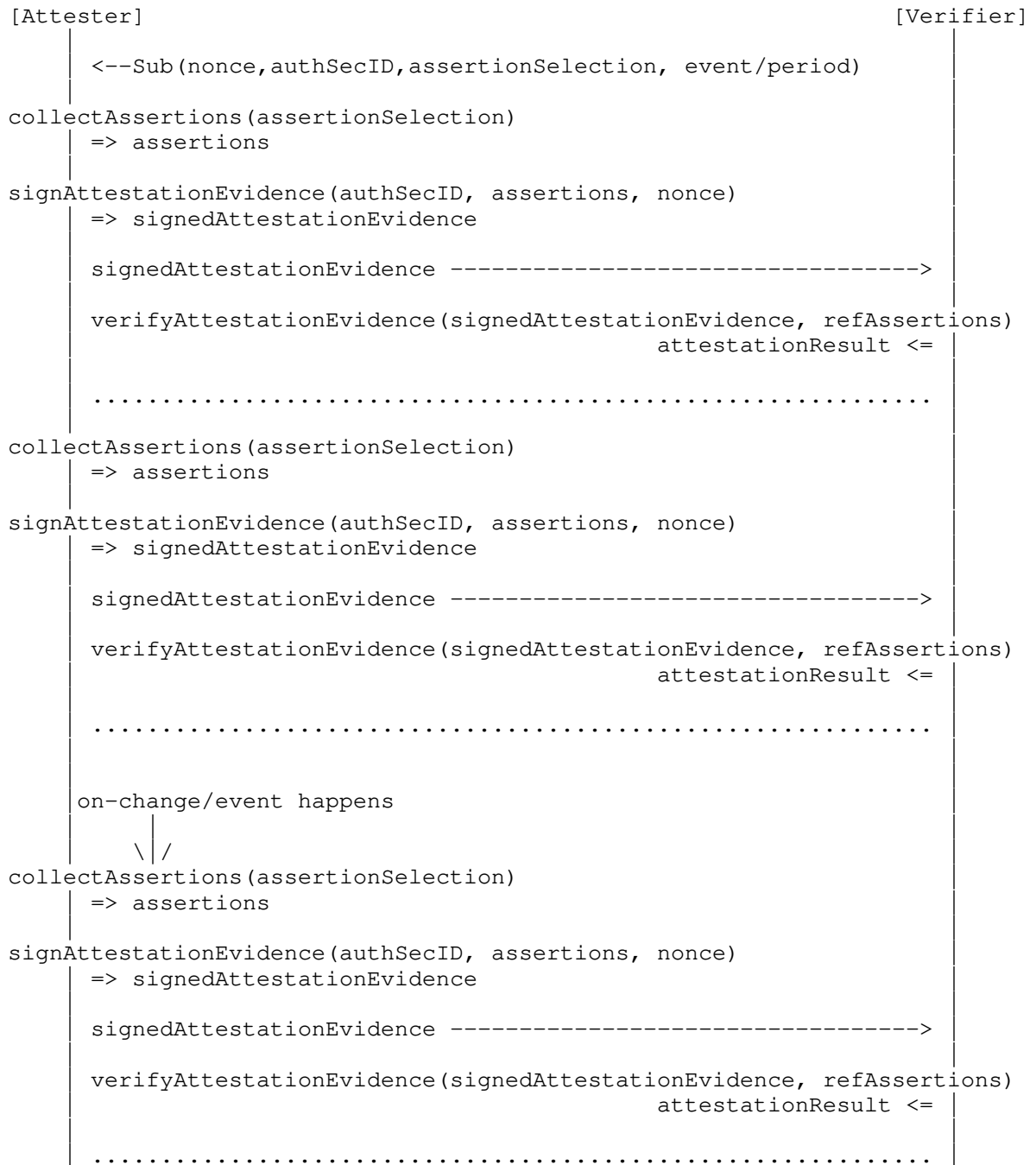


Figure 1: Pub/sub model of Remote Attestation

In short, the basic idea of pub/sub model for remote attestation is the verifier subscribes its interested event streams about the integrity evidence to the attester. After the subscription succeeds, the attester sends the subscribed integrity evidence back to the verifier. During subscription, the verifier may also specify how the attester returns the subscribed information, that is, the update trigger as periodic subscription or on-change subscription. And when the selection filters are applied to the subscription, only the information that pass the filter will be distributed out.

More detailed, the key steps of the remote attestation workflow with this model can be summarized as below (using the network devices as the example):

- o The verifier subscribes its interested event streams about the integrity evidence to the attester. More specifically:
  - \* The event stream here refers to various integrity evidence information related to device trustworthiness. The basic event streams may include: software integrity information (including PCR values and system boot logs) of each layer of the trust chain recorded during device booting time; device identity certificates & Attestation Key certificate; operating system, application dynamic integrity information (e.g., IMA logs) and the device configuration information recorded during device running time.
  - \* Periodic subscription is mainly used by the verifier for the general and non-critical information collection, which are not strictly time sensitive and aims for collecting the latest integrity evidence and checking the possible deviation. In contrary, on-change subscription is basically used to monitor the critical integrity evidence (e.g., integrity values and log files during device booting time, or integrity values of some key service processes). If these integrity values change, notifications are sent immediately.
  - \* The selection filters may be applied to the subscription, so that only the event records that pass the filter will be distributed out. Some specific examples include: event records of a component (e.g., line card) in the composite device, the event records in a specific time period that includes a start time and an end time, and so on.
- o Consider how to send the existing parameters (i.e., nonce, hash signature algorithm, and specified TPM name, etc.) carried in the challenge message of the previous challenge-response model to the attester through the subscription message of the new sub/pub model

in advance, and the follow-up usage of them. A very important point is how to ensure that the nonce carried in every notification message is different, and both the attester and the verifier know the correct value in advance.

- o Both configuration subscription and dynamic subscription are considered. More specifically:
  - \* Configure subscription is for the important security event stream. For example, it enables the monitoring the important integrity information by using the on-change mode.
  - \* Dynamic subscription is for the normal integrity information, that is, periodically receive those related information during NETCONF Session. The corresponding subscription RPC needs to be established dynamically. This way can reduce unnecessary NETCONF sessions.
- o In addition to the update trigger of on-change, the other possible update trigger may be certain pre-defined events according to [I-D.bryskin-netconf-automation-yang], that is: When these events occur, the specified integrity information is triggered to be sent, which is the relevant event stream plus optional selection filter. The events may include: device startup completion, device upgrade completion, specific attack event, active/standby switchover, line card insertion/removal/switchover, certificate life cycle event (expiration), etc.
- o The attester notification delivery mechanisms thus vary as the above subscription mechanisms of verifier vary.

The following sections describe the above key steps one by one.

### 3.2. Remote Attestation Event Stream Definition

The event streams here refers to various integrity evidence information related to device trustworthiness. The basic event streams may include: software integrity information (including PCR values and system boot logs) of each layer of the trust chain recorded during device booting time, device identity certificates & Attestation Key certificate generation, operating system and application dynamic integrity information (e.g., IMA logs) recorded during device running time.

The event streams are created and managed by the attester. And their formal definition should be conformed to the information model definition like Attestation Evidence or others in [I-D.birkholz-rats-information-model], and the claim data model

definition in [I-D.birkholz-rats-basic-yang-module] with YANG data format, and [I-D.ietf-rats-eat] with COSE data format.

### 3.3. Remote Attestation Subscription Definition

NETCONF pub/sub model provides several suscription types in which appropriate one or more types are choosed and possibly used together to meet the service requirements.

Particularly, periodic subscription is mainly used by the verifier for the general and non-critical information collection, which are not strictly time sensitive and aims for collecting the latest integrity information and checking the possible deviation. In contrary, on-change subscription is basically used to monitor the critical integrity evidence (e.g., integrity values and log files during device booting time, or integrity values of some key service processes). If these integrity values change, notifications are sent immediately.

Besides, both configuration subscription and dynamic subscription are considered. In which, configure subscription is for the important security data stream as it lasts even the NETCONF session is closed. For example, it enables the monitoring of the status of important security event stream by using the on-change mode. On the other hand, dynamic subscription is for the general security event stream, that is, periodically receive those related information during NETCONF Session. The corresponding subscription RPC needs to be established dynamically. This way can reduce unnecessary NETCONF sessions.

Furthermore, certain pre-defined events can be the update trigger too, that is: When these events occur, the specified integrity information is triggered to be sent, which is the relevant event stream plus optional selection filter. The events may include: device startup completion, device upgrade completion, specific attack event, active/standby switchover, line card insertion/removal/switchover, certificate life cycle event (expiration), etc.

### 3.4. Remote Attestation Selection Filters Definition

The selection filters may be applied to the subscription, so that only the event that pass the filter will be distributed out.

A concrete example of selection filter is limiting the delivered event stream to those originated from a specific component with id ("xxxxxxxxxx") of a designated vendor ("xxx-vendor-device").

### 3.5. Remote Attestation Subscription Parameters Handling

Most of the parameters carried in the subscription message are not changed during the remote attestation procedure, like: hash signature algorithm, specified TPM name and so on. Their main goal is to enable the dynamic negotiation with the attester about what information the verifier needs and how to construct them together. A very important point is how to ensure that the nonce carried in every notification message is different, and both the attester and the verifier know the correct value in advance. For this purpose, the basic idea is to ensure that the nonce in two sides of the communication is synchronously changed, and the randomness of the nonce is maintained. Specifically, there may be several ways to do it:

- o Verifier sends a seed with hash algorithm to the attester in the subscription message, and then perform the synchronization operation on both sides.
- o In fact, the nonce does not need to be random every time. As long as the receive endpoint (here for verifier) can identify duplicated packets, other means may be used. For example: The timestamp and increasing count.
- o The RATS TUDA mechanism [I-D.birkholz-rats-tuda] can also be used here to ensure the freshness of information.

### 3.6. Remote Attestation Notification Distribution

To be written.

### 3.7. Summary

Based on the above discussion, this section gives some examples to illustrate the overall application of sub/pub model to remote attestation procedure.

Below is a configured subscription example with on-change update trigger, with specific contents as:

- o There are 3 integrity evidence related event streams as follows: pcr-trust-evidence, bios-log-trust-evidence and ima-log-trust-evidence. The subscribed one is pcr-trust-evidence.
- o The other parameters of the subscription include: pcr-list: {{1, 3, 7}}, tcg-hash-algo-id: TPM\_ALG\_SHA256, nonce-value: 0x564ac291, TPM\_ALG\_ID-value: TPM\_ALG\_ECDSA, pub-key-id: 0x784a22bf, tpms: {"tpm1"}.

- o The selection filter is set as follows: a specific component with id ("xxxxxxxxxx") of a designated vendor ("xxx-vendor-device").

```

<edit-config>
  <subscriptions
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <subscription>
      <id>100</id>
      <stream>pcr-trust-evidence</stream>
      <stream-subtree-filter>
        <xxx-vendor-device
          xmlns="urn:xxx:params:xml:ns:yang:xxx-vendor-device ">
          <device-id>xxxxxxxxxx</device-id>
        </xxx-vendor-device>
      </stream-subtree-filter>
      <pcr-list>
        <pcr>
          <pcr-indices>1</pcr-indices>
          <pcr-indices>3</pcr-indices>
          <pcr-indices>7</pcr-indices>
          <hash-algo>
            <tcg-hash-algo-id>TPM_ALG_SHA256</tcg-hash-algo-id>
          </hash-algo>
        </pcr>
      </pcr-list>
      <nonce-value>0x564ac291</nonce-value>
      <TPM_ALG_ID-value>TPM_ALG_ECDSA</TPM_ALG_ID-value>
      <pub-key-id>0x784a22bf</pub-key-id>
      <tpms>
        <tpm-name>tpm1</tpm-name>
      </tpms>
      <yp:on-change>
        <yp:dampening-period>100</yp:dampening-period>
      </yp:on-change>
    </subscription>
  </subscriptions>
</edit-config>

```

Figure 2: Configured On-change Subscription Message

Below is a dynamic subscription RPC example with periodic update trigger, with specific contents as:

- o There are 3 integrity evidence related event streams as follows: pcr-trust-evidence, bios-log-trust-evidence and ima-log-trust-evidence. The subscribed one is bios-log-trust-evidence.

- o The other parameters of the dynamic subscription include: tpms: {"tpm1"}, last-entry-value: 0xa34568baac79, log-type: bios, pcr-list: {{2, 4, 8}}, tcg-hash-algo-id: TPM\_ALG\_SHA256.
- o The selection filter is set as follows: a specific component with id ("xxxxxxxxxx") of a designated vendor ("xxx-vendor-device").
- o Subscription period: 500 centiseconds.

```

<rpc netconf:message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <stream>bios-log-trust-evidence</stream>
    <stream-subtree-filter>
      <xxx-vendor-device
        xmlns="urn:xxx:params:xml:ns:yang:xxx-vendor-device ">
        <device-id>xxxxxxxxxx</device-id>
      </xxx-vendor-device>
    </stream-subtree-filter>
    <tpms>
      <tpm-name>tpm1</tpm-name>
    </tpms>
    <last-entry-value>0xa34568baac79</last-entry-value>
    <log-type>bios</log-type>
    <pcr-list>
      <pcr>
        <pcr-indices>2</pcr-indices>
        <pcr-indices>4</pcr-indices>
        <pcr-indices>8</pcr-indices>
        <hash-algo>
          <tcg-hash-algo-id>TPM_ALG_SHA256</tcg-hash-algo-id>
        </hash-algo>
      </pcr>
    </pcr-list>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
  </establish-subscription>
</rpc>

```

Figure 3: Dynamic Periodic Subscription Message

Below is a configured subscription RPC example with pre-defined events as the update trigger, with specific contents as:



- o There are 3 integrity evidence related event streams as follows: pcr-trust-evidence, bios-log-trust-evidence and ima-log-trust-evidence. The subscribed one is pcr-trust-evidence.
- o The other parameters of the subscription include: pcr-list: {{1, 3, 7}}, tcg-hash-algo-id: TPM\_ALG\_SHA256, nonce-value: 0x564ac291, TPM\_ALG\_ID-value: TPM\_ALG\_ECDSA, pub-key-id: 0x784a22bf, tpms: {"tpm1"}.
- o The selection filter is set as follows: a specific component with id ("xxxxxxxxxx") of a designated vendor ("xxx-vendor-device").
- o The event which triggers the integrity evidence delivery is defined as: id: 1001, type: master-slave-switchover

Figure 4: Configured Event-triggered Subscription Message

#### 4. The YANG Module for Sub/pub Model Remote Attestation Procedures

##### 4.1. Tree Format

To be written.

##### 4.2. Raw Format

To be written.

#### 5. Security Considerations

To be written

#### 6. Acknowledgements

Thanks to ...

#### 7. IANA Considerations

To be written, possibly.

#### 8. References

##### 8.1. Normative References

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

- [RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <<https://www.rfc-editor.org/info/rfc8639>>.
- [RFC8640] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Dynamic Subscription to YANG Events and Datastores over NETCONF", RFC 8640, DOI 10.17487/RFC8640, September 2019, <<https://www.rfc-editor.org/info/rfc8640>>.
- [RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <<https://www.rfc-editor.org/info/rfc8641>>.

## 8.2. Informative References

- [I-D.birkholz-rats-architecture]  
Birkholz, H., Wiseman, M., Tschofenig, H., and N. Smith, "Remote Attestation Procedures Architecture", draft-birkholz-rats-architecture-02 (work in progress), September 2019.
- [I-D.birkholz-rats-basic-yang-module]  
Birkholz, H., Eckel, M., Bhandari, S., Sulzen, B., Voit, E., Xia, L., Laffey, T., and G. Fedorkow, "YANG Module for Basic Challenge-Response-based Remote Attestation Procedures", draft-birkholz-rats-basic-yang-module-01 (work in progress), July 2019.
- [I-D.birkholz-rats-information-model]  
Birkholz, H. and M. Eckel, "An Information Model for Assertions used in RATS", draft-birkholz-rats-information-model-00 (work in progress), July 2019.
- [I-D.birkholz-rats-reference-interaction-model]  
Birkholz, H. and M. Eckel, "Reference Interaction Model for Challenge-Response-based Remote Attestation", draft-birkholz-rats-reference-interaction-model-01 (work in progress), July 2019.
- [I-D.birkholz-rats-tuda]  
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-rats-tuda-01 (work in progress), September 2019.

[I-D.bryskin-netconf-automation-yang]

Bryskin, I., Liu, X., Clemm, A., Birkholz, H., and T. Zhou, "Generalized Network Control Automation YANG Model", draft-bryskin-netconf-automation-yang-03 (work in progress), July 2019.

[I-D.ietf-rats-eat]

Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-01 (work in progress), July 2019.

[I-D.richardson-rats-usecases]

Richardson, M., Wallace, C., and W. Pan, "Use cases for Remote Attestation common encodings", draft-richardson-rats-usecases-05 (work in progress), October 2019.

#### Authors' Addresses

Liang Xia (Frank)  
Huawei  
101 Software Avenue, Yuhuatai District,  
Nanjing, Jiangsu 210012  
China

Email: frank.xialiang@huawei.com

Wei Pan  
Huawei  
101 Software Avenue, Yuhuatai District  
Nanjing, Jiangsu 210012  
China

Email: william.panwei@huawei.com