Abstract

Remote ATtestation ProcedureS (RATS) architecture facilitates the attestation of device characteristics that, in general, are based on specific trustworthiness qualities intrinsic to a device or service. It includes trusted computing functionality provided by device hardware and software that allows trustworthiness qualities to be asserted and verified as part of, or pre-requisite to, the device’s normal operation. The RATS architecture maps corresponding attestation functions and capabilities to specific RATS Roles. The goal is to enable an appropriate conveyance of evidence about device trustworthiness via network protocols. RATS Roles provide the endpoint context for understanding the various interaction semantics of the attestation lifecycle. The RATS architecture provides the building block concepts, semantics, syntax and framework for interoperable attestation while remaining hardware-agnostic. This flexibility is intended to address a significant variety of use-cases and scenarios involving interoperable attestation. Example usages include, but are not limited to: financial transactions, voting machines, critical safety systems, network equipment health, or trustworthy end-user device management. Existing industry attestation efforts may be helpful toward informing RATS architecture. Such as: Remote Integrity VERification (RIVER), the creation of Entity Attestation Tokens (EAT), software integrity Measurement And ATtestation (MAAT)

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 September 13, 2019.

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. What is Remote Attestation ...................... 4
   1.2. The purpose of RATS Architecture and Terminology .... 4
   1.3. Requirements notation ......................... 4
2. RATS Architecture ...................................... 4
3. Architectural Components ............................... 5
   3.1. RATS Roles .................................. 5
4. RATS Actors ........................................... 6
   4.1. RATS Duties ................................ 7
      4.1.1. Attester Duties ........................... 8
      4.1.2. Verifier Duties ........................... 8
      4.1.3. Claimant Duties ........................... 8
      4.1.4. Relying Party Duties ........................ 8
      4.1.5. RATS Interactions .......................... 9
5. Application of RATS .................................... 10
   5.1. Trust and Trustworthiness ....................... 11
   5.2. Claims and Evidence ............................. 12
   5.3. RATS Information Flows ........................... 12
6. Exemplary Composition of Roles ........................ 13
   6.1. Conveyance of Trusted Claim Sets Validated by Signature . 13
   6.2. Conveyance of Attestation Evidence Appraised by a
1. Introduction

In general, this document provides normative guidance how to use, create or adopt network protocols that facilitate remote attestation procedures. The RATS Architecture anticipates broad deployment contexts that range from IoT to Cloud and Edge ecosystems. The foundation of the RATS architecture is the specification of RATS Roles that can be chained via RATS Interactions and – as a result – may be composed into use-case specific Remote Attestation Procedures. RATS Actors establish an ecosystem neutral context where RATS Roles are hosted and where a variety of Remote Attestation Procedure interactions are defined independent of specific conveyance protocols or message formats. In summary, the goal of the RATS Architecture is to enable interoperable interaction between the RATS Roles. Hence, the RATS Architecture is designed to enable interoperability via well-defined semantics of the information model (attestation assertions/claims), associated with RATS Roles following a conveyance
model (RATS Interactions) that may be used to compose domain-specific remote attestation solutions.

1.1. What is Remote Attestation

Unfortunately, the term Attestation itself is an overloaded term. In consequence, the term Remote Attestation covers a spectrum of meanings. The common denominator encompasses the creation, conveyance, and appraisal of evidence pertaining to the trustworthiness characteristics of the creator of the evidence. In essence, RATS are used to enable the assessment of the trustworthiness of a communication partner.

1.2. The purpose of RATS Architecture and Terminology

To consolidate the utilization of existing and emerging network protocols in the context of RATS, this document provides a detailed definition of Attestation Terminology that enables interoperability between different types of RATS. Specifically, this document illustrates and remediates the impedance mismatch of terms related to Remote Attestation Procedures used in different domains today. As an additional contribution, new terms defined by this document provide a common basis that simplifies future work on RATS in the IETF and beyond.

1.3. Requirements notation

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

2. RATS Architecture

One of the goals of the RATS Architecture is to provide the building blocks – the roles defined by the RATS Architecture – to enable the composition of service-chains/hierarchies and work-flows that can create and appraise evidence about the trustworthiness of devices and services.

The RATS Architecture is based on the use-cases defined in [I-D.richardson-rats-ucases].

The RATS architecture specifies:

- The building blocks to create remote attestation procedures applicable Actors, Roles, Duties, and Interactions,
Mandatory and optional trust relationships between its Roles, that may assume a Root-of-Trust context,

The interaction between Roles that reside on separate Actors and interact via network protocols,

Protocol/message framing that allows for well-defined and opaque payloads,

The means to prove, preserve and convey trust properties, such as identity, varacity, freshness, or provenance, and

Primitives necessary for the construction of interoperable attestation payloads.

3. Architectural Components

The basic architectural components defined in this document are:

- RATS Roles
- RATS Actors
- RATS Duties
- RATS Interactions

The following sub-section define and elaborate on these terms:

3.1. RATS Roles

A Role in the context of usage scenarios for remote attestation procedures is providing a service to other Roles. Roles are building blocks that can be providers and consumers of information. In the RATS architecture, devices or services can take on RATS roles. They are composites of internal functions (RATS Duties) and external functions (RATS Interactions) that facilitate a required (sometimes optional) task in a remote attestation procedure.

The base set of RATS roles is:

Claimant: The producer of trustworthiness assertions pertaining to an Attester; that may or not have a root-of-trust for measurement.

It is not guaranteed that a Verifier Role can appraise the output of a Claimant via reference values (in contrast to the output of an Attester).
Examples of Claimant assertions include: * The hardware, firmware and software components of the Attester. * The manufacturer of Attester components. * The Attester’s current configuration. * The method by which binding of an attester to an RTR. * The identifier(s) available for identifying and authenticating the Attester – e.g. Universal Entity ID (UEID).

Typically, claimant role are taken on by RATS Actors that supply chain entities (SCE). Various assertions (often represented as Claims or Trusted Claims Sets, e.g. [I-D.mandyam-eat] or [I-D.tschafenig-rats-psa-token]).

Attester: The producer of attestation evidence that has a root of trust for reporting (RTR) and implements a conveyance protocol, authenticates using an attestation credential, consumes assertions about itself and presents it to a consumer of evidence (e.g. a relying party or a verifier). Every output of an attester can be appraised via reference values.

Authentication Checker: The consumer of signed assertions such as trusted claim sets or attestation evidence that assesses the trustworthiness or other trust relationships of the information consumed via trusted third parties or external trust authorities, such as a privacy certificate authority. In certain environments, an Authentication Checker can assess a system’s trustworthiness via external trust anchors, implicitly.

Verifier: The consumer of attestation evidence that has a root of trust for verification (RTV), implements conveyance protocols, appraises attestation evidence against reference values or policies, and makes verification results available to relying parties.

Relying Party: The consumer and assessor of verifier or Authentication Checker results for the purpose of improved risk management, operational efficiency, security, privacy (natural or legal person) or safety. The verifier and/or authentication checker roles and the relying party role may be tightly integrated.

4. RATS Actors

RATS Actors may be any entity, such as an user, organization, execution environment, device or service provider, that takes on (implements) one or more RATS Roles and performs RATS Duties and/or RATS Interactions. RATS Interactions occur between RATS Actors. The methods whereby RATS Actors are identified, discovered, and
connectivity established are out-of-scope for this architecture. In contrast, if multiple RATS Roles reside on a single RATS Actor, the definition of RATS Interactions is out-of-scope of the RATS architecture, if no network protocols are required.

Figure 1: RATS Actor-Role Interactions

RATS Actors have the following properties:

* Multiplicity - Multiple instances of RATS Actors that possess the same RATS Roles can exist.
* Decomposability - A singleton RATS Actor possessing multiple RATS Roles can be separated into multiple RATS Actors. RATS Interactions may occur between them.
* Composability - RATS Actors possessing different RATS Roles can be combined into a singleton RATS Actor possessing the union of RATS Roles. RATS Interactions between combined RATS Actors ceases.

Interactions between RATS Roles belonging to the same RATS Actor are generally believed to be uninteresting. Actor operations that apply resiliency, scaling, load balancing or replication are generally believed to be uninteresting.

4.1. RATS Duties

A RATS Role can take on one or more duties. RATS Duties are role-internal functions that do not require interaction with other RATS Roles. In general, and RATS Duties are typically associated with a RATS Role. The list presented in this document is exhaustive. Also, there can be usage scenario where RATS Duties are associated with other RATS Roles than illustrated below:
4.1.1. Attester Duties

- Acquisition or collection of assertions about itself
- Provide or create proof that an assertion is bound to the Attester
- Create Evidence from assertion bundles via roots-of-trust

4.1.2. Verifier Duties

- Acquisition and storage of assertion semantics
- Acquisition and storage of appraisal policies
- Verification of Attester Identity (attestation provenance)
- Comparing assertions or evidence with reference values according to appraisal policies
- Validate authentication information based on public keys, signatures, secrets that are shielded, or secrets that are access restricted via protection profiles

4.1.3. Claimant Duties

- Hardens the device or service that implements the Attester role
- Provisions device identities and/or key material accessible to the Attester role
- Evaluates trustworthiness during manufacturing, supply chain and onboarding
- Produces trustworthiness assertions applicable to the Attestor role
- Embeds trustworthiness assertions about the Attester role in the device or service during manufacturing, supply chain or onboarding

4.1.4. Relying Party Duties

- Evaluate assertions/evidence locally as far as possible
- Compare trust policies to attestation-results based on assertions or evidence
- Enforce policies or create input for risk engines
4.1.5. RATS Interactions

The flow of information between RATS Roles located on RATS Actors compose individual remote attestation procedures. The RATS Architecture provides a set of standard interactions between the RATS Roles defined in this document in order to enable this composability. In this section, common interactions between roles are specified. This list of interactions is not exhaustive, but provides the basis to create various standard RATS.

Every RATS Interaction specified below is based on the information flow between two RATS Roles defined above. Every RATS Interaction is conducted via an Interconnect between corresponding RATS Roles that RATS Actors take on. If more than one RATS Role resides on the same RATS Actor, a network protocol might not be required. If RATS Roles are collapsed into a singular RATS Actor in this way, the method of conveying information is out-of-scope of this document. If network protocols are used to convey corresponding information between RATS Roles (collapsed on a singular RATS Actor or not), the definitions and requirements defined in this document apply.

In essence, an Interconnect is an abstract "distance-less" channel between RATS Actors that can range from General Purpose Input Output (GPIO) interfaces to the Internet.

Attester/Verifier: The most basic RATS interaction is between the creator of evidence (Attester) and its complementary remote attestation service (Verifier). In order to convey evidence (or assertions that are not accompanied by a proof of their validity) this RATS Interaction is required.

Attester/Relying-Party: A Relying Party typically requires external help to either validate authentication information or to appraise evidence presented by an Attester. In most cases, a Relying Party requires a corresponding Verifier to process the assertions/evidence received. In consequence, (a subset of) the information received by an Attester must be relayed securely to a Verifier.

Relying-Party/Verifier: Typically, trusted assertions or evidence are conveyed from an Attester to a Relying Party. In an open ecosystem, such as the Internet, the appraisal of the evidence presented by an Attester provided in order to assess its trustworthiness requires a remote attestation service. Hence, either the RATS roles of Verifier and Relying Party are collapsed and compose a single RATS Actor, or - if they reside on separate RATS Actors - a Relying Party requires appropriate configuration or a discovery/join/rendezvous service to initiate a RATS Interaction with an appropriate and trusted Verifier.
Attestation information originating from an Attester that is relayed via a Relying Party must be protected from replay or relay attacks, accordingly. In a closed ecosystem, trustworthiness with respect to the Attester can be achieved via a simply query to the Verifier. In an open ecosystem, the information conveyed in this interaction can include integrity measurements of every distinguishable software component that has been executed since its last boot cycle.

In the scope of RATS, this interaction encompasses the largest variety of information conveyed.

Claimant/Verifier: The intended operational state an Attester is intended to be in, is defined by the supply chain entities that manufacture and maintain the Attestor. In order to appraise trusted assertions or evidence conveyed by the Attester, every distinguishable system component the Attester is composed of can provide trusted assertions or evidence about its trustworthiness. A corresponding verifier that is tasked with assessing the trustworthiness of the Attester potentially requires a multitude of sources of reference values according to policies and the information provided. As Relying Parties often have to discover an appropriate Verifier, a Verifier has to obtain and potentially store appropriate reference values in order to assess assertions or evidence about trustworthiness.

Claimant/Attester: To enable RATS, trustworthy assertions have to be embedded in an Attester by its manufacturer. In some cases this involves various types of roots of trust. In other cases shielded pre-master secrets in combination with key derivation functions (KDF) provide this binding of trusted information to an Attester. A supply chain entity can embed additional trusted assertions to an Attester. These assertion can also be used to assert the trustworthiness on behalf of a separate RATS Actor or they can originate from an external entity (e.g. a security certification authority).

5. Application of RATS

Attester are typically composite devices (in the case of atomically integrated devices that would result in a composite device with one component) or services. Services are software components - e.g. a daemon, a virtual network function (vnf) or a network security function (nsf) - that can reside on one or more Attester and are not necessarily bound to a specific set of hardware devices.
Relevant decision-factors that influence the composition of RATS Roles on RATS Actors, which result in specific work-flows are (amongst others):

- which RATS Role (or correspondingly, which RATS Actore that is taking on specific RATS roles) is triggering a Remote Attestation Procedure
- which entities are involved in a Remote Attestation Procedure (e.g. the Attester itself, trusted third parties, specific trust anchors, or other sources of assertions)
- the capabilities of the protocols used (e.g. challenge-response based, RESTful, or uni-directional)
- the security requirements and security capabilities of systems in a domain of application
- the risks and corresponding threats that are intended to be mitigated

5.1. Trust and Trustworthiness

[RFC4949] provides definitions that highlight the difference between a "trusted system" and a "trustworthy system". The following definitions exclude the explicit specialization of concepts that are "environmental disruption" as well as "human user and operator errors".

A trusted system in the context of RATS "operates as expected, according to design and policy, doing what is required and not doing other things" [RFC4949]. A trustworthy system is a system "that not only is trusted, but also warrants that trust because the system’s behavior can be validated in some convincing way, such as through formal analysis or code review" [RFC4949].

The goal of RATS is to convey information about system component characteristics, such as integrity or authenticity, that can be appraised in a convincing way.

RATS require trust relationships with third parties that qualify assertions about, for example, origin of data, the manufacturer or the capabilities of a system, or the origination of attestation evidence (attestation provenance). Without trusted authorities (e.g. a certificate authority) it is virtually impossible to assess the level of assurance (or resulting level of confidence, correspondingly) of information produced by RATS. Trusting a system does not make it trustworthy. Assessing trustworthiness requires the
conveyance of evidence that a system is a trustworthy system, which has to originate from the system itself and has to be convincing. If the convincing information is not originating from the system itself, it comprises trusted claim sets and not evidence. In essence, the attestation provenance of attestation evidence is the system that intends to present its trustworthiness in a believable manner.

The essential basis for trust in the information created via RATS are roots of trust.

Roots of trust are defined by the NIST special publication 800-164 draft as "security primitives composed of hardware, firmware and/or software that provide a set of trusted, security-critical functions. They must always behave in an expected manner because their misbehavior cannot be detected. As such, RoTs need to be secured by their design. Hardware RoTs are preferred over software RoTs due to their immutability, smaller attack surface, and more reliable behavior."

If the root of trust involved is a root of trust for measurement (RTM), the producer of information takes on the role of a asserter. An asserter can also make use of a root of trust for integrity (RTI) in order to increase the level of assurance in the assertions produced. If the root of trust involved is a root of trust for reporting (RTR), the producer of information takes on the role of an attester.

5.2. Claims and Evidence

The RATS asserter role produces measurements about the system’s characteristics in the form of signed (sometimes un-signed) claim sets in order to convey information. A secret signing key is required for this procedure, which is typically stored in a shielded location that can be trusted, for example, via a root of trust for storage (RTS).

The RATS attester role produces signed attestation evidence in order to convey information. The secret key required for this procedure is stored in a shielded location that only allows access to that key, if a specific operational state of the system is met. The trust with respect to this origination is based on a root of trust for reporting.

5.3. RATS Information Flows

There are six roles defined in the RATS architecture. iFigure 2 provides a simplified overview of the RATS Roles defined above,
illustrating a general Interconnect in the center that facilitates all RATS Interactions.

<table>
<thead>
<tr>
<th>Attester</th>
<th>Interconnect</th>
<th>Relying Party</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Overall Relationships of Roles in the RATS Architecture

6. Exemplary Composition of Roles

In order to provide an intuitive understanding how the roles used in RATS can be composed into work-flows, this document provides a few example work-flows. Boxes in the following examples that include more than one role are systems that take on more than one role.

6.1. Conveyance of Trusted Claim Sets Validated by Signature

If there is a trust relationship between a trusted third party that can assert that signed claims created by a claimant guarantee a trustworthy origination of claim, the work-flow depicted in Figure 3 can facilitate a trust-based implicit remote attestation procedure. The information conveyed are signed claim sets that are trusted via an authoritative third party. In this work-flow claim emission is triggered by the claimant. Variations based on requests emitted by the relying party can be easily facilitated by the same set of roles.
6.2. Conveyance of Attestation Evidence Appraised by a Verifier

If there is trust in the root of trust for reporting based on the assertions of a trusted third party, the work-flow depicted in Figure 4 can facilitate an evidence-based explicit remote attestation procedure. The information conveyed is signed attestation evidence that is created by the trusted verifier. In this work-flow claims do not necessarily have to be signed and the work-flow is triggered by the attestor that aggregates claims from a root of trust of measurement. Variations based on requests emitted by the verifier can be easily facilitated by the same set of roles.

7. The Scope of RATS

During its evolution, the term Remote Attestation has been used in multiple contexts and multiple scopes and in consequence accumulated various connotations with slightly different semantic meaning.
Correspondingly, Remote Attestation Procedures (RATS) are employed in various usage scenarios and different environments.

In order to better understand and grasp the intent and meaning of specific RATS in the scope of the security area - including the requirements that are addressed by them - this document provides an overview of existing work, its background, and common terminology. As the contribution, from that state-of-the-art a set of terms that provides a stable basis for future work on RATS in the IETF is derived.

In essence, a prerequisite for providing an adequate set of terms and definitions for the RATS architecture is a general understanding and a common definitions of "what" RATS can accomplish "how" RATS can be used.

Please note that this section is still missing various references and is considered "under construction". The majority of definitions is still only originating from IETF work. Future iterations will pull in more complementary definitions from other SDO (e.g. Global Platform, TCG, etc.) and a general structure template to highlight semantic relationships and capable of resolving potential discrepancies will be introduced. A section of context awareness will provide further insight on how Attestation procedures are vital to ongoing work in the IETF (e.g. I2NSF & tokbind). The definitions in the section about RATS are still self-describing in this version. Additional explanatory text will be added to provide more context and coherence.

7.1. The Lying Endpoint Problem

A very prominent goal of RATS is to address the "lying endpoint problem". The lying endpoint problem is characterized as a condition of a Computing Context where the information or behavior embedded, created, relayed, stored, or emitted by the Computing Context is not "correct" according to expectations of the authorized system designers, operators and users. There can be multiple reasons why these expectations are incorrect, either from malicious Activity, unanticipated conditions or accidental means. The observed behavior, nevertheless, appears to be a compromised Computing Context.

Attempts to "scrub" the data or "proxy" control elements implies the existence of a more fundamental trusted endpoint that is operating correctly. Therefore, Remote Attestation - the technology designed to detect and mitigate the "lying endpoint problem" - must be trusted to behave correctly independent of other controls.

Consequently, a "lying endpoint" cannot also be a "trusted system".
Remote Attestation procedures are intended to enable the consumer of information emitted by a Computing Context to assess the validity and integrity of the information transferred. The approach is based, for example, on the assumption that if attestation evidence can be provided in order to prove the integrity of every software instance installed involved in the activity of creating the emitted information in question, the emitted information can be considered valid and integer.

In contrast, such Evidence has to be impossible to create if the software instances used in a Computing Context are compromised. Attestation activities that are intended to create this Evidence therefore also provide guarantees about the validity of the Evidence they can create.

7.1.1. How the RATS Architecture Addresses the Lying Endpoint Problem

RATS imply the involvement of at least two players (roles) who seek to overcome the lying endpoint problem. The Verifier wishes to consume application data supplied by a Computing Context. But before application data is consumed, the Verifier obtains Attestation Evidence about the Computing Context to assess likelihood of poisoned data due to endpoint compromise or failure. Remote Attestation argues that a system’s integrity characteristics should not be believed until rationale for believability is presented to the relying party seeking to interact with the system.

An Interconnect defines an untrusted channel between subject and object wherein the rationale for believability is securely exchanged. The type of interconnect technology could vary widely, ranging from GPIO pins, to a PC peripheral IO bus, to the Internet, to a direct physical connection, to a wireless radio-receiver association, or to a world wide mesh of peers. In other words, virtually every kind communication path could be used as the "Interconnect" in RATS. In fact, a single party could take on all roles at the same time (e.g. Self Encrypting Devices).

Attestation evidence can be thought of as the topics of the exchange that is created the operational primitives of a root of trust for reporting. Evidence may be structured in an interoperable format called claims that may include references to the claimants which are asserting the claims. RATS aims to define "interoperable Remote Attestation" such that evidence can be created and consumed by different ecosystem systems and can be securely exchanged by a broad set of network protocols.
8. RATS Terminology

This document relies on terminology found in [RFC4949]. This document presumes the reader is familiar with the following terms.

- Cryptography
- Entity (System entity)
- Identity
- Object
- Principal
- Proof-of-possession protocol
- Security environment (Environment)
- Security perimeter
- Subject
- Subsystem
- System
- Target-of-Evaluation (TOE)
- Trusted Computing Base (TCB)
- Trusted Platform Module (TPM)
- Trusted (Trustworthy) system
- Verification

Terminology defined by this document is preceded by a dollar sign ($) to distinguish it from terms defined elsewhere and as a way to disambiguate term definition from explanatory text.

Terms defined by this document that are subsequently used by this document are distinguished by capitalizing the first letter of the term (e.g. Term or First_word Second_word).
8.1. Computing Context

This section introduces the term Computing Context in order to specialize the notions of environment and endpoint to terminology that has relevance to trusted computing. Attestation is a discipline of trusted computing.

A Computing Context could refer to a large variety of endpoints. Examples include but are not limited to: the compartmentalization of physical resources, the separation of software instances with different dependencies in dedicated containers, and the nesting of virtual components via hardware-based and software-based solutions. The number of approaches and techniques to construct an endpoint continuously changes with new innovation. Hence, it isn't a goal of this document to define remote attestation for a fixed set of endpoints. Rather, it attempts to define endpoints conceptually and rely on Claims management as a way to clarify the details and specific attributes of conceptual endpoints.

Computing Contexts may be recursive in nature in that it could be composed of a system that is itself a composite of subsystems. In consequence, a system may be composed of other systems that may be further composed of one or more Computing Contexts capable of taking on the RATS roles. The scope and application of these roles can range from:

- Continuous mutual Attestation procedures of every subsystem inside a composite device, to
- Sporadic Remote Attestation of unknown parties via heterogeneous Interconnects.

Analogously, the increasing number of features and functions that constitute components of a device start to blur the lines that are required to categorize each solution and approach precisely. To address this increasingly challenging categorization, the term Computing Context defines the characteristics of the (sub)systems that can take on the role of an Attester and/or the role of a Verifier. This approach is intended to provide a stable basis of definitions for future solutions that continuous to remain viable long-term.

$\text{Computing Context} : \text{An umbrella term that combines the scope of the definitions of endpoint [ref NEA], device [ref 1ar], and thing [ref t2trg], including hardware-based and software-based sub-contexts that constitute independent, isolated and distinguishable slices of a Computing Context created by compartmentalization}$
mechanisms, such as Trusted Execution Environments (TEE), Hardware Security Modules (HSM) or Virtual Network Function (VNF) contexts.


While the semantic relationships highlighted above constitute the fundamental basis to provide a define Computing Context, the following list of object characteristics is intended to improve the application of the term and provide a better understanding of its meaning:


Computing context characteristics provide the following: * An independent environment in regard to executing and running software, * An isolated control plane state (by potentially interacting with other Computing Contexts), * A dedicated management interface by which control plane behavior can be effected, * Unique identification towards reliable disambiguation within a given scope.

Computing context characteristics do not necessarily include a network interface with associated network addresses (as required by the definition of an endpoint) – although it is very likely to have (access to) one.

[Issue: This conclusion could be incorrect] In contrast, a container context is not a distinguishable isolated slice of an information system and therefore is not an independent Computing Context. [more feedback on this statement is required as the capabilities of docker-like functions evolve continuously]

Examples include: a smart phone, a nested virtual machine, a virtualized firewall function running distributed on a cluster of physical and virtual nodes, or a trust-zone.

8.1.2. Computing Context Semantic Relationships

Computing Contexts may relate to other Computing Contexts that are decomposable in a variety of ways.

- Singleton,
- Tuples (e.g. 2-tuple, n-tuple),

o Nested,

o Clustered (homogeneous),

o Grouped (heterogenous).

The scope of Computing Context encompasses a broad spectrum of systems including, but not limited to:

o An information system,

o An object,

o A composition of objects,

o A system component,

o A system sub-component,

o A composition of system sub-components,

o A system entity,

o A composition of system entities.

A Computing Context may be realized in a variety of ways including, but not limited to:

o A process, thread or task as defined by an operating system,

o A privileged operating system task, interrupt handler or event handler,

o A virtual machine,

o A virtual machine monitor,

o A processor mode (e.g. system management mode),

o A co-processor,

o A peripheral device,

o A secure element,

o A trusted execution environment,

o A controller, sensor, actuator, switch, router or gateway,
- An FPGA,
- An ASIC,
- A memory resource,
- A storage resource.

Analogously, a computing sub-context is a decomposition of a Computing Context; a subsystem is a decomposition of a system; a sub-component is a decomposition of a component; and a peer node is a decomposition of a node cluster.

A formal semantic relationship is therefore expressed using an information model that captures interactions, relationships, bindings and interfaces among systems, subsystems, system components, system entities or objects.

[Issue: A tangible relationship to an information model is required here] An information model that richly captures Computing Context semantics is therefore believed to be relevant if not fundamental to Remote Attestation.

8.1.3. Computing Context Identity

The identity of a Computing Context implies there is a binding operation between an identifier and the Computing Context.

$ Computing Context Identity: Computing Context Identity provides the basis for associating attestation Evidence about a particular Computing Context to create believable knowledge about attestation provenance.

Confidence in the identity assurance level [NIST SP-800-63-3] or the assurance levels for identity authentication [RFC4949] is a property of the identifier uniqueness properties and binding operation veracity. Such properties impact the trustworthiness of associated attestation Evidence.

8.2. Remote Attestation Concepts

Attestation Evidence created by RATS is a form of telemetry about a computing environment that enables better security risk management through disclosure of security properties of the environment. Attestation may be performed locally (within the same computing environment) or remotely (between different computing environments). The exchange of attestation evidence can be formalized to include well-defined protocol, message syntax and semantics.
8.3. Core RATS Terminology

$ Attestation: The creation of evidence by the Attester based on measurements or other claimant output.

A form of telemetry involving the delivery of Claims describing various security properties of a Computing Context by an Attester, such that the Claims can be used as Evidence toward convincing a Verifier regarding trustworthiness of the Computing Context.

$ Conveyance: The transfer of Evidence from the Attester to the Verifier.

$ Verification: The appraisal of Evidence by the Verifier who evaluates it against a reference policy. See also RFC4949 [1].

$ Remote Attestation: A procedure involving Attestation, Conveyance and Verification.

8.4. RATS Information Model Terminology

Evidence conveyed to a Verifier by an Attester is structured to facilitate syntactic and semantic interoperability. An information model defines the tag namespaces used to create tag-value pairs containing discrete bits of Evidence.

$ Evidence: A set of Measurements, quality metrics, quality procedures or assurance criteria about a Computing Context’s behavioral, operational and intrinsic characteristics.

$ Claim: Structured Evidence asserted about a Computing Context. It contains metadata that informs regarding the type, class, representation and semantics of Evidence information. A Claim is represented as a name-value pair consisting of a Claim Name and a Claim Value [RFC7519]. In the context of SACM, a Claim is also specialized as an attribute-value pair that is intended to be related to a statement [I-D.ietf-sacm-terminology].

$ Attestable Claim: Structured Evidence including one or more Claims that are asserted by a Claimant (Note: an Attester role doubles as a Claimant role). An Attestable Claim has the following structure:

1. A Claim or Claims.
4. Proof the Claimant intended to make these Claims.

Note: Proofs of Claims assertions may be separated from the Claim itself. For example, a secure transport over which Claims are conveyed where Claimant’s signing key integrity protects the transport payload could be used as proof of Claim assertion. Alternatively, each Claim could be separately signed by a Claimant.

$ Attested (Asserted) Claim: An Attestable Claim where the proof elements are populated.

$ Evidence (Claims) Creation: Instantiation of Attested Claims by a Claimant.

$ Evidence (Claims) Collection: Assembling of Attested Claims by an Attester for the purpose of Conveyance.

$ Verified (Valid) Claim: An Attested Claim where the proof elements have been verified by a Verifier according to a policy that identifies trusted Claimants and/or trusted Evidence values.

8.5. RATS Work-Flow Terminology

This section introduces terms and definitions that are required to illustrate the scope and the granularity of RATS workflows in the domain of security automation. Terms defined in the following sections will be based on this workflow-related definitions.

In general, RATS are composed of iterative activities that can be conducted in intervals. It is neither a generic set of actions nor simply a task, because the actual actions to be conducted by RATS can vary significantly depending on the protocols employed and types of Computing Contexts involved.

$ Activity: A sequence of actions conducted by Computing Contexts that compose a Remote Attestation procedure. The actual composition of actions can vary, depending on the characteristics of the Computing Context they are conducted by/in and the protocols used to utilize an Interconnect. A single Activity provides only a minimal amount of semantic context, e.g. defined by the Activity’s requirements imposed upon the Computing Context, or via the set of actions it is composed of. Example: The Conveyance of cryptographic Evidence or the appraisal of Evidence via imperative guidance.

$ Task: A unit of work to be done or undertaken.
In the scope of RATS, a task is a procedure to be conducted. Example: A Verifier can be tasked with the appraisal of Evidence originating from a specific type of Computing Contexts providing appropriate identities.

$ Action:  The accomplishment of a thing usually over a period of time, in stages, or with the possibility of repetition.

In the scope of RATS, an action is the execution of an operation or function in the scope of an Activity conducted by a Computing Context. A single action provides no semantic context by itself, although it can limit potential semantic contexts of RATS to a specific scope. Example: Signing an existing public key via a specific openssl library, transmitting data, or receiving data are actions.

$ Procedure:  A series of actions that are done in a certain way or order.

In the scope of RATS, a procedure is a composition of activities (sequences of actions) that is intended to create a well specified result with a well established semantic context. Example: The activities of Attestation, Conveyance and Verification compose a Remote Attestation procedure.

8.6. RATS Reference Use Cases

A "lying endpoint" is not trustworthy.

This document provides NNN prominent examples of use cases Attestation procedures are intended to address:

- Verification of the source integrity of a Computing Context via data integrity proofing of installed software instances that are executed, and
- Verification of the identity proofing of a Computing Context.

8.6.1. Use Case A

8.6.2. Use Case B

8.7. RATS Reference Terminology

$ Attestable Computing Context:  A Computing Context where a Claimant is able to create Claims, an Attester is able to Attest those Claims and a Verifier is able to verify the Claims.
$ Attestation Identity: An identity that refers to an Attester.

$ Attestation Identity Credential: A credential used to authenticate an Attestation Identity.

$ Attestation Identity Key (AIK): An Attestation Identity Credential in the form of an asymmetric cryptographic key where the AIK private key is protected by a Computing Context with protection properties that are stronger than the Computing Context about which the AIK attests. A root-of-trust Computing Context normally protects AIK private keys.

$ Claimant Identity: An identity that refers to a Claimant.

$ Claimant Identity Credential: A credential used to authenticate a Claimant Identity.

$ Measurements / Integrity Measurements: Metrics of Computing Context characteristics (i.e. composition, configuration and state) that affect the confidence in the trustworthiness of a Computing Context. Digests of integrity Measurements can be stored in shielded locations (e.g. a PCR of a TPM).

$ Reference Integrity Measurements: Signed Measurements about a Computing Context’s characteristics that are provided by a vendor or manufacturer and are intended to be used as declarative guidance [I-D.ietf-sacm-terminology] (e.g. a signed CoSWID).

$ Root-of-trust: The Computing Context that protects the following where no other Computing Context is expected to provide its Attestation Evidence: + Attestation Evidence. + AIKs. + Code used during the collection and reporting of Attestation Evidence.

$ Root-of-trust-for-measurement (RTM): A trusted Computing Context where a Claimant creates integrity Measurements and other Evidence about a Computing Context where no other Computing Context is expected to provide its Attestation Evidence.

$ Root-of-trust-for-reporting (RTR): A trusted Computing Context where an Attester stages reporting of Claims where no other Computing Context is expected to provide its Attestation Evidence.

$ Root-of-trust-for-storage (RTS): A trusted Computing Context where a Claimant or Attester stores Claims, Evidence, credentials or policies associated with Attestation where no other Computing Context is expected to provide its Attestation Evidence.
8.8. Interpretations of RFC4949 Terminology for Attestation

Assurance: An attribute of an information system that provides grounds for having confidence that the system operates such that the system’s security policy is enforced [RFC4949] (see Trusted System below).

In common criteria, assurance is the basis for the metric level of assurance, which represents the "confidence that a system’s principal security features are reliably implemented".

The NIST Handbook [get ref from 4949] notes that the levels of assurance defined in Common Criteria represent "a degree of confidence, not a true measure of how secure the system actually is. This distinction is necessary because it is extremely difficult—and in many cases, virtually impossible—to know exactly how secure a system is."

Historically, assurance was well-defined in the Orange Book [http://csrc.nist.gov/publications/history/dod85.pdf] as "guaranteeing or providing confidence that the security policy has been implemented correctly and that the protection-relevant elements of the system do, indeed, accurately mediate and enforce the intent of that policy. By extension, assurance must include a guarantee that the trusted portion of the system works only as intended."

Confidence: The definition of correctness integrity in [RFC4949] notes that "source integrity refers to confidence in data values". Hence, confidence in an Attestation procedure is referring to the degree of trustworthiness of an Attestation Activity that produces Evidence (Attester), of an Conveyance Activity that transfers Evidence (interconnect), and of a Verification Activity that appraises Evidence (Verifier), in respect to correctness integrity.

Correctness: The property of a system that is guaranteed as the result of formal Verification activities.
Correctness integrity: The property that the information represented by data is accurate and consistent.

Data Integrity: (a) The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner. (See: data integrity service. Compare: correctness integrity, source integrity.)

(b) The property that information has not been modified or destroyed in an unauthorized manner.

Entity: A principal, Subject, relying party or stake holder in an Attestation ecosystem.

Identity: The set of attributes that distinguishes a principal.

Identifier: The set of attributes that distinguishes an object.

Identity Proofing: A vetting process that verifies the information used to establish the identity of a system entity.

(Information) System: An organized assembly of computing and communication resources and procedures - i.e., equipment and services, together with their supporting infrastructure, facilities, and personnel - that create, collect, record, process, store, transport, retrieve, display, disseminate, control, or dispose of information to accomplish a specified set of functions.

Object: A system component that contains or receives information.

Source Integrity: The property that data is trustworthy (i.e., worthy of reliance or trust), based on the trustworthiness of its sources and the trustworthiness of any procedures used for handling data in the system.

Subject: A Computing Context acting in accordance with the interests of a principal.

Subsystem: A collection of related system components that together perform a system function or deliver a system service.

System Component: An instance of a system resource that (a) forms a physical or logical part of the system, (b) has specified functions and interfaces, and (c) is extant (e.g., by policies or specifications) outside of other parts of the system. (See: subsystem.)
An identifiable and self-contained part of a Target-of-Evaluation.

Token: A data structure suitable for containing Claims.

Trusted (Trustworthy) System: A system that operates as expected, according to design and policy, doing what is required - despite environmental disruption, human user and operator errors, and attacks by hostile parties - and not doing other things.

Verification: (a) The process of examining information to establish the truth of a claimed fact or value.

(b) The process of comparing two levels of system specification for proper correspondence, such as comparing a security model with a top-level specification, a top-level specification with source code, or source code with object code.

8.9. Building Block Vocabulary (Not in RFC4949)

[working title, pulled from various sources, vital]

Attribute: TBD

Characteristic: TBD

Context: TBD

Endpoint: TBD

Environment: TBD

Manifest: TBD

Telemetry: An automated communications process by which data, readings, Measurements and Evidence are collected at remote points and transmitted to receiving equipment for monitoring and analysis. Derived from the Greek roots tele = remote, and metron = measure.

9. IANA considerations

This document will include requests to IANA:

- first item
- second item
10. Security Considerations
   There are always some.

11. Acknowledgements
   Maybe.

12. Change Log
   No changes yet.

13. References

13.1. Normative References


13.2. Informative References

   [I-D.ietf-sacm-terminology]

   [I-D.mandyam-eat]

   [I-D.richardson-rats-usecases]
   Richardson, M., "Use cases for Remote Attestation common encodings", draft-richardson-rats-usecases-00 (work in progress), March 2019.
13.3. URIs


Authors’ Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Monty Wiseman
GE Global Research
USA

Email: monty.wiseman@ge.com

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

Email: hannes.tschofenig@gmx.net
The Entity Attestation Token (EAT)
draft-mandyam-eat-00

Abstract

An attestation format based on concise binary object representation (CBOR) is proposed that is suitable for inclusion in a CBOR Web Token (CWT), know as the Entity Attestation Token (EAT). The associated data can be used by a relying party to assess the security state of a remote device or module.

Contributing

TBD

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 November 20, 2018.

Copyright Notice

Copyright (c) 2018 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.  Entity Overview ....................................... 4
   1.2.  Use of CBOR and COSE ................................. 4
   1.3.  EAT Operating Models ................................. 5
   1.4.  What is Not Standardized ............................... 6
      1.4.1.  Transmission Protocol ........................... 6
      1.4.2.  Signing Scheme ................................ 6
   1.5.  Terminology ........................................... 7
   2.  The Claims .................................................. 8
       3.1.  Universal Entity ID (UEID) Claim ................... 8
       3.2.  Origination (origination) Claims ................... 10
       3.3.  OEM identification by IEEE OUI .................... 10
       3.4.  Security Level (seclevel) Claim .................. 11
       3.5.  Nonce (nonce) Claim ................................ 12
       3.6.  Secure Boot and Debug Enable State Claims .......... 12
          3.6.1.  Secure Boot Enabled (secbootenabled) Claim .... 12
          3.6.2.  Debug Disabled (debugdisabled) Claim .......... 12
          3.6.3.  Debug Disabled Since Boot (debugdisabledsinceboot) Claim ........................................ 12
          3.6.4.  Debug Permanent Disable (debugpermanentdisable) Claim ................................................ 12
          3.6.5.  Debug Full Permanent Disable (debugfullpermanentdisable) Claim ........................................ 13
       3.7.  Location (loc) Claim .................................. 13
          3.7.1.  lat (latitude) claim ............................... 13
          3.7.2.  long (longitude) claim ........................... 13
          3.7.3.  alt (altitude) claim ............................. 13
          3.7.4.  acc (accuracy) claim ............................. 13
          3.7.5.  altacc (altitude accuracy) claim ................ 14
          3.7.6.  heading claim .................................... 14
          3.7.7.  speed claim ...................................... 14
       3.8.  ts (timestamp) claim .................................. 14
       3.9.  age claim .............................................. 14
       3.10. uptime claim .......................................... 14
       3.11. The submods Claim .................................... 15
          3.11.1.  The submod_name Claim .......................... 15
          3.11.2.  Nested EATs, the eat Claim .................... 15
   4.  IANA Considerations ........................................ 15
       4.1.  Reuse of CBOR Web Token (CWT) Claims Registry ... 15
          4.1.1.  Claims Registered by This Document ............ 16

1. Introduction

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

Remote attestation is a fundamental service that can underly other protocols and services that need to know about the trustworthiness of the device before proceeding. One good example is biometric authentication where the biometric matching is done on the device. The relying party needs to know that the device is one that is known to do biometric matching correctly. Another example is content protection where the relying party wants to know the device will protect the data. This generalizes on to corporate enterprises that might want to know that a device is trustworthy before allowing corporate data to be accessed by it.

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

- Proof of the make and model of the device hardware (HW)
- Proof of the make and model of the device processor, particularly for security oriented chips
- Measurement of the software (SW) running on the device
- Configuration and state of the device
- Environmental characteristics of the device such as its GPS location
The required data format should be general purpose and extensible so that it can work across many use cases. This is why CBOR (see [RFC7049]) is chosen as the format -- it already supports a rich set of data types, and is both expressive and extensible. It translates well to JSON for good interoperation with web technology. It is compact and can work on very small IoT device. The format proposed here is small enough that a limited version can be implemented in pure hardware gates with no software at all. Moreover, the attestation data is defined in the form of claims that is the same as CBOR Web Token (CWT, see [RFC8392]). This is the motivation for defining the Entity Attestation Token, i.e. EAT.

1.1. Entity Overview

An "entity" can be any device or device subassembly ("submodule") that can generate its own attestation in the form of an EAT. The attestation should be cryptographically verifiable by the EAT consumer. An EAT at the device-level can be composed of several submodule EAT’s. It is assumed that any entity that can create an EAT does so by means of a dedicated root-of-trust (RoT).

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

1.2. Use of CBOR and COSE

Fundamentally this attestation format is a verifiable data format. It is a collection of data items that can be signed by an attestation key, hashed, and/or encrypted. As per Section 7 of [RFC8392], the verification method is in the CWT using the CBOR Object Signing and Encryption (COSE) methodology (see [RFC8152]).

In addition, the reported attestation data could be determined within the secure operating environment or written to it from an external and presumably less trusted entity on the device. In either case, the source of the reported data must be identifiable by the relying party.
This attestation format is a single relatively simple signed message. It is designed to be incorporated into many other protocols and many other transports. It is also designed such that other SW and apps can add their own data to the message such that it is also attested.

1.3. EAT Operating Models

At least the following three participants exist in all EAT operating models. Some operating models have additional participants.

The Entity. This is the phone, the IoT device, the sensor, the sub-assembly or such that the attestation provides information about.

The Manufacturer. The company that made the entity. This may be a chip vendor, a circuit board module vendor or a vendor of finished consumer products.

The Relying Party. The server, service or company that makes use of the information in the EAT about the entity.

In all operating models, the manufacturer provisions some secret attestation key material (AKM) into the entity during manufacturing. This might be during the manufacturer of a chip at a fabrication facility (fab) or during final assembly of a consumer product or any time in between. This attestation key material is used for signing EATs.

In all operating models, hardware and/or software on the entity create an EAT of the format described in this document. The EAT is always signed by the attestation key material provisioned by the manufacturer.

In all operating models, the relying party must end up knowing that the signature on the EAT is valid and consistent with data from claims in the EAT. This can happen in many different ways. Here are some examples.

- The EAT is transmitted to the relying party. The relying party gets corresponding key material (e.g. a root certificate) from the manufacturer. The relying party performs the verification.

- The EAT is transmitted to the relying party. The relying party transmits the EAT to a verification service offered by the manufacturer. The server returns the validated claims.

- The EAT is transmitted directly to a verification service, perhaps operated by the manufacturer or perhaps by another party. It verifies the EAT and makes the validated claims available to the
relying party. It may even modify the claims in some way and re-sign the EAT (with a different signing key).

This standard supports all these operating models and does not prefer one over the other. It is important to support this variety of operating models to generaly facilitate deployment and to allow for some special scenarios. One special scenario has a validation service that is monetized, most likely by the manufacturer. In another, a privacy proxy service processes the EAT before it is transmitted to the relying party. In yet another, symmetric key material is used for signing. In this case the manufacturer should perform the verification, because any release of the key material would enable a participant other than the entity to create valid signed EATs.

1.4. What is Not Standardized

1.4.1. Transmission Protocol

EATs may be transmitted by any protocol. For example, they might be added in extension fields of other protocols, bundled into an HTTP header, or just transmitted as files. This flexibility is intentional to allow broader adoption. This flexibility is possible because EAT’s are self-secured with signing (and possibly additionally with encryption and anti-replay). The transmission protocol is not required to fulfill any additional security requirements.

For certain devices, a direct connection may not exist between the EAT-producing device and the Relying Party. In such cases, the EAT should be protected against malicious access. The use of COSE allows for signing and encryption of the EAT. Therefore even if the EAT is conveyed through intermediaries between the device and Relying Party, such intermediaries cannot easily modify the EAT payload or alter the signature.

1.4.2. Signing Scheme

The term "signing scheme" is used to refer to the system that includes end-end process of establishing signing attestation key material in the entity, signing the EAT, and verifying it. This might involve key IDs and X.509 certificate chains or something similar but different. The term "signing algorithm" refers just to the algorithm ID in the COSE signing structure. No particular signing algorithm or signing scheme is required by this standard.

There are three main implementation issues driving this. First, secure non-volatile storage space in the entity for the attestation
key material may be highly limited, perhaps to only a few hundred bits, on some small IoT chips. Second, the factory cost of provisioning key material in each chip or device may be high, with even millisecond delays adding to the cost of a chip. Third, privacy-preserving signing schemes like ECDAA (Elliptic Curve Direct Anonymous Attestation) are complex and not suitable for all use cases.

Eventually some form of standardization of the signing scheme may be required. This might come in the form of another standard that adds to this document, or when there is clear convergence on a small number of signing schemes this standard can be updated.

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], COSE [RFC8152], and CWT [RFC8392].

StringOrURI. The "StringOrURI" term in this specification has the same meaning and processing rules as the JWT "StringOrURI" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR text string instead of a JSON text string.

NumericDate. The "NumericDate" term in this specification has the same meaning and processing rules as the JWT "NumericDate" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR numeric date (from Section 2.4.1 of [RFC7049]) instead of a JSON number. The encoding is modified so that the leading tag 1 (epoch-based date/time) MUST be omitted.

Claim Name. The human-readable name used to identify a claim.

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

Claim Value. The CBOR map value representing the value of the claim.

CWT Claims Set. The CBOR map that contains the claims conveyed by the CWT.

FloatOrNumber. The "FloatOrNumber" term in this specification is the type of a claim that is either a CBOR positive integer, negative integer or floating point number.
Attestation Key Material (AKM). The key material used to sign the EAT token. If it is done symmetrically with HMAC, then this is a simple symmetric key. If it is done with ECC, such as an IEEE DevID [IDevID], then this is the private part of the EC key pair. If ECDAA is used, (e.g., as used by Ehnaced Privacy ID, i.e. EPID) then it is the key material needed for ECDAA.

3. The Claims

3.1. Universal Entity ID (UEID) Claim

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

It is identified by Claim Key X (X is TBD).

UEID’s must be universally and globally unique across manufacturers and countries. UEIDs must also be unique across protocols and systems, as tokens are intended to be embedded in many different protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries. should have the same UEID (if they are not global and universal in this way then relying parties receiving them will have to track other characteristics of the device to keep devices distinct between manufacturers).

The UEID should be permanent. It should never change for a given device / entity. In addition, it should not be reprogrammable.

UEID’s are binary byte-strings (resulting in a smaller size than text strings). When handled in text-based protocols, they should be base-64 encoded.

UEID’s are variable length with a maximum size of 33 bytes (1 type byte and 256 bits). A receivers of a token with UEIDs may reject the token if a UEID is larger than 33 bytes.

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

A UEID is a byte string. From the consumer’s view (the rely party) it is opaque with no bytes having any special meaning.
When the entity constructs the UEID, the first byte is a type and the following bytes the ID for that type. Several types are allowed to accommodate different industries and different manufacturing processes and to give options to avoid paying fees for certain types of manufacturer registrations.

<table>
<thead>
<tr>
<th>Type Byte</th>
<th>Type Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>GUID</td>
<td>This is a 128 to 256 bit random number generated once and stored in the device. The GUID may be constructed from various identifiers on the device using a hash function or it may be just the raw random number. In any case, the random number must have entropy of at least 128 bits as this is what gives the global</td>
</tr>
<tr>
<td>0x02</td>
<td>IEEE EUI</td>
<td>This makes use of the IEEE company identification registry. An EUI is made up of an OUI and OUI-36 or a CID, different registered company identifiers, and some unique per-device identifier. EUIs are often the same as or similar to MAC addresses. (Note that while devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) TODO: normative references to IEEE.</td>
</tr>
<tr>
<td>0x03</td>
<td>IMEI</td>
<td>TODO: figure how to specify IMEIs</td>
</tr>
</tbody>
</table>

Table 1: UEID Composition Types

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

- UEIDs types may vary freely from one manufacturer to the next.
- New types of UEIDs may be created. For example a type 0x04 UEID may be created based on some other manufacturer registration scheme.
- Device manufacturers are allowed to change from one type of UEID to another anytime they want. For example they may find they can optimize their manufacturing by switching from type 0x01 to type
0x02 or vice versa. The main requirement on the manufacturer is that UEIDs be universally unique.

3.2. Origination (origination) Claims

This claim describes the parts of the device or entity that are creating the EAT. Often it will be tied back to the device or chip manufacturer. The following table gives some examples:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme-TEE</td>
<td>The EATs are generated in the TEE authored and configured by &quot;Acme&quot;</td>
</tr>
<tr>
<td>Acme-TPM</td>
<td>The EATs are generated in a TPM manufactured by &quot;Acme&quot;</td>
</tr>
<tr>
<td>Acme-Linux-Kernel</td>
<td>The EATs are generated in a Linux kernel configured and shipped by &quot;Acme&quot;</td>
</tr>
<tr>
<td>Acme-TA</td>
<td>The EATs are generated in a Trusted Application (TA) authored by &quot;Acme&quot;</td>
</tr>
</tbody>
</table>

The claim is represented by Claim Key X+15. It is type StringOrURI.

TODO: consider a more structure approach where the name and the URI and other are in separate fields.

TODO: This needs refinement. It is somewhat parallel to issuer claim in CWT in that it describes the authority that created the token.

3.3. OEM identification by IEEE OUI

This claim identifies a device OEM by the IEEE OUI. Reference TBD. It is a byte string representing the OUI in binary form in network byte order (TODO: confirm details).

Companies that have more than one IEEE OUI registered with IEEE should pick one and prefer that for all their devices.

Note that the OUI is in common use as a part of MAC Address. This claim is only the first bits of the MAC address that identify the manufacturer. The IEEE maintains a registry for these in which many companies participate. This claim is represented by Claim Key TBD.
3.4. Security Level (seclevel) Claim

EATs have a claim that roughly characterizes the device / entities ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is done by roughly defining four security levels as described below. This is similar to the security levels defined in the Metadata Service defined by the Fast Identity Online (FIDO) Alliance (TODO: reference).

These claims describe security environment and countermeasures available on the end-entity / client device where the attestation key reside and the claims originate.

This claim is identified by Claim Key X+2. The value is an integer between 1 and 4 as defined below.

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 should not be general-purpose operating environments that host features such as app download systems, web browsers and complex productivity applications. It is akin to the Secure Restricted level (see below) without the security orientation. Examples include a WiFi subsystem, an IoT camera, or sensor device.

3 - Secure Restricted  Entities at this level must meet the criteria defined by FIDO Allowed Restricted Operating Environments (TODO: reference). Examples include TEE’s and schemes using virtualization-based security. Like the FIDO security goal, security at this level is aimed at defending well against large-scale network / remote attacks against the device.

4 - Hardware  Entities at this level must include substantial defense against physical or electrical attacks against the device itself. It is assumed any potential attacker has captured the device and can disassemble it. Example include TPMs and Secure Elements.

This claim is not intended as a replacement for a proper end-device security certification schemes such as those based on FIPS (TODO: reference) or those based on Common Criterion (TODO: reference). The claim made here is solely a self-claim made by the Entity Originator.
3.5. Nonce (nonce) Claim

The "nonce" (Nonce) claim represents a random value that can be used to avoid replay attacks. This would be ideally generated by the CWT consumer. This value is intended to be a CWT companion claim to the existing JWT claim **_IANAJWT_** (TODO: fix this reference). The nonce claim is identified by Claim Key X+3.

3.6. Secure Boot and Debug Enable State Claims

3.6.1. Secure Boot Enabled (secbootenabled) Claim

The "secbootenabled" (Secure Boot Enabled) claim represents a boolean value that indicates whether secure boot is enabled either for an entire device or an individual submodule. If it appears at the device level, then this means that secure boot is enabled for all submodules. Secure boot enablement allows a secure boot loader to authenticate software running either in a device or a submodule prior allowing execution. This claim is identified by Claim Key X+4.

3.6.2. Debug Disabled (debugdisabled) Claim

The "debugdisabled" (Debug Disabled) claim represents a boolean value that indicates whether debug capabilities are disabled for an entity (i.e. value of ‘true’). Debug disablement is considered a prerequisite before an entity is considered operational. This claim is identified by Claim Key X+5.

3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim

The "debugdisabledsinceboot" (Debug Disabled Since Boot) claim represents a boolean value that indicates whether debug capabilities for the entity were not disabled in any way since boot (i.e. value of ‘true’). This claim is identified by Claim Key X+6.

3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim

The "debugpermanentdisable" (Debug Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of ‘true’). This value can be set to ‘true’ also if only the manufacturer is allowed to enabled debug, but the end user is not. This claim is identified by Claim Key X+7.
3.6.5.  Debug Full Permanent Disable (debugfullpermanentdisable) Claim

The "debugfullpermanentdisable" (Debug Full Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can only be set to 'true' if no party can enable debug capabilities for the entity. Often this is implemented by blowing a fuse on a chip as fuses cannot be restored once blown. This claim is identified by Claim Key X+8.

3.7.  Location (loc) Claim

The "loc" (location) claim is a CBOR-formatted object that describes the location of the device entity from which the attestation originates. It is identified by Claim Key X+10. It is comprised of an array of additional subclaims that represent the actual location coordinates (latitude, longitude and altitude). The location coordinate claims are consistent with the WGS84 coordinate system [WGS84]. In addition, a subclaim providing the estimated accuracy of the location measurement is defined.

3.7.1.  lat (latitude) claim

The "lat" (latitude) claim contains the value of the device location corresponding to its latitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+11.

3.7.2.  long (longitude) claim

The "long" (longitude) claim contains the value of the device location corresponding to its longitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+12.

3.7.3.  alt (altitude) claim

The "alt" (altitude) claim contains the value of the device location corresponding to its altitude coordinate (if available). It is of data type FloatOrNumber and identified by Claim Key X+13.

3.7.4.  acc (accuracy) claim

The "acc" (accuracy) claim contains a value that describes the location accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+14.
3.7.5. altacc (altitude accuracy) claim

The "altacc" (altitude accuracy) claim contains a value that describes the altitude accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+15.

3.7.6. heading claim

The "heading" claim contains a value that describes direction of motion for the entity. Its value is specified in degrees, between 0 and 360. It is of data type FloatOrNumber and identified by Claim Key X+16.

3.7.7. speed claim

The "speed" claim contains a value that describes the velocity of the entity in the horizontal direction. Its value is specified in meters/second and must be non-negative. It is of data type FloatOrNumber and identified by Claim Key X+17.

3.8. ts (timestamp) claim

The "ts" (timestamp) claim contains a timestamp derived using the same time reference as is used to generate an "iat" claim (see Section 3.1.6 of [RFC8392]). It is of the same type as "iat" (integer or floating-point), and is identified by Claim Key X+18. It is meant to designate the time at which a measurement was taken, when a location was obtained, or when a token was actually transmitted. The timestamp would be included as a subclaim under the "submod" or "loc" claims (in addition to the existing respective subclaims), or at the device level.

3.9. age claim

The "age" claim contains a value that represents the number of seconds that have elapsed since the token was created, measurement was made, or location was obtained. Typical attestable values are sent as soon as they are obtained. However in the case that such a value is buffered and sent at a later time and a sufficiently accurate time reference is unavailable for creation of a timestamp, then the age claim is provided. It is identified by Claim Key X+19.

3.10. uptime claim

The "uptime" claim contains a value that represents the number of seconds that have elapsed since the entity or submod was last booted. It is identified by Claim Key X+20.
3.11. The submods Claim

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

The claims for each these can be grouped together in a submodule.

Specifically, the "submods" claim is an array. Each item in the array is a CBOR map containing all the claims for a particular submodule. It is identified by Claim Key X+22.

The security level of the submod is assumed to be at the same level as the main entity unless there is a security level claim in that submodule indicating otherwise. The security level of a submodule can never be higher (more secure) than the security level of the EAT it is a part of.

3.11.1. The submod_name Claim

Each submodule should have a submod_name claim that is descriptive name. This name should be the CBOR txt type.

3.11.2. Nested EATs, the eat Claim

It is allowed for one EAT to be embedded in another. This is for complex devices that have more than one subsystem capable of generating an EAT. Typically one will be the device-wide EAT that is low to medium security and another from a Secure Element or similar that is high security.

The contents of the "eat" claim must be a fully signed, optionally encrypted, EAT token. It is identified by Claim Key X+23.

4. IANA Considerations

4.1. Reuse of CBOR Web Token (CWT) Claims Registry

Claims defined for EAT are compatible with those of CWT so the CWT Claims Registry is re used. New new IANA registry is created. All EAT claims should be registered in the CWT Claims Registry.
4.1.1. Claims Registered by This Document

- Claim Name: UEID
- Claim Description: The Universal Entity ID
- JWT Claim Name: N/A
- Claim Key: X
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): *this document*

TODO: add the rest of the claims in here

4.2. EAT CBOR Tag Registration

How an EAT consumer determines whether received CBOR-formatted data actually represents a valid EAT is application-dependent, much like a CWT. For instance, a specific MIME type associated with the EAT such as "application/eat" could be sufficient for identification of the EAT. Note however that EAT’s can include other EAT’s (e.g. a device EAT comprised of several submodule EAT’s). In this case, a CBOR tag dedicated to the EAT will be required at least for the submodule EAT’s and the tag must be a valid CBOR tag. In other words - the EAT CBOR tag can optionally prefix a device-level EAT, but a EAT CBOR tag must always prefix a submodule EAT. The proposed EAT CBOR tag is 71.

4.2.1. Tag Registered by This Document

- CBOR Tag: 71
- Data Item: Entity Attestation Token (EAT)
- Semantics: Entity Attestation Token (CWT), as defined in *this_doc*
- Reference: *this_doc*
- Point of Contact: Giridhar Mandyam, mandyam@qti.qualcomm.com
5. 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.

5.1. UEID Privacy Considerations

A UEID is usually not privacy preserving. Any set of relying parties that receives tokens that happen to be from a single device will be able to know the tokens are all from the same device and be able to track the device. Thus, in many usage situations ueid violates governmental privacy regulation. In other usage situations 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.

There are several strategies that can be used to still be able to put UEID’s in tokens:

- The device obtains explicit permission from the user of the device to use the UEID. 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.

- The UEID is used only in a particular context or particular use case. It is used only by one relying party.

- The device authenticates the relying party and generates a derive UEID just for that particular relying party. For example, the relying party could prove their identity cryptographically to the device, then the device generates a UEID just for that relying party by hashing a proofed relying party ID with the main device UEID.

Note that some of these privacy preservation strategies result in multiple UEIDs per device. Each UEID is used in a different context, use case or system on the device. However, from the view of the relying party, there is just one UEID and it is still globally universal across manufacturers.
6. Security Considerations

TODO: Perhaps this can be the same as CWT / COSE, but not sure yet because it involves so much entity / device security that those do not.

7. References

7.1. Normative References


7.2. Informative References


Appendix A. Examples

A.1. Very Simple EAT

This is shown in CBOR diagnostic form. Only the payload signed by COSE is shown.

```
{
  / nonce /  11:h'948f8860d13a463e8e',
  / UEID /  8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled /  13:true,
  / debugpermanentdisable / 15:true,
  / ts /  21:1526542894,
}
```

A.2. Example with Submodules, Nesting and Security Levels

```
{
  / nonce /  11:h'948f8860d13a463e8e',
  / UEID /  8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled /  13:true,
  / debugpermanentdisable / 15:true,
  / ts /  21:1526542894,
  / seclevel /  10:3, / secure restricted OS /
  / submods /  30:
    [  
      / 1st submod, an Android Application /  
        / submod_name /  30:'Android App "Foo"',
        / seclevel /  10:1, / unrestricted /
        / app data / -70000:'text string'
    ],
    / 2nd submod, A nested EAT from a secure element /  
      / submod_name /  30:'Secure Element EAT',
      / eat /  31:71( 18(  
        / an embedded EAT /  [ /...COSE_Sign1 bytes with payload.../ ]
      )
    }
    / 3rd submod, information about Linux Android /  
      / submod_name/  30:'Linux Android',
      / seclevel /  10:1, / unrestricted /
      / custom - release / -80000:'8.0.0',
      / custom - version / -80001:'4.9.51+'
    }
  }
```

Authors’ Addresses

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Laurence Lundblade
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 658 3584
EMail: llundbla@qti.qualcomm.com

Miguel Ballesteros
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 4299
EMail: mballest@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
The Entity Attestation Token (EAT)
draft-mandyam-eat-01

Abstract

An attestation format based on concise binary object representation (CBOR) is proposed that is suitable for inclusion in a CBOR Web Token (CWT), know as the Entity Attestation Token (EAT). The associated data can be used by a relying party to assess the security state of a remote device or module.

Contributing TBD

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 23, 2019.

Copyright Notice

Copyright (c) 2018 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. Entity Overview ....................................... 4
   1.2. Use of CBOR and COSE ................................. 5
   1.3. EAT Operating Models ................................. 5
   1.4. What is Not Standardized .............................. 6
      1.4.1. Transmission Protocol .......................... 6
      1.4.2. Signing Scheme ................................ 7
   2. Terminology .............................................. 7
   3. The Claims .............................................. 8
      3.1. Universal Entity ID (UEID) Claim .................. 8
      3.2. Origination (origination) Claims .................. 10
      3.3. OEM identification by IEEE OUI .................... 10
      3.4. Security Level (seclevel) Claim .................. 11
      3.5. Nonce (nonce) Claim ................................ 12
      3.6. Secure Boot and Debug Enable State Claims ....... 12
         3.6.1. Secure Boot Enabled (secbootenabled) Claim .. 12
         3.6.2. Debug Disabled (debugdisabled) Claim ........ 12
         3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim ........................................ 12
         3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim 12
         3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim 13
      3.7. Location (loc) Claim ................................ 13
         3.7.1. lat (latitude) claim ............................ 13
         3.7.2. long (longitude) claim .......................... 13
         3.7.3. alt (altitude) claim ............................ 13
         3.7.4. acc (accuracy) claim ............................ 13
         3.7.5. altacc (altitude accuracy) claim ............... 14
         3.7.6. heading claim .................................. 14
         3.7.7. speed claim .................................... 14
      3.8. ts (timestamp) claim .................................. 14
      3.9. age claim ............................................ 14
      3.10. uptime claim ......................................... 14
      3.11. The submods Claim .................................. 15
         3.11.1. The submod_name Claim ........................ 15
         3.11.2. Nested EATs, the eat Claim .................. 15
Remote device attestation is fundamental service that allows a remote device such as a mobile phone, an Internet-of-Things (IoT) device, or other endpoint to prove itself to a relying party, a server or a service. This allows the relying party to know some characteristics about the device and decide whether it trusts the device.

Remote attestation is a fundamental service that can underlie other protocols and services that need to know about the trustworthiness of the device before proceeding. One good example is biometric authentication where the biometric matching is done on the device. The relying party needs to know that the device is one that is known to do biometric matching correctly. Another example is content protection where the relying party wants to know the device will protect the data. This generalizes on to corporate enterprises that might want to know that a device is trustworthy before allowing corporate data to be accessed by it.

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

- Proof of the make and model of the device hardware (HW)
o Proof of the make and model of the device processor, particularly for security oriented chips

o Measurement of the software (SW) running on the device

o Configuration and state of the device

o Environmental characteristics of the device such as its GPS location

The required data format should be general purpose and extensible so that it can work across many use cases. This is why CBOR (see [RFC7049]) was chosen as the format -- it already supports a rich set of data types, and is both expressive and extensible. It translates well to JSON for good interoperation with web technology. It is compact and can work on very small IoT device. The format proposed here is small enough that a limited version can be implemented in pure hardware gates with no software at all. Moreover, the attestation data is defined in the form of claims that is the same as CBOR Web Token (CWT, see [RFC8392]). This is the motivation for defining the Entity Attestation Token, i.e. EAT.

1.1. Entity Overview

An "entity" can be any device or device subassembly ("submodule") that can generate its own attestation in the form of an EAT. The attestation should be cryptographically verifiable by the EAT consumer. An EAT at the device-level can be composed of several submodule EAT’s. It is assumed that any entity that can create an EAT does so by means of a dedicated root-of-trust (RoT).

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

Fundamentally this attestation format is a verifiable data format. It is a collection of data items that can be signed by an attestation key, hashed, and/or encrypted. As per Section 7 of [RFC8392], the verification method is in the CWT using the CBOR Object Signing and Encryption (COSE) methodology (see [RFC8152]).

In addition, the reported attestation data could be determined within the secure operating environment or written to it from an external and presumably less trusted entity on the device. In either case, the source of the reported data must be identifiable by the relying party.

This attestation format is a single relatively simple signed message. It is designed to be incorporated into many other protocols and many other transports. It is also designed such that other SW and apps can add their own data to the message such that it is also attested.

1.3. EAT Operating Models

At least the following three participants exist in all EAT operating models. Some operating models have additional participants.

The Entity. This is the phone, the IoT device, the sensor, the sub-assembly or such that the attestation provides information about.

The Manufacturer. The company that made the entity. This may be a chip vendor, a circuit board module vendor or a vendor of finished consumer products.

The Relying Party. The server, service or company that makes use of the information in the EAT about the entity.

In all operating models, the manufacturer provisions some secret attestation key material (AKM) into the entity during manufacturing. This might be during the manufacturer of a chip at a fabrication facility (fab) or during final assembly of a consumer product or any time in between. This attestation key material is used for signing EATs.

In all operating models, hardware and/or software on the entity create an EAT of the format described in this document. The EAT is always signed by the attestation key material provisioned by the manufacturer.

In all operating models, the relying party must end up knowing that the signature on the EAT is valid and consistent with data from
claims in the EAT. This can happen in many different ways. Here are some examples.

- The EAT is transmitted to the relying party. The relying party gets corresponding key material (e.g., a root certificate) from the manufacturer. The relying party performs the verification.

- The EAT is transmitted to the relying party. The relying party transmits the EAT to a verification service offered by the manufacturer. The server returns the validated claims.

- The EAT is transmitted directly to a verification service, perhaps operated by the manufacturer or perhaps by another party. It verifies the EAT and makes the validated claims available to the relying party. It may even modify the claims in some way and re-sign the EAT (with a different signing key).

This standard supports all these operating models and does not prefer one over the other. It is important to support this variety of operating models to generally facilitate deployment and to allow for some special scenarios. One special scenario has a validation service that is monetized, most likely by the manufacturer. In another, a privacy proxy service processes the EAT before it is transmitted to the relying party. In yet another, symmetric key material is used for signing. In this case, the manufacturer should perform the verification, because any release of the key material would enable a participant other than the entity to create valid signed EATs.

1.4. What is Not Standardized

1.4.1. Transmission Protocol

EATs may be transmitted by any protocol. For example, they might be added in extension fields of other protocols, bundled into an HTTP header, or just transmitted as files. This flexibility is intentional to allow broader adoption. This flexibility is possible because EATs are self-secured with signing (and possibly additionally with encryption and anti-replay). The transmission protocol is not required to fulfill any additional security requirements.

For certain devices, a direct connection may not exist between the EAT-producing device and the Relying Party. In such cases, the EAT should be protected against malicious access. The use of COSE allows for signing and encryption of the EAT. Therefore even if the EAT is conveyed through intermediaries between the device and Relying Party,
such intermediaries cannot easily modify the EAT payload or alter the signature.

1.4.2. Signing Scheme

The term "signing scheme" is used to refer to the system that includes end-end process of establishing signing attestation key material in the entity, signing the EAT, and verifying it. This might involve key IDs and X.509 certificate chains or something similar but different. The term "signing algorithm" refers just to the algorithm ID in the COSE signing structure. No particular signing algorithm or signing scheme is required by this standard.

There are three main implementation issues driving this. First, secure non-volatile storage space in the entity for the attestation key material may be highly limited, perhaps to only a few hundred bits, on some small IoT chips. Second, the factory cost of provisioning key material in each chip or device may be high, with even millisecond delays adding to the cost of a chip. Third, privacy-preserving signing schemes like ECDAA (Elliptic Curve Direct Anonymous Attestation) are complex and not suitable for all use cases.

Eventually some form of standardization of the signing scheme may be required. This might come in the form of another standard that adds to this document, or when there is clear convergence on a small number of signing schemes this standard can be updated.

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], COSE [RFC8152], and CWT [RFC8392].

StringOrURI. The "StringOrURI" term in this specification has the same meaning and processing rules as the JWT "StringOrURI" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR text string instead of a JSON text string.

NumericDate. The "NumericDate" term in this specification has the same meaning and processing rules as the JWT "NumericDate" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR numeric date (from Section 2.4.1 of [RFC7049]) instead
of a JSON number. The encoding is modified so that the leading
tag 1 (epoch-based date/time) MUST be omitted.

Claim Name. The human-readable name used to identify a claim.

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

Claim Value. The CBOR map value representing the value of the claim.

CWT Claims Set. The CBOR map that contains the claims conveyed by
the CWT.

FloatOrNumber. The "FloatOrNumber" term in this specification is the
type of a claim that is either a CBOR positive integer, negative
integer or floating point number.

Attestation Key Material (AKM). The key material used to sign the
EAT token. If it is done symmetrically with HMAC, then this is a
simple symmetric key. If it is done with ECC, such as an IEEE
DevID [IDevID], then this is the private part of the EC key pair.
If ECDAA is used, (e.g., as used by Enhanced Privacy ID, i.e.
EPID) then it is the key material needed for ECDAA.

3. The Claims

3.1. Universal Entity ID (UEID) Claim

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

It is identified by Claim Key X (X is TBD).

UEID’s must be universally and globally unique across manufacturers
and countries. UEIDs must also be unique across protocols and
systems, as tokens are intended to be embedded in many different
protocols and systems. No two products anywhere, even in completely
different industries made by two different manufacturers in two
different countries. should have the same UEID (if they are not
global and universal in this way then relying parties receiving them
will have to track other characteristics of the device to keep
devices distinct between manufacturers).

The UEID should be permanent. It should never change for a given
device / entity. In addition, it should not be reprogrammable.
UEID’s are binary byte-strings (resulting in a smaller size than text strings). When handled in text-based protocols, they should be base-64 encoded.

UEID’s are variable length with a maximum size of 33 bytes (1 type byte and 256 bits). A receiver of a token with UEIDs may reject the token if a UEID is larger than 33 bytes.

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

A UEID is a byte string. From the consumer’s view (the rely party) it is opaque with no bytes having any special meaning.

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>GUID</td>
<td>This is a 128 to 256 bit random number generated once and stored in the device. The GUID may be constructed from various identifiers on the device using a hash function or it may be just the raw random number. In any case, the random number must have entropy of at least 128 bits as this is what gives the global</td>
</tr>
<tr>
<td>0x02</td>
<td>IEEE EUI</td>
<td>This makes use of the IEEE company identification registry. An EUI is made up of an OUI and OUI-36 or a CID, different registered company identifiers, and some unique per-device identifier. EUIs are often the same as or similar to MAC addresses. (Note that while devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) TODO: normative references to IEEE.</td>
</tr>
<tr>
<td>0x03</td>
<td>IMEI</td>
<td>TODO: figure how to specify IMEIs</td>
</tr>
</tbody>
</table>

Table 1: UEID Composition Types

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

- UEIDs types may vary freely from one manufacturer to the next.
- New types of UEIDs may be created. For example a type 0x04 UEID may be created based on some other manufacturer registration scheme.
- Device manufacturers are allowed to change from one type of UEID to another anytime they want. For example they may find they can optimize their manufacturing by switching from type 0x01 to type 0x02 or vice versa. The main requirement on the manufacturer is that UEIDs be universally unique.

3.2. Origination (origination) Claims

This claim describes the parts of the device or entity that are creating the EAT. Often it will be tied back to the device or chip manufacturer. The following table gives some examples:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme-TEE</td>
<td>The EATs are generated in the TEE authored and configured by &quot;Acme&quot;</td>
</tr>
<tr>
<td>Acme-TPM</td>
<td>The EATs are generated in a TPM manufactured by &quot;Acme&quot;</td>
</tr>
<tr>
<td>Acme-Linux-Kernel</td>
<td>The EATs are generated in a Linux kernel configured and shipped by &quot;Acme&quot;</td>
</tr>
<tr>
<td>Acme-TA</td>
<td>The EATs are generated in a Trusted Application (TA) authored by &quot;Acme&quot;</td>
</tr>
</tbody>
</table>

The claim is represented by Claim Key X+1. It is type StringOrURI.

TODO: consider a more structure approach where the name and the URI and other are in separate fields.

TODO: This needs refinement. It is somewhat parallel to issuer claim in CWT in that it describes the authority that created the token.

3.3. OEM identification by IEEE OUI

This claim identifies a device OEM by the IEEE OUI. Reference TBD. It is a byte string representing the OUI in binary form in network byte order (TODO: confirm details).
Companies that have more than one IEEE OUI registered with IEEE should pick one and prefer that for all their devices.

Note that the OUI is in common use as a part of MAC Address. This claim is only the first bits of the MAC address that identify the manufacturer. The IEEE maintains a registry for these in which many companies participate. This claim is represented by Claim Key TBD.

3.4. Security Level (seclevel) Claim

EATs have a claim that roughly characterizes the device / entities ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is done by roughly defining four security levels as described below. This is similar to the security levels defined in the Metadata Service defined by the Fast Identity Online (FIDO) Alliance (TODO: reference).

These claims describe security environment and countermeasures available on the end-entity / client device where the attestation key reside and the claims originate.

This claim is identified by Claim Key X+2. The value is an integer between 1 and 4 as defined below.

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 should not be general-purpose operating environments that host features such as app download systems, web browsers and complex productivity applications. It is akin to the Secure Restricted level (see below) without the security orientation. Examples include a WiFi subsystem, an IoT camera, or sensor device.

3 - Secure Restricted Entities at this level must meet the criteria defined by FIDO Allowed Restricted Operating Environments (TODO: reference). Examples include TEE’s and schemes using virtualization-based security. Like the FIDO security goal, security at this level is aimed at defending well against large-scale network / remote attacks against the device.

4 - Hardware Entities at this level must include substantial defense against physical or electrical attacks against the device itself. It is assumed any potential attacker has captured the device and can disassemble it. Example include TPMs and Secure Elements.
This claim is not intended as a replacement for a proper end-device security certification schemes such as those based on FIPS (TODO: reference) or those based on Common Criteria (TODO: reference). The claim made here is solely a self-claim made by the Entity Originator.

3.5. Nonce (nonce) Claim

The "nonce" (Nonce) claim represents a random value that can be used to avoid replay attacks. This would be ideally generated by the CWT consumer. This value is intended to be a CWT companion claim to the existing JWT claim **_IANAJWT_** (TODO: fix this reference). The nonce claim is identified by Claim Key X+3.

3.6. Secure Boot and Debug Enable State Claims

3.6.1. Secure Boot Enabled (secbootenabled) Claim

The "secbootenabled" (Secure Boot Enabled) claim represents a boolean value that indicates whether secure boot is enabled either for an entire device or an individual submodule. If it appears at the device level, then this means that secure boot is enabled for all submodules. Secure boot enablement allows a secure boot loader to authenticate software running either in a device or a submodule prior allowing execution. This claim is identified by Claim Key X+4.

3.6.2. Debug Disabled (debugdisabled) Claim

The "debugdisabled" (Debug Disabled) claim represents a boolean value that indicates whether debug capabilities are disabled for an entity (i.e. value of ‘true’). Debug disablement is considered a prerequisite before an entity is considered operational. This claim is identified by Claim Key X+5.

3.6.3. Debug Disabled Since Boot (debugdisabledsincelboot) Claim

The "debugdisabledsincelboot" (Debug Disabled Since Boot) claim represents a boolean value that indicates whether debug capabilities for the entity were not disabled in any way since boot (i.e. value of ‘true’). This claim is identified by Claim Key X+6.

3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim

The "debugpermanentdisable" (Debug Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of ‘true’). This value can be set to ‘true’ also if only the manufacturer is allowed to enabled debug, but the end user is not. This claim is identified by Claim Key X+7.
3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim

The "debugfullpermanentdisable" (Debug Full Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can only be set to 'true' if no party can enable debug capabilities for the entity. Often this is implemented by blowing a fuse on a chip as fuses cannot be restored once blown. This claim is identified by Claim Key X+8.

3.7. Location (loc) Claim

The "loc" (location) claim is a CBOR-formatted object that describes the location of the device entity from which the attestation originates. It is identified by Claim Key X+10. It is comprised of an array of additional subclaims that represent the actual location coordinates (latitude, longitude and altitude). The location coordinate claims are consistent with the WGS84 coordinate system [WGS84]. In addition, a subclaim providing the estimated accuracy of the location measurement is defined.

3.7.1. lat (latitude) claim

The "lat" (latitude) claim contains the value of the device location corresponding to its latitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+11.

3.7.2. long (longitude) claim

The "long" (longitude) claim contains the value of the device location corresponding to its longitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+12.

3.7.3. alt (altitude) claim

The "alt" (altitude) claim contains the value of the device location corresponding to its altitude coordinate (if available). It is of data type FloatOrNumber and identified by Claim Key X+13.

3.7.4. acc (accuracy) claim

The "acc" (accuracy) claim contains a value that describes the location accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+14.
3.7.5. altacc (altitude accuracy) claim

The "altacc" (altitude accuracy) claim contains a value that describes the altitude accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+15.

3.7.6. heading claim

The "heading" claim contains a value that describes direction of motion for the entity. Its value is specified in degrees, between 0 and 360. It is of data type FloatOrNumber and identified by Claim Key X+16.

3.7.7. speed claim

The "speed" claim contains a value that describes the velocity of the entity in the horizontal direction. Its value is specified in meters/second and must be non-negative. It is of data type FloatOrNumber and identified by Claim Key X+17.

3.8. ts (timestamp) claim

The "ts" (timestamp) claim contains a timestamp derived using the same time reference as is used to generate an "iat" claim (see Section 3.1.6 of [RFC8392]). It is of the same type as "iat" (integer or floating-point), and is identified by Claim Key X+18. It is meant to designate the time at which a measurement was taken, when a location was obtained, or when a token was actually transmitted. The timestamp would be included as a subclaim under the "submod" or "loc" claims (in addition to the existing respective subclaims), or at the device level.

3.9. age claim

The "age" claim contains a value that represents the number of seconds that have elapsed since the token was created, measurement was made, or location was obtained. Typical attestable values are sent as soon as they are obtained. However in the case that such a value is buffered and sent at a later time and a sufficiently accurate time reference is unavailable for creation of a timestamp, then the age claim is provided. It is identified by Claim Key X+19.

3.10. uptime claim

The "uptime" claim contains a value that represents the number of seconds that have elapsed since the entity or submod was last booted. It is identified by Claim Key X+20.
3.11. The submods Claim

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

The claims for each these can be grouped together in a submodule.

Specifically, the "submods" claim is an array. Each item in the array is a CBOR map containing all the claims for a particular submodule. It is identified by Claim Key X+22.

The security level of the submod is assumed to be at the same level as the main entity unless there is a security level claim in that submodule indicating otherwise. The security level of a submodule can never be higher (more secure) than the security level of the EAT it is a part of.

3.11.1. The submod_name Claim

Each submodule should have a submod_name claim that is descriptive name. This name should be the CBOR txt type.

3.11.2. Nested EATs, the eat Claim

It is allowed for one EAT to be embedded in another. This is for complex devices that have more than one subsystem capable of generating an EAT. Typically one will be the device-wide EAT that is low to medium security and another from a Secure Element or similar that is high security.

The contents of the "eat" claim must be a fully signed, optionally encrypted, EAT token. It is identified by Claim Key X+23.

4. CBOR Interoperability

EAT is a one-way protocol. It only defines a single message that goes from the entity to the server. The entity implementation will often be in a contained environment with little RAM and the server will usually not be. The following requirements for interoperability take that into account. The entity can generally use whatever encoding it wants. The server is required to support just about every encoding.

Canonical CBOR encoding is explicitly NOT required as it would place an unnecessary burden on the entity implementation.
4.1. Integer Encoding (major type 0 and 1)

The entity may use any integer encoding allowed by CBOR. The server MUST accept all integer encodings allowed by CBOR.

4.2. String Encoding (major type 2 and 3)

The entity can use any string encoding allowed by CBOR including indefinite lengths. It may also encode the lengths of strings in any way allowed by CBOR. The server must accept all string encodings.

Major type 2, bstr, SHOULD be have tag 21, 22 or 23 to indicate conversion to base64 or such when converting to JSON.

4.3. Map and Array Encoding (major type 4 and 5)

The entity can use any array or map encoding allowed by CBOR including indefinite lengths. Sorting of map keys is not required. Duplicate map keys are not allowed. The server must accept all array and map encodings. The server may reject maps with duplicate map keys.

4.4. Date and Time

The entity should send dates as tag 1 encoded as 64-bit or 32-bit integers. The entity may not send floating point dates. The server must support tag 1 epoch based dates encoded as 64-bit or 32-bit integers.

The entity may send tag 0 dates, however tag 1 is preferred. The server must support tag 0 UTC dates.

4.5. URIs

URIs should be encoded as text strings and marked with tag 32.

4.6. Floating Point

Encoding data in floating point is to be used only if necessary. Location coordinates are always in floating point. The server must support decoding of all types of floating point.

4.7. Other types

Use of Other types like bignums, regular expressions and so SHOULD NOT be used. The server MAY support them, but is not required to.

Use of these tags is
5. IANA Considerations

5.1. Reuse of CBOR Web Token (CWT) Claims Registry

Claims defined for EAT are compatible with those of CWT so the CWT Claims Registry is re-used. New new IANA registry is created. All EAT claims should be registered in the CWT Claims Registry.

5.1.1. Claims Registered by This Document

- Claim Name: UEID
- Claim Description: The Universal Entity ID
- JWT Claim Name: N/A
- Claim Key: X
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): *this document*

TODO: add the rest of the claims in here

5.2. EAT CBOR Tag Registration

How an EAT consumer determines whether received CBOR-formatted data actually represents a valid EAT is application-dependent, much like a CWT. For instance, a specific MIME type associated with the EAT such as "application/eat" could be sufficient for identification of the EAT. Note however that EAT’s can include other EAT’s (e.g. a device EAT comprised of several submodule EAT’s). In this case, a CBOR tag dedicated to the EAT will be required at least for the submodule EAT’s and the tag must be a valid CBOR tag. In other words - the EAT CBOR tag can optionally prefix a device-level EAT, but a EAT CBOR tag must always prefix a submodule EAT. The proposed EAT CBOR tag is 71.

5.2.1. Tag Registered by This Document

- CBOR Tag: 71
- Data Item: Entity Attestation Token (EAT)
- Semantics: Entity Attestation Token (CWT), as defined in *this_doc*
6. 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.

6.1. UEID Privacy Considerations

A UEID is usually not privacy preserving. Any set of relying parties that receives tokens that happen to be from a single device will be able to know the tokens are all from the same device and be able to track the device. Thus, in many usage situations ueid violates governmental privacy regulation. In other usage situations 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.

There are several strategies that can be used to still be able to put UEID’s in tokens:

- The device obtains explicit permission from the user of the device to use the UEID. 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.

- The UEID is used only in a particular context or particular use case. It is used only by one relying party.

- The device authenticates the relying party and generates a derived UEID just for that particular relying party. For example, the relying party could prove their identity cryptographically to the device, then the device generates a UEID just for that relying party by hashing a proofed relying party ID with the main device UEID.

Note that some of these privacy preservation strategies result in multiple UEIDs per device. Each UEID is used in a different context, use case or system on the device. However, from the view of the relying party, there is just one UEID and it is still globally universal across manufacturers.
7. Security Considerations

TODO: Perhaps this can be the same as CWT / COSE, but not sure yet because it involves so much entity / device security that those do not.

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Examples

A.1. Very Simple EAT

This is shown in CBOR diagnostic form. Only the payload signed by COSE is shown.

```
{
  / nonce /          11:h'948f8860d13a463e8e',
  / UEID /           8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled / 13:true,
  / debugpermanentdisable / 15:true,
  / ts /            21:1526542894,
}
```

A.2. Example with Submodules, Nesting and Security Levels

```
{
  / nonce /          11:h'948f8860d13a463e8e',
  / UEID /           8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled / 13:true,
  / debugpermanentdisable / 15:true,
  / ts /            21:1526542894,
  / seclevel /       10:3, / secure restricted OS /
  / submods / 30:
  [              
    / 1st submod, an Android Application / {  
      / submod_name /   30:'Android App "Foo"',
      / seclevel /      10:1, / unrestricted /
      / app data /     -70000:'text string'
    },
    / 2nd submod, A nested EAT from a secure element / {  
      / submod_name / 30:'Secure Element EAT',
      / eat /         31:71( 18(  
        / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]
      )
    }
    / 3rd submod, information about Linux Android / {  
      / submod_name/ 30:'Linux Android',
      / seclevel /   10:1, / unrestricted /
      / custom - release / -80000:'8.0.0',
      / custom - version / -80001:'4.9.51+'
    }
  ]
}
```
Authors’ Addresses

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA
Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Laurence Lundblade
Security Theory LLC
EMail: lgl@island-resort.com

Miguel Ballesteros
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA
Phone: +1 858 651 4299
EMail: mballest@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