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).
Internet-Draft RATS Arch & Terms March 2019

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-usecases].

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 Attester’s current location - e.g. GPS coordinates. * 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.tschofenig-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.

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

4.1.2. Verifier Duties
  o Acquisition and storage of assertion semantics
  o Acquisition and storage of appraisal policies
  o Verification of Attester Identity (attestation provenance)
  o Comparing assertions or evidence with reference values according to appraisal policies
  o 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
  o Hardens the device or service that implements the Attester role
  o Provisions device identities and/or key material accessible to the Attester role
  o Evaluates trustworthiness during manufacturing, supply chain and onboarding
  o Produces trustworthiness assertions applicable to the Attestor role
  o Embeds trustworthiness assertions about the Attester role in the device or service during manufacturing, supply chain or onboarding

4.1.4. Relying Party Duties
  o Evaluate assertions/evidence locally as far as possible
  o Compare trust policies to attestation-results based on assertions or evidence
  o 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 simple 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 Attestor 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 Attestor 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 Attestor. A supply chain entity can embed additional trusted assertions to an Attestor. 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

Attestor 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 Attestor 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 Actor 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 an 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. Figure 2 provides a simplified overview of the RATS Roles defined above,
illustrating a general Interconnect in the center that facilitates all RATS Interactions.

```
+------------+                     +------------------+
|            |                     |                  |
|  Attester  |                  +->|  Verifier        |
|            |                  |  |                  |
|            |                  |  +------------------+
|            |                  |                         |
|  Interconnect  <+<|                        |
|  v          |                        |                         |
|            |                  +->|  Relying Party       |
+------------+                     +------------------+
```

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.

$ Computing Context : 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 [ref docker, find a more general term here] 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.

o Singleton,

o 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,
o An FPGA,
o An ASIC,
o A memory resource,
o 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 an 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.
$ Trustworthy Computing Context: A Computing Context that guarantees trustworthy behavior and/or composition (with respect to certain declarative guidance and a scope of confidence). A trustworthy Computing Context is a trustworthy system.

<NMS: is this necessary?> Trustworthy Statement: Evidence conveyed by a Computing Context that is not necessarily trustworthy.

[update with tamper related terms]

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


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
Abstract

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

Status of This Memo

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

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

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

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

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

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119].

2. The YANG Module for Basic Remote Attestation Procedures

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

2.1. Tree format

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

rpcs:
  +---x tpm12-challenge-response-attestation
     |  +---w input
     |     |  +---w tpm1-attestation-challenge
     |     |     |  +---w pcr-indices*    uint8
     |     |     |  +---w nonce-value      binary
     |     |     |  +---w TPM_SIG_SCHEME-value    uint8
</CODE BEGINS>
2.2. Raw Format

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

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

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

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

grouping hash {
  description "The hash value including hash-algo identifier";
  list hash-digests {
    description "The list of hashes.";
    container hash-digest {
      description "A hash value based on a hash algorithm registered by an
uses hash-algo;
leaf hash-value {
    type binary;
    description
        "The binary representation of the hash value.";
}
}
}

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

grouping tpm12-pcr-selection {
    description
        "A Verifier can request one or more PCR values using its
         individually created Attestation Key Certificate (AC).
         The corresponding selection filter is represented in this grouping.
         Requesting a PCR value that is not in scope of the AC used, detailed
         exposure via error msg should be avoided.";
leaf-list pcr-indices {
    type uint8;
    description
        "The numbers/indexes of the PCRs. At the moment this is limited
         to 32.";
}
}

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

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

grouping pcr-selector {
  description
  "A Verifier can request the generation of an attestation certificate (a signed public attestation key (non-migratable, tpm-resident) wrt one or more PCR values. The corresponding creation input is represented in this grouping. Requesting a PCR value that is not supported results in an error, detailed exposure via error msg should be avoided.";
  list pcr-list {
    description
    "For each PCR in this list an individual hash-algo can be requested.";
    container pcr {
      description
      "The composite of a PCR number and corresponding bank numbers.";
      leaf-list pcr-index {
        type uint8;
        description
        "The numbers of the PCRs that are associated with the created key. At the moment the highest number is 32";
      }
      uses hash-algo;
    }
  }
}
grouping tpm12-signature-scheme {
   description
   "The signature scheme used to sign the evidence via a TPM 1.2.";
   leaf TPM_SIG_SCHEME-value {
      type uint8;
      mandatory true;
      description
      "Selects the signature scheme that is used to sign the TPM quote information response. Allowed values can be found in the table at the bottom of page 32 in the TPM 1.2 Structures specification (Level 2 Revision 116, 1 March 2011).";
   }
}

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

grouping tpm12-attestation-key-identifier {
description
  "A selector for a suitable key identifier for a TPM 1.2."
choice key-identifier {
  description
  "Identifier for the attestation key to use for signing
  attestation evidence.";
  case public-key {
    leaf pub-key-id {
      type binary;
      description
        "The value of the identifier for the public key.";
    }
  }
  case TSS_UUID {
    description
      "Use a YANG agent generated (and maintained) attestation
      key UUID that complies with the TSS_UUID datatype of the TCG
      Software Stack (TSS) Specification, Version 1.10 Golden,
      August 20, 2003.";
    container TSS_UUID-value {
      description
        "A detailed structure that is used to create the
        TPM 1.2 native TSS_UUID as defined in the TCG Software
        Stack (TSS) Specification, Version 1.10 Golden,
        August 20, 2003.";
      leaf ulTimeLow {
        type uint32;
        description
          "The low field of the timestamp.";
      }
      leaf usTimeMid {
        type uint16;
        description
          "The middle field of the timestamp.";
      }
      leaf usTimeHigh {
        type uint16;
        description
          "The high field of the timestamp multiplexed with the
          version number.";
      }
      leaf bClockSeqHigh {
        type uint8;
        description
          "The high field of the clock sequence multiplexed with
          the variant.";
      }
      leaf bClockSeqLow {  

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

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

grouping tpm-name {
  description
  "In a system with multiple-TPMs get the data from a specific TPM identified by the name and physical-index."
  leaf tpm_name {
    type string;
description
"The spatially unique node identifier.";
    }
  }
}
"Name of the TPM or All";
}
leaf tpm-physical-index {
  if-feature ietfhw:entity-mib;
  type int32 {
    range "1..2147483647";
  }
  config false;
  description
    "The entPhysicalIndex for the TPM."
  reference
    "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex"
}
}
grouping compute-node {
  description
    "In a distributed system with multiple compute nodes
    this is the node identified by name and physical-index."
  leaf node-name {
    type string;
    description
      "Name of the compute node or All"
  }
  leaf node-physical-index {
    if-feature ietfhw:entity-mib;
    type int32 {
      range "1..2147483647";
    }
    config false;
    description
      "The entPhysicalIndex for the compute node."
    reference
      "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex"
  }
}
}
grouping tpm12-pcr-info-short {
  description
    "This structure is for defining a digest at release when the only
    information that is necessary is the release configuration."
  uses tpm12-pcr-selection;
  leaf locality-at-release {
    type uint8;
    description
      "This SHALL be the locality modifier required to release the
      information (TPM 1.2 type TPM_LOCALITY_SELECTION)"
  }
  leaf digest-at-release {

type binary;

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

grouping tpm12-version {

description
   "This structure provides information relative the version of
the TPM."
list version {

description
   "This indicates the version of the structure
(TPM 1.2 type TPM_STRUCT_VER). This MUST be 1.1.0.0."
leaf major {

type uint8;

description
   "Indicates the major version of the structure.
   MUST be 0x01."
}
leaf minor {

type uint8;

description
   "Indicates the minor version of the structure.
   MUST be 0x01."
}
leaf revMajor {

type uint8;

description
   "Indicates the rev major version of the structure.
   MUST be 0x00."
}
leaf revMinor {

type uint8;

description
   "Indicates the rev minor version of the structure.
   MUST be 0x00."
}
}

grouping tpm12-quote-info-common {

description
   "These statements are used in bot quote variants of the TPM 1.2"
leaf fixed {

type binary;
description
    "This SHALL always be the string 'QUOT' or 'QUO2'
    (length is 4 bytes).";
}
leaf external-data {
    type binary;
    description
        "160 bits of externally supplied data, typically a nonce.";
}
leaf signature-size {
    type uint32;
    description
        "The size of TPM 1.2 'signature' value.";
}
leaf signature {
    type binary;
    description
        "Signature over SHA-1 hash of tpm12-quote-info2'.";
}
}

grouping tpm12-quote-info {
    description
        "This structure provides the mechanism for the TPM to quote the
        current values of a list of PCRs (as used by the TPM_Quote2
        command).";
    uses tpm12-version;
    leaf digest-value {
        type binary;
        description
            "This SHALL be the result of the composite hash algorithm using
            the current values of the requested PCR indices
            (TPM 1.2 type TPM_COMPOSITE_HASH).";
    }
}

grouping tpm12-quote-info2 {
    description
        "This structure provides the mechanism for the TPM to quote the
        current values of a list of PCRs
        (as used by the TPM_Quote2 command).";
    leaf tag {
        type uint8;
        description
            "This SHALL be TPM_TAG_QUOTE_INFO2.";
    }
    uses tpm12-pcr-info-short;
}
grouping tpm12-cap-version-info {
  description "TPM returns the current version and revision of the TPM 1.2.";
  list TPM_PCR_COMPOSITE {
    description "The TPM 1.2 TPM_PCRVALUEs for the pcr-indices.";
    uses tpm12-pcr-selection;
    leaf value-size {
      type uint32;
      description "This SHALL be the size of the 'tpm12-pcr-value' field (not the number of PCRs).";
    }
    leaf-list tpm12-pcr-value {
      type binary;
      description "The list of TPM_PCRVALUEs from each PCR selected in sequence of tpm12-pcr-selection.";
    }
  }
  leaf-list version-info {
    description "An optional output parameter from a TPM 1.2 TPM_Quote2.";
    leaf tag {
      type uint16;
      description "The TPM 1.2 version and revision (TPM 1.2 type TPM_STRUCTURE_TAG). This MUST be TPM_CAP_VERSION_INFO (0x0030)";
    }
    uses tpm12-version;
    leaf spec-level {
      type uint16;
      description "A number indicating the level of ordinals supported.";
    }
    leaf errata-rev {
      type uint8;
      description "A number indicating the errata version of the specification.";
    }
    leaf tpm-vendor-id {
      type binary;
      description "The vendor ID unique to each TPM manufacturer.";
    }
    leaf vendor-specific-size {
      type uint16;
    }
  }
}
grouping tpm12-pcr-composite {
    description "The actual values of the selected PCRs (a list of TPM_PCRVALUEs (binary) and associated metadata for TPM 1.2.";
    list TPM_PCR_COMPOSITE {
        description "The TPM 1.2 TPM_PCRVALUEs for the pcr-indices.";
        uses tpm12-pcr-selection;
        leaf value-size {
            type uint32;
            description "This SHALL be the size of the 'tpm12-pcr-value' field (not the number of PCRs).";
        }
        leaf-list tpm12-pcr-value {
            type binary;
            description "The list of TPM_PCRVALUEs from each PCR selected in sequence of tpm12-pcr-selection.";
        }
    }
}

grouping node-uptime {
    description "Uptime in seconds of the node.";
    leaf up-time {
        type uint32;
        description "Uptime in seconds of this node reporting its data";
    }
}

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

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

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

grouping boot-event-log {
  description
    "Defines an event log corresponding to the event that extended the PCR."
  leaf event-number {
    type uint32;
    description
      "Unique event number of this event"
  }
  leaf event-type {
    type uint32;
    description
      "log event type"
  }
  leaf pcr-index {
    type uint16;
    description
      "Defines the PCR index that this event extended"
  }
  list digest-list {
    description "Hash of event data";
}
uses hash-algo;
leaf-list digest {
  type binary;
  description
    "The hash of the event data";
}
leaf event-size {
  type uint32;
  description
    "Size of the event data";
}
leaf-list event-data {
  type uint8;
  description
    "the event data size determined by event-size";
}
}
grouping ima-event {
  description
    "Defines an hash log extend event for IMA measurements";
  leaf event-number {
    type uint64;
    description
      "Unique number for this event for sequencing";
  }
  leaf ima-template {
    type string;
    description
      "Name of the template used for event logs
      for e.g. ima, ima-ng";
  }
  leaf filename-hint {
    type string;
    description
      "File that was measured";
  }
  leaf filedata-hash {
    type binary;
    description
      "Hash of filedata";
  }
  leaf template-hash-algorithm {
    type string;
    description
      "Algorithm used for template-hash";
  }
}
leaf template-hash {
    type binary;
    description
        "hash(filedata-hash, filename-hint)";
}
leaf pcr-index {
    type uint16;
    description
        "Defines the PCR index that this event extended";
}
leaf signature {
    type binary;
    description
        "The file signature";
}

grouping bios-event-log {
    description
        "Measurement log created by the BIOS/UEFI.";
    list bios-event-entry {
        key event-number;
        description
            "Ordered list of TCG described event log
             that extended the PCRs in the order they
             were logged";
        uses boot-event-log;
    }
}

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

grouping event-logs {
    description
        "A selector for the log and its type.";
    choice log-type {
        mandatory true;
        description
            "Event log type determines the event logs content.";
    }
}
case bios {
  description
  "BIOS/UEFI event logs";
  container bios-event-logs {
    description
    "This is an index referencing the TCG Algorithm
    Registry based on TPM_ALG_ID.";
    uses bios-event-log;
  }
}
case ima {
  description
  "IMA event logs";
  container ima-event-logs {
    description
    "This is an index referencing the TCG Algorithm
    Registry based on TPM_ALG_ID.";
    uses ima-event-log;
  }
}
}

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

rpc tpm12-challenge-response-attestation {
  description
  "This RPC accepts the input for TSS TPM 1.2 commands of the
  managed device. ComponentIndex from the hardware manager YANG
  module to refer to dedicated TPM in composite devices,
  e.g. smart NICs, is still a TODO.";
  input {
    container tpm1-attestation-challenge {
      description
      "This container includes every information element defined
      in the reference challenge-response interaction model for
      remote attestation. Corresponding values are based on
      TPM 1.2 structure definitions";
      uses tpm12-pcr-selection;
      uses nonce;
      uses tpm12-signature-scheme;
      uses tpm12-attestation-key-identifier;
      leaf add-version {
        type boolean;
        description
        "Whether or not to include TPM_CAP_VERSION_INFO; if true,
        then TPM_Quote2 must be used to create the response.";
      }
      uses tpm-name;
    }
  }
}
output {
  list tpm12-attestation-response {
    key tpm_name;
    description
      "The binary output of TPM 1.2 TPM_Quote/TPM_Quote2, including
       the PCR selection and other associated attestation evidence
       metadata";
    uses tpm-name;
    uses node-uptime;
    uses compute-node;
    uses tpm12-quote-info-common;
    choice tpm12-quote {
      mandatory true;
      description
        "Either a tpm12-quote-info or tpm12-quote-info2, depending
         on whether TPM_Quote or TPM_Quote2 was used
         (cf. input field add-verison).");
      case tpm12-quote1 {
        description
          "BIOS/UEFI event logs";
        uses tpm12-quote-info;
        uses tpm12-pcr-composite;
      }
      case tpm12-quote2 {
        description
          "BIOS/UEFI event logs";
        uses tpm12-quote-info2;
      }
    }
  }
}
}

rpc tpm20-challenge-response-attestation {
  description
    "This RPC accepts the input for TSS TPM 2.0 commands of the
     managed device. ComponentIndex from the hardware manager YANG
     module to refer to dedicated TPM in composite devices,
     e.g. smart NICs, is still a TODO.";
  input {
    container tpm20-attestation-challenge {
      description
        "This container includes every information element defined
         in the reference challenge-response interaction model for
         remote attestation. Corresponding values are based on
         TPM 2.0 structure definitions";
      uses tpm20-pcr-selection;
    }
  }
}
uses nonce;
uses tpm20-signature-scheme;
uses tpm20-attestation-key-identifier;
}
list tpms {
key tpm_name;
description
"TPMs to fetch the attestation information.";
uses tpm-name;
}
}
output {
list tpm20-attestation-response {
key tpm_name;
description
"The binary output of TPM2b_Quote. An TPMS_ATTEST structure
including a length, encapsulated in a signature";
uses tpm-name;
uses node-up-time;
uses compute-node;
container tpms-attest {
leaf pcrdigest {
type binary;
description
"split out value of TPMS_QUOTE_INFO for convenience";
}
leaf tpms-attest-result {
type binary;
description
"The complete TPM generate structure including
signature.";
}
leaf tpms-attest-result-length {
type uint32;
description
"Length of attest result provided by the TPM structure.";
}
description
"A composite of value and length and list of selected
pcrs (original name: [type]attested)";
}
leaf tpmt-signature {
type binary;
description
"Split out value of the signature for convenience.
TODO: check for length values that complent binary value
data node leaves.";
}
rpc basic-trust-establishment {
    description
        "This RPC creates a tpm-resident, non-migratable key to be used
in TPM_Quote commands, an attestation certificate.";
    input {
        uses nonce;
        uses tpm20-signature-scheme;
        uses tpm-name;
        leaf certificate-name {
            type string;
            description
                "An arbitrary name for the identity certificate chain
requested.";
        }
    }
    output {
        list attestation-certificates {
            key tpm_name;
            description
                "Attestation Certificate data from a TPM identified by the TPM
name";
            uses tpm-name;
            uses node-uptime;
            uses compute-node;
            leaf certificate-name {
                type string;
                description
                    "An arbitrary name for this identity certificate or
certificate chain.";
            }
            leaf attestation-certificate {
                type ietfct:end-entity-cert-cms;
                description
                    "The binary signed certificate chain data for this identity
certificate.";
            }
            uses tpm20-attestation-key-identifier;
        }
    }
}

rpc log-retrieval {
    description
        "Logs Entries are either identified via indices or via providing
the last line received. The number of lines returned can be limited. The type of log is a choice that can be augmented."

input {
  list log-selector {
    key node-name;
    description
      "Selection of log entries to be reported.";
    uses compute-node;
    choice index-type {
      description
        "Last log entry received, log index number, or timestamp.";
      case last-entry {
        description
          "The last entry of the log already retrieved.";
        leaf last-entry-value {
          type binary;
          description
            "Content of an log event which matches 1:1 with a unique event record contained within the log. Log entries subsequent to this will be passed to the requester. Note: if log entry values are not unique, this MUST return an error.";
        }
      }
      case index {
        description
          "Numeric index of the last log entry retrieved, or zero.";
        leaf index-number {
          type uint64;
          description
            "The numeric index number of a log entry. Zero means to start at the beginning of the log. Entries subsequent to this will be passed to the requester.";
        }
      }
      case timestamp {
        leaf timestamp {
          type yang:date-and-time;
          description
            "Timestamp from which to start the extraction. The next log entry subsequent to this timestamp is to be sent.";
        }
        description
          "Timestamp from which to start the extraction.";
      }
  }
}
uses log-identifier;
uses tpm20-pcr-selection;
leaf log-entry-quantity {
  type uint16;
  description
    "The number of log entries to be returned. If omitted, it
    means all of them.";
}
output {
  container system-event-logs {
    description
      "The requested data of the measurement event logs";
    list node-data {
      key "node-name tpm_name";
      description
        "Event logs of a node in a distributed system
         identified by the node name";
      uses compute-node;
      uses node-uptime;
      uses tpm-name;
      container log-result {
        description
          "The requested entries of the corresponding log.";
        uses event-logs;
      }
    }
  }
}
container rats-support-structures {
  config false;
  description
    "The datastore definition enabling verifiers or relying
     parties to discover the information necessary to use the
     remote attestation RPCs appropriately.";
  leaf-list supported-algos {
    type uint16;
    description
      "Supported TPM_ALG_ID values for the TPM in question.
       Will include ComponentIndex soon."
  }
  list tpms {
    key tpm_name;
    uses tpm-name;
    description
      "A list of TPMs in this composite
device that rats can be conducted with.

list certificates {
    description "The TPM’s endorsement-certificate."
    container certificate {
        leaf certificate-name {
            type string;
            description "An arbitrary name for this identity certificate or certificate chain."
        }
        leaf certificate-type {
            type enumeration {
                enum endorsement-cert {
                    value 0;
                    description "EK Cert type."
                }
                enum attestation-cert {
                    value 1;
                    description "AK Cert type."
                }
            }
            description "Type of this certificate"
        }
        leaf certificate-value {
            type ietfct:end-entity-cert-cms;
            description "The binary signed public endorsement key (EK), attestation key(AK) and corresponding assertions (EK,AK Certificate). In a TPM 2.0 the EK,AK Certificate resides in a well-defined NVRAM location by the TPM vendor."
        }
    }
}

list compute-nodes {
    key node-name;
    uses compute-node;
    description "A list names of hardware components in this composite device that rats can be conducted with."
}
3. IANA considerations

This document will include requests to IANA:

To be defined yet.

4. Security Considerations

There are always some.

5. Acknowledgements

Not yet.

6. Change Log

Changes from version 00 to version 01:

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

7. References

7.1. Normative References

[I-D.birkholz-rats-reference-interaction-model]
7.2. Informative References

[I-D.birkholz-attestation-terminology]

Authors’ Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Michael Eckel
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: michael.eckel@sit.fraunhofer.de

Shwetha Bhandari
Cisco Systems

Email: shwethab@cisco.com

Bill Sulzen
Cisco Systems

Email: bsulzen@cisco.com
Eric Voit  
Cisco Systems  
Email: evoit@cisco.com

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

Tom Laffey  
Hewlett Packard Enterprise  
Email: tom.laffey@hpe.com

Guy C. Fedorkow  
Juniper Networks  
10 Technology Park Drive  
Westford, Massachusetts  01886  
Email: gfedorkow@juniper.net
Abstract

This document defines a standardized information model (IM) for assertions that can be used in remote attestation procedures (RATS). The information elements defined include attestation assertions which provide information about system components characteristics, as well as commonly used attributes and attribute structures that are required by protocols facilitating remote attestation.

Status of This Memo

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

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

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

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

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
Remote attestation procedures (RATS) are used to increase the trust in the trustworthiness of an attester. This is typically accomplished by conveying attestation evidence from an attester to a verifier that is able to appraise the evidence. The exact definitions of RATS roles, such as an attester or a verifier, are specified in the RATS architecture [I-D.birkholz-rats-architecture]. This document defines the common information elements (IE) that are able to express the characteristics of an attester. Ultimately, these IE can be used to compose attestation evidence (attestation assertions that are accompanied by a proof of their validity).

In general, RATS convey information elements that:

- enable the functionality of remote attestation protocols,
- are able to express assertions about an attester’s composition, configuration, or operational state,
- represent the provenance of assertions, including entities that provide assertions on behalf of the attester,
- compose a type of proof of validity with respect to other assertions, and that
- are either verifiable (via comparison with trusted reference values) or non-verifiable.
1.1. Document Structure

Every information element listed is annotated with one or more of these attributes:

Protocol (P): This IE is used on a remote attestation protocol layer, typically on the control plane or as protocol-specific data plane content.

Hardware (H): This IE expresses characteristics about an attester’s hardware components or the composition of its hardware components.

Software (S): This IE expresses characteristics about an attester’s software components or their semantic relationship. The term software component - in the scope of this document - subsumes firmware, bootloader, BIOS/(U)EFI, and microcode.

Operational State (O): This IE is used to convey information about the combination of applied configuration and system state as defined in [RFC8342].

Verifiable (V): This IE requires reference integrity measurements (RIM), compliance-policy, certification-path, or another type of trust-chain in order to be appraised appropriately by a verifier.

Additionally, every IE definition includes a reference to the source of its definition, if it is not specified in this document for the first time (which is the most likely case). If a source of a definition is not a specification or (proposed) standard, but a draft, a web resource, or source that cannot be attribute with a DOI or ISSN, the following attribute is associated.

Unstable (U): The source of the definition of this IE may change in the future and is not considered to be stable at the time of publication of this document.

Information elements might reference other information elements or have to be associated in a set (with or without a specific order) in order to convey the intended meaning to a verifier. Reference to other IE inside this documents simply use their name as reference. In consequence, an IE can be a superstructure composed of other IE with its own name (and potentially additional definition text that defines its purpose and or usage).

The RATS Information Model allows for expressing a hierarchical taxonomy. If an IE is a specialisation of another IE, the last sentence in the definition includes a "This IE is a specialization of _IE NAME_."
The ordering of IE is in descending alphabetical order; independent of source or semantic relationship to other IE, or other types of hierarchy.

2. RATS Information Elements

Age: The latency between the creation of an assertion value (e.g. by asserters such as a hardware sensors or the Linux Integrity Measurement Architecture) including its composition into attestation evidence and its following conveyance to another RATS Actor/Role in RATS. The Age IE does not require a threshold at which point another information element is considered "old" and an age information element has to be included.

Reference: [I-D.ietf-rats-eat]

Assertion Selection: [P]

A filter expression that enables the conveyance of a subset of all attestation assertions available to the attester, if requested by a verifier.

Attestation Evidence: [H, S, O, V]

A composite IE that must include at least an Authentication-Secret Identifier, an Attester Identity, and at least one Attestation Assertion. Attestation Evidence is always signed via the Authentication Secret and thereby binds the listed information elements cryptographically. Attestation Evidence can only be trusted by a verifier if it is associated with a trust anchor the verifier also trusts.

Attestor Identifier: [P, O, V]

A value associated or bound to a distinguishable attester that is intended to uniquely identify it, but is not directly associated with a trust anchor. Additional Endorsement Documents can increase the level of confidence in an Attestor Identifier.

Attestor Identity: [P, S, V]

A document about a distinguishable attester issued and signed by a third party. If not cryptographically associated with a trust anchor directly or indirectly, this IE is a specialization of Attestor Identifier.

Attestation Result: [P]
A set of one or more values that are created by an appraisal action of a verifier. Attestation Result is the most generic definition of the output of RATS and are typically consumed by relying parties.

**Authentication-Secret Identifier: [O, V]**

An identifier that is associated with an authentication secret used to sign attestation evidence.

**Authorization Challenge: [P]**

The input to an challenge-response protocol hand-shake. This IE can be Nonce, but also the output of a local attestation procedure.

Reference [I-D.tschofenig-rats-psa-token]

**Endorsement Document: [P, H, S, V]**

A document about the capabilities and functionality of one or more sub-components of a distinguishable attester issued and signed by a third party. Endorsement Documents are intended to render Attestation Evidence trustworthy. If not cryptographically associated with a trust anchor directly or indirectly, this IE is a specialization of System Component Identifier.

**Location:** A global standardized set of coordinates and related attributes representing the geographic position of a device based on a geodetic system, such as Navstar GPS. The coordinate values can have different meaning with respect to the geographic position of a device depending on the geodetic system used. The default is WGS-84.

The basic location attributes include: latitude, longitude, altitude, accuracy, altitude accuracy, heading, and velocity.

Reference [I-D.ietf-rats-eat]

**Measured Boot Characteristics: [H, S, V]**

If every piece of software is measured by a root-of-trust for measuring during boot time and across staged computing contexts (e.g. UEFI, Bootloader, Kernel, Rich OS), associated information about how and in which operational states these measurements are conducted is vital to RATS. This IE represents several states of a (composite) device with respect to measured boot (previously often called secure boot) including: "Secure Boot Enabled", "Debug
Disabled", "Debug Disabled Since Boot", "Debug Permanent Disable", "Debug Full Permanent Disable".

Nonce:  [P]

An information element with two major uses: the prevention of replay-attacks and as an IE that can be used in a challenge-response interaction model. It is created by the requester to provide evidence about the freshness of the corresponding response. It is important to highlight that a nonce by itself does not protect from relay-attacks.

OEM Identifier:  [H, S, V]

A organizationally unique identifier (OUI) assigned by the IEEE Registration Authority (IEEE RA). This IE is associated with a device or a distinguishable sub-component of a composite device with its own computing context. It intended to identify a device(component) during its life-cycle. This is a specialization of System Component Identifier.

Reference [I-D.ietf-rats-eat]

Origination:  [P, S, V]

An IE representing attestation provenance. Attestation Assertions or Attestation Evidence are produced by a specific source of information that is intended to be uniquely identifiable. The source of information is a distinguishable computing context (see [I-D.birkholz-rats-architecture]) of a device or the sub-components of a composite device.

Reference [I-D.ietf-rats-eat]

Universal Entity ID:  [P, H, V]

A unique identifier permanently associated with an individual manufactured entity / device, such as a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. This IE is intended to either identify an device or a submodule or subsystem of a device. 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. This IE is a specialization of System Component Identifier.

Reference [I-D.ietf-rats-eat]

Uptime:  [H, S]
An IE representing the number of seconds since the first computing context of a (composite) device is able to measure it.

Reference [I-D.ietf-rats-eat]

Security Level: [H, S, V]

A level of confidence with respect to the resilience against attacks intended to compromise attestation evidence. A Security Level can be associated with an Origination. This IE is context specific and requires a scope-specific definition of values as part of a security framework. The [I-D.ietf-rats-eat] document, for example, provides an enumeration of security levels that is similar to the Metadata Service defined by the Fast Identity Online (FIDO) Alliance.

Reference [I-D.ietf-rats-eat]

Software Component Identifier: [S, V]

An IE representing one or more distinguishable Software Components [I-D.ietf-sacm-terminology] that were loaded and measured by an appropriate root-of-trust. The use of this IE typically requires the use of Measured Boot.

Reference [I-D.tschofenig-rats-psi-token]

System Component Identifier: [H, S, V]

An identifier intended to uniquely identify a distinguishable system component. System components can be hardware components or software components (e.g. a virtual machine). The system component can be an "atomic" device (i.e. a composite device with only one hardware component) or a part of a composite device.

Timestamp: [P, S]

A generic information element that represents a certain point of time in the past. The level of confidence in the value of a timestamp is based on the trustworthiness of the source of time, which can be local or remote, a composite of multiple time sources to represent the state synchronization, as well as the precision and the accuracy of the source of time itself.

Timestamps can be time-zone specific and therefore change their meaning if the definition of time zones changes.

Verification Service Indicator: [P, S, V]
This IE provides a hint (typically consumed by a Relying Party) that enables the
discovery of an appropriate Verification Service or Remote Attestation Service, e.g. a URL.

Reference [I-D.tschofenig-rats-psi-token]

3. Security Considerations

   Probably none

4. Acknowledgments

   TBD

5. Change Log

   Initial version -00

6. Contributors

   TBD

7. References

   7.1. Normative References

      [I-D.birkholz-rats-architecture]
      Birkholz, H., Wiseman, M., Tschofenig, H., and N. Smith,
      "Architecture and Reference Terminology for Remote
      Attestation Procedures", draft-birkholz-rats-architecture-01 (work in progress), March 2019.

      [I-D.birkholz-rats-basic-yang-module]
      Birkholz, H., Eckel, M., Bhandari, S., Sulzen, B., Voit, E., and G. Fedorkow,
      "YANG Module for Basic Challenge-Response-based Remote Attestation Procedures",
      draft-birkholz-rats-basic-yang-module-00 (work in progress), March 2019.

      [I-D.birkholz-rats-tuda]
      Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann,
      "Time-Based Uni-Directional Attestation",
      draft-birkholz-rats-tuda-00 (work in progress), March 2019.

      [I-D.ietf-rats-eat]
      Mandyam, G., Lundblade, L., Ballesteros, M., and J. O’Donoghue,
      "The Entity Attestation Token (EAT)",
      draft-ietf-rats-eat-01 (work in progress), July 2019.
7.2. Informative References

[I-D.birkholz-rats-reference-interaction-model]

[I-D.ietf-sacm-terminology]

[I-D.richardson-rats-usecases]

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

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

Email: michael.eckel@sit.fraunhofer.de
Abstract

This document defines an interaction model for a basic remote attestation procedure. Additionally, the required information elements are illustrated.

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Remote attestation procedures (RATS) are a combination of activities, in which a Verifier creates assertions about assertions of integrity and about characteristics of other system entities by the appraisal of corresponding signed assertions (evidence). In this document, a reference interaction model for a generic challenge-response-based remote attestation procedure is provided. The minimum set of components, roles and information elements that have to be conveyed between Verifier and Attester are defined as a standard reference to derive more complex RATS from.

1.1. Requirements notation

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

The term "Remote Attestation" is a common expression and often associated with certain properties. The term "Remote" in this context does not necessarily refer to a remote system entity in the scope of network topologies or the Internet. It rather refers to a decoupled system or different computing context, which also could be present locally as components of a composite device. Examples include: a Trusted Execution Environment (TEE), Baseboard Management Controllers (BMCs), as well as other physical or logical protected/isolated execution environments.

3. Scope

This document focuses on a generic interaction model between Verifiers and Attesters. Complementary processes, functions and activities that are required for a complete semantic binding of RATS are not in scope. Examples include: identity establishment, key enrollment, and certificate revocation. Furthermore, any processes and activities that go beyond carrying out the remote attestation process are out of scope. For instance, using the result of a remote attestation that is emitted by the Verifier, such as triggering remediation actions and recovery processes, as well as the remediation actions and recovery processes themselves, are out of scope.

4. Component Roles

The Reference Interaction Model for Challenge-Response-based Remote Attestation is based on the standard roles defined in [I-D.birkholz-rats-architecture]:

Attester: The role that designates the subject of the remote attestation. A system entity that is the provider of evidence takes on the role of an Attester.

Verifier: The role that designates the system entity and that is the appraiser of evidence provided by the Attester. A system entity that is the consumer of evidence takes on the role of a Verifier.

5. Prerequisites

Attester Identity:

Attestation Authenticity: An Attestation MUST be authentic.

An attestation, in order to be authentic, MAY This Identity MUST be part of the signed assertions (attestation evidence) that the
Attester conveys to the Verifier. An Identity MAY be a unique identity or it MAY be included in a zero-knowledge proof (ZKP) or be part of a group signature.

Authentication Secret: An Authentication Secret MUST be present on the Attester. The Attester MUST sign assertions with that Authentication Secret, proving the authenticity of the assertions. The Authentication Secret MUST be established before a remote attestation procedure can take place. How it is established is out of scope for this reference model.

6. Remote Attestation Interaction Model

This section defines the information elements that have to be conveyed via a protocol, enabling the conveyance of Evidence between Verifier and Attester, as well as the interaction model for a generic challenge-response remote attestation scheme.

6.1. Information Elements

Attester Identity (‘attesterIdentity’): _mandatory_

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

Authentication Secret ID (‘authSecID’): _mandatory_

An identifier that MUST be associated with the Authentication Secret which is used to sign evidence.

Nonce (‘nonce’): _mandatory_

The Nonce (number used once) is intended to be unique and practically infeasible to guess. In this reference interaction model the Nonce MUST be provided by the Verifier and MUST be used as proof of freshness. With respect to conveyed evidence, it ensures the result of an attestation activity to be created recently, e. g. sent or derived by the challenge from the Verifier. As such, the Nonce MUST be part of the signed Attestation Evidence that is sent from the Attester to the Verifier.

Assertions (‘assertions’): _mandatory_

Assertions represent characteristics of an Attester. They are required for proving the integrity of an Attester. Examples are assertions about sensor data, policies that are active on the
system entity, versions of composite firmware of a platform, running software, routing tables, or information about a local time source.

Reference Assertions ('refAssertions') _mandatory_

Reference Assertions are used to verify the assertions received from an Attester in an attestation verification process. For example, Reference Assertions MAY be Reference Integrity Measurements (RIMs) or assertions that are implicitly trusted because they are signed by a trusted authority. RIMs represent (trusted) assertions about the intended platform operational state of the Attester.

Assertion Selection ('assertionSelection'): _optional_

An Attester MAY provide a selection of assertions in order to reduce or increase retrieved assertions to those that are relevant to the conducted appraisal. Usually, all available assertions that are available to the Attester SHOULD be conveyed. The Assertion Selection MAY be composed as complementary signed assertions or MAY be encapsulated assertions in the signed Attestation Evidence. An Attester MAY decide whether or not to provide all requested assertions or not. An example for an Assertion Selection is a Verifier requesting (signed) RIMs from an Attester.

(Signed) Attestation Evidence ('signedAttestationEvidence'): _mandatory_

Attestation Evidence consists of the Authentication Secret ID that identifies an Authentication Secret, the Attester Identity, the Assertions, and the Verifier-provided Nonce. Attestation Evidence MUST cryptographically bind all of those elements. The Attestation Evidence MUST be signed by the Authentication Secret. The Authentication Secret MUST be trusted by the Verifier as authoritative.

Attestation Result ('attestationResult'): _mandatory_

An Attestation Result is produced by the Verifier as a result of a Verification of Attestation Evidence. The Attestation Result represents assertions about integrity and other characteristics of the corresponding Attester.
6.2. Interaction Model

The following sequence diagram illustrates the reference remote attestation procedure defined by this document.

```plaintext
[Attester]                                                      [Verifier]

|                                                               |
| <--- requestAttestation(nonce, authSecID, assertionSelection) |
|                                                               |
| collectAssertions(assertionSelection)                         |
| => assertions                                                 |
|                                                               |
| signAttestationEvidence(authSecID, assertions, nonce)        |
| => signedAttestationEvidence                                 |
|                                                               |
| signedAttestationEvidence ----------------------------------> |
|                                                               |
| verifyAttestationEvidence(signedAttestationEvidence, refAssertions)
| attestationResult <=                                       |
```

The remote attestation procedure is initiated by the Verifier, sending an attestation request to the Attester. The attestation request consists of a Nonce, a Authentication Secret ID, and an Assertion Selection. The Nonce guarantees attestation freshness. The Authentication Secret ID selects the secret with which the Attester is requested to sign the Attestation Evidence. The Assertions Selection narrows down or increases the amount of received Assertions, if required. If the Assertions Selection is empty, then by default all assertions that are available on the system of the Attester SHOULD be signed and returned as Attestation Evidence. For example, a Verifier may only be interested in particular information about the Attester, such as proof of with which BIOS and firmware it booted up, and not include information about all currently running software.

The Attester, after receiving the attestation request, collects the corresponding Assertions to compose the Attestation Evidence that the Verifier requested—or, in case the Verifier did not provide an Assertions Selection, the Attester collects all information that can be used as complementary Assertions in the scope of the semantics of the remote attestation procedure. After that, the Attester produces Attestation Evidence by signing the Attester Identity, the Assertions, and the Nonce with the Authentication Secret identified by the Authentication Secret ID. Then the Attester sends the signed Attestation Evidence back to the Verifier.
Important at this point is that Assertions, the Nonce as well as the Attester Identity information MUST be cryptographically bound to the signature of the Attestation Evidence. It is not required for them to be present in plain text, though. Cryptographic blinding MAY be used at this point. For further reference see Security and Privacy Considerations (Section 8)

As soon as the Verifier receives the signed Attestation Evidence, it verifies the signature, the Attester Identity, the Nonce, and the Assertions. This process is application-specific and can be carried out by, e. g., comparing the Assertions to known (good), expected Reference Assertions, such as Reference Integrity Measurements (RIMs), or evaluating it in other ways. The final output of the Verifier is the Attestation Result. It constitutes an new assertion about properties and characteristics of the Attester, i. e. whether or not it is compliant to policies, or even can be "trusted".

7. Further Context

Depending on the use cases to cover, there may be additional requirements. Some of them are mentioned in this section.

7.1. Confidentiality

Confidentiality of exchanged attestation information may be desirable. This requirement usually is present when communication takes place over insecure channels, such as the public Internet. In such cases, TLS may be uses as a suitable communication protocol that preserves confidentiality. In private networks, such as carrier management networks, it must be evaluated whether or not the transport medium is considered confidential.

7.2. Mutual Authentication

In particular use cases mutual authentication may be desirable in such a way that a Verifier also needs to prove its identity to the Attester, instead of only the Attester proving its identity to the Verifier.

7.3. Hardware-Enforcement/Support

Depending on the requirements, hardware support for secure storage of cryptographic keys, crypto accelerators, or protected or isolated execution environments may be useful. Well-known technologies are Hardware Security Modules (HSM), Physically Unclonable Functions (PUFs), Shielded Secrets, and Trusted Executions Environments (TEEs).
8. Security and Privacy Considerations

In a remote attestation process the Verifier or the Attester MAY want to cryptographically blind several attributes. For instance, information can be part of the signature after applying a one-way function (e.g. a hash function).

There is also a possibility to scramble the Nonce or Attester Identity with other information that is known to both the Verifier and Attester. A prominent example is the IP address of the Attester that usually is known by the Attester itself as well as the Verifier. This extra information can be used to scramble the Nonce in order to counter certain types of relay attacks.

9. Acknowledgments

Very likely.

10. Change Log

- Initial draft -00
- Changes from version 00 to version 01:
  * Added details to the flow diagram
- Changes from version 01 to version 02:
  * Integrated comments from Ned Smith (Intel)
  * Reorganized sections and
  * Updated interaction model
- Changes from version 02 to version 03:
  * Replaced "claims" with "assertions"
  * Added proof-of-concept CDDL for CBOR via CoAP based on a TPM 2.0 quote operation

11. References

11.1. Normative References
11.2. Informative References

[I-D.birkholz-rats-architecture]
Birkholz, H., Wiseman, M., Tschofenig, H., and N. Smith,
draft-birkholz-rats-architecture-01 (work in progress), March 2019.

Appendix A. CDDL Specification for a simple CoAP Challenge/Response Interaction

The following CDDL specification is an examplary proof-of-concept to illustrate a potential implementation of the Reference Interaction Model. The transfer protocol used is CoAP using the FETCH operation. The actual resource operated on can be empty. Both the Challenge Message and the Response Message are exchanged via the FETCH Request and FETCH Response body.

In this example, the root-of-trust for reporting primitive operation "quote" is provided by a TPM 2.0.
RAIM-Bodies = CoAP-FETCH-Body / CoAP-FETCH-Response-Body

CoAP-FETCH-Body = [ hello: bool, ; if true, the AK-Cert is conveyed
    nonce: bytes,
    pcr-selection: [ + [ tcg-hash-alg-id: uint .size 2, ; TPM2_A
      LG_ID
        [ + pcr: uint .size 1 ],
        ]
    ],
    ]

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

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

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

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt  64295
Germany

Email: henk.birkholz@sit.fraunhofer.de
Network Device Attestation Workflow
draft-fedorkow-rats-network-device-attestation-00

Abstract

This document describes a workflow for network device attestation.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 1, 2020.

Copyright Notice

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

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

There are many components to consider in fielding a trusted computing device, from operating systems to applications. Part of that is a trusted supply chain, where manufacturers can certify that the product they intended to build is actually the one that was installed at a customer’s site.

Attestation is defined here as the process of creating, conveying and appraising assertions about Platform trustworthiness characteristics,
including Roots of Trust, supply chain trust, identity, platform provenance, shielded locations, protected capabilities, software configuration, hardware configuration, platform composition, compliance to test suites, functional and assurance evaluations, etc.

The supply chain itself has many elements, from validating suppliers of electronic components, to ensuring that shipping procedures protect against tampering through many stages of distribution and warehousing. One element that helps maintain the integrity of the supply chain after manufacturing is Attestation.

Within the Trusted Computing Group context, attestation is the process by which an independent Verifier can obtain cryptographic proof as to the identity of the device in question, evidence of the integrity of software loaded on that device when it started up, and then verify that what’s there is what’s supposed to be there. For networking equipment, a verifier capability can be embedded in a Network Management Station (NMS), a posture collection server, or other network analytics tool (such as a software asset management solution, or a threat detection and mitigation tool, etc.). While informally referred to as attestation, this document focuses on a subset defined here as Remote Integrity Verification (RIV). RIV takes a network equipment centric perspective that includes a set of protocols and procedures for determining whether a particular device was launched with untampered software, starting from Roots of Trust. While there are many ways to accomplish attestation, RIV sets out a specific set of protocols and tools that work in environments commonly found in Networking Equipment. RIV does not cover other platform characteristics that could be attested, although it does provide evidence of a secure infrastructure to increase the level of trust in other platform characteristics attested by other means.

This profile outlines the RIV problem, and then identifies components that are necessary to get the complete attestation procedure working in a scalable solution using commercial products.

This document focuses primarily on software integrity verification using the Trusted Platform Module (TPM) as a root of trust.

1.1. Requirements Language

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

The RIV attestation workflow outlined in this document is intended to meet the following high-level goals:

- **Provable Device Identity** - The ability to identify a device using a cryptographic identifier is a critical prerequisite to software inventory attestation.

- **Software Inventory** - A key goal is to identify the software release installed on the device, and to provide evidence of its integrity.

- **Verification** - Verification of software and configuration of the device shows that the software that’s supposed to be installed on there actually has been launched, without unauthorized modification.

This document itself is non-normative; the document does not define protocols, but rather identifies protocols that can be used together to achieve the goals above, and in some cases, highlights gaps in existing protocols.

1.3. Problem Description

RIV is a procedure that assures a network operator that the equipment on their network can be reliably identified, and that untampered software of a known version is installed on each endpoint. In this context, endpoint might include the conventional endpoints like servers and laptops, but also network equipment itself, such as routers, switches and firewalls.

RIV can be viewed as a link in a trusted supply chain, and includes three major processes:

- **Creation of Evidence** is the process whereby an endpoint generates cryptographic proof (evidence) of claims about platform properties. In particular, the platform identity and its software configuration are of critical importance.
**Platform Identity refers to the mechanism assuring the**
**attestation verifier (typically a network administrator)**
**that the equipment on their network can be reliably identified,**
**and that its manufacturer is certified by a trusted authority.**
**This certification provides the verifier with assurance that**
**the Root of Trust elements of the device were verified by the**
**manufacturer before the device was shipped.**

**Software used to boot a platform can be described as a chain**
**of measurements, starting from a Root of Trust for Measurement,**
**that normally ends when the system software is loaded.**
**Measurement records the identity, integrity and version of each**
**software component registered with the TPM, so that the**
**subsequent appraisal stage can determine whether the software**
**installed is authentic and free of tampering. Clearly the second**
**part of the problem, attesting the state of mutable components**
**of a given device, is of little value without the first part,**
**reliable identification of the device in question. By the**
**same token, unambiguous identity of a device is necessary,**
**but is insufficient to assure the operator of the provenance**
**of the device through the supply chain, or that the device is**
**configured to behave properly.**

- **Conveyance of Evidence** is the process of reliably transporting evidence from an endpoint to an appraiser/verifier, e.g. a management station. The transport is typically carried out via a management network. The channel must provide integrity and authenticity, and, in some use cases, may also require confidentiality.

- **Appraisal of Evidence** is the process of verifying the evidence received by a verifier/appraiser from a device, and using verified evidence to inform decision making. In this context, verification means comparing the device characteristics reported as evidence with the configuration expected by the system administrator. This step can work only when there is a way to express what should be there, often referred to as golden measurements, or Reference Integrity Measurements, representing the intended configured state of an endpoint.

As a part of a trusted supply chain, RIV attestation provides two important benefits:

- **Platform Identity** is the mechanism providing trusted identity can reassure network managers that the specific devices they ordered from authorized manufacturers for attachment to their network are the ones that were installed, and that they continue to be present in their network. As part of the mechanism for Platform Identity,
cryptographic proof of the identity of the manufacturer is also provided.

- Software Configuration is the mechanism that reports the state of mutable components on the device can assure network managers that they have known, untampered software configured to run in their network.

An implementation of RIV requires three technologies

1. Identity: Platform identity can be based on IEEE 802.1AR Device Identity [IEEE-802-1AR], coupled with careful supply-chain management by the manufacturer. The DevID certificate contains a statement by the manufacturer that establishes the provenance of the device as it left the factory. Some applications with a more-complex post-manufacture supply chain (e.g. Value Added Resellers), or with privacy concerns, may want to use an alternate mechanism for platform authentication based on TCG Platform Certificates [Platform-Certificates].

2. Platform Attestation provides evidence of configuration of software elements throughout the product lifecycle. This form of attestation can be implemented with TPM PCR, Quote and log mechanisms, which provide an authenticated mechanism to report what software actually starts up on the device each time it reboots. Note that the TPM requires separate keys for identity (DevID) and attestation (PCR Quotes0) (see Section 2.2).

3. Reference Integrity Measurements must be conveyed from the software authority (often the manufacturer for embedded systems) to the system in which verification will take place

Network operators benefit from a trustworthy attestation mechanism that provides assurance that their comprises authentic equipment, and has loaded software free of known vulnerabilities and unauthorized tampering.

1.4. Solution Requirements

An Attestation solution must meet a number of requirements to make it simple to deploy at scale.

1. Easy to Use - This solution should work "out of the box" as far as possible, that is, with the fewest possible steps needed at the end-user’s site. Eliminate complicated databases or provisioning steps that would have to be executed by the owner of a new device. Network equipment is often required to "self-configure", to reliably reach out without manual intervention to
prove its identity and operating posture, then download its own configuration. See [RFC8572] for an example of Secure Zero Touch Provisioning.

2. Multi-Vendor - This solution should identify standards-based interfaces that allow attestation to work with attestation-capable devices and verifiers supplied by different vendors in one network.

3. Scalable - The solution must not depend on choke points that limit the number of endpoints that could be evaluated in one network domain.

4. Extensible - A network equipment attestation solution needs to expand over time as new features are added. The solution must allow new features to be added easily, providing for a smooth transition and allowing newer and older architectural components to continue to work together. Further, a network equipment attestation solution and the specifications referenced here must define safe extensibility mechanisms that enable innovation without breaking interoperability.

5. Efficient - A network equipment attestation solution should, to the greatest extent feasible, continuously monitor the health and posture status of network devices. Posture measurements should be updated in real-time as changes to device posture occur and should be published to remote integrity validators. Validation reports should also be shared with their relying parties (for example, network administrators, or network analytics that rely on these reports for posture assessment) as soon as they are available.

1.5. Scope

This document includes a number of assumptions to limit the scope:

- This solution is for use in non-privacy-preserving applications (for example, networking, Industrial IoT), avoiding the need for a Privacy Certificate Authority for attestation keys

- This document applies primarily to "embedded" applications, where the device manufacturer ships the software image for the device.

- The approach outlined in this document assumes a physical TPM.
1.5.1. Out of Scope

- Run-Time Attestation: Run-time attestation of Linux or other multi-threaded operating system processes considerably expands the scope of the problem. Many researchers are working on that problem, but this document defers the run-time attestation problem.

- Multi-Vendor Embedded Systems: Additional coordination would be needed for devices that themselves comprise hardware and software from multiple vendors, integrated by the end user.

- Processor Sleep Modes: Embedded equipment typically does not "sleep", so sleep and hibernate modes are not considered.

- Virtualization and Containerization: These technologies are increasingly used in embedded systems, but are not considered in this revision of the document.

1.5.2. Why Remote Integrity Verification?

Remote Integrity Verification can go a long way to solving the "Lying Endpoint" problem, in which malicious software on an endpoint may both subvert the intended function, and also prevent the endpoint from reporting its compromised status. Man-in-the Middle attacks are also made more difficult through a strong focus on device identity Attestation data can be used for asset management, vulnerability and compliance assessment, plus configuration management.

1.5.3. Network Device Attestation Challenges

There have been demonstrations of attestation using TPMs for years, accompanied by compelling security reasons for adopting attestation. Despite this, the technology has not been widely adopted, in part, due to the difficulties in deploying TPM-based attestation. Some of those difficulties are:

- Standardizing device identity. Creating and using unique device identifiers is difficult, especially in a privacy-sensitive environment. But attestation is of limited value if the operator is unable to determine which devices pass attestation validation tests, and which fail. This problem is substantially simplified for infrastructure devices like network equipment, where identity can be explicitly coded using IEEE 802.1AR, but doing so relies on adoption of 802.1AR [IEEE-802-1AR] by manufacturers and hardware system integrators.
o Standardizing attestation representations and conveyance. Interoperable remote attestation has a fundamental dependence on vendors agreeing to a limited set of network protocols for communicating attestation data. Network device vendors will be slow to adopt the protocols necessary to implement remote attestation without a fully-realized plan for deployment.

o Interoperability. Networking equipment operates in a fundamentally multi-vendor environment, putting additional emphasis on the need for standardized procedures and protocols.

o Attestation evidence is complex. Operating systems used in larger embedded devices are often multi-threaded, so the order of completion for individual processes is non-deterministic. While the hash of a specific component is stable, once extended into a PCR, the resulting values are dependent on the (non-deterministic) ordering of events, so there will never be a single known-good value for some PCRs. Careful analysis of event logs can provide proof that the expected modules loaded, but it’s much more complicated than simply comparing hashes.

o Software configurations can have seemingly infinite variability. This problem is nearly intractable on PC and Server equipment, where end users have unending needs for customization and new applications. However, embedded systems, like networking equipment, are often simpler, in that there are fewer variations and releases, with vendors typically offering fewer options for mixing and matching.

o Software updates can be complex. Even the most organized network operator may have many different releases in their network at any given time, with the result that there’s never a single digest or fingerprint that indicates the software is "correct"; digests formed by hashing software modules on a device can only show the correct combination of versions for a specific device at a specific time.

None of these issues are insurmountable, but together, they’ve made deployment of attestation a major challenge. The intent of this document is to outline an attestation profile that’s simple enough to deploy, while yielding enough security to be useful.

1.5.4. Why is OS Attestation Different?

Even in embedded systems, adding Attestation at the OS level (e.g. Linux IMA, Integrity Measurement Architecture [IMA]) increases the number of objects to be attested by one or two orders of magnitude,
involves software that’s updated and changed frequently, and introduces processes that complete in unpredictable order.

TCG and others (including the Linux community) are working on methods and procedures for attesting the operating system and application software, but standardization is still in process.

2. Solution Outline

2.1. 2.1 RIV Software Configuration Attestation using TPM

RIV Attestation is a process for determining the identity of software running on a specifically-identified device. Remote Attestation is broken into two phases, shown in Figure 1:

- During system startup, measurements (i.e., hashes computed as fingerprints of files) are extended, or cryptographically folded, into the TPM. Entries are also added to an informational log. The measurement process generally follows the Chain of Trust model used in Measured Boot, where each stage of the system measures the next one before launching it.

- Once the device is running and has operational network connectivity, a separate, trusted server (called a Verifier in this document) can interrogate the network device to retrieve the logs and a copy of the digests collected by hashing each software object, signed by an attestation private key known only to the TPM.

The result is that the Verifier can verify the device’s identity by checking the certificate corresponding to the TPM’s attestation private key, and can validate the software that was launched by comparing digests in the log with known-good values, and verifying their correctness by comparing with the signed digests from the TPM.

It should be noted that attestation and identity are inextricably linked; signed evidence that a particular version of software was loaded is of little value without cryptographic proof of the identity of the device producing the evidence.
In Step 1, measurements are "extended" into the TPM as processes start. In Step 2, signed PCR digests are retrieved from the TPM for offbox analysis after the system is operational.

Figure 1: TCG Attestation Model

2.2. RIV Keying

TPM 1.2 and TPM 2.0 have a variety of rules separating the functions of identity and attestation, allowing for use-cases where software configuration must be attested, but privacy must be maintained.

To accommodate these rules in an environment where device privacy is not normally a requirement, the TCG Guidance for Securing Network Equipment [NetEq] suggests using separate keys for Identity (i.e., DevID) and Attestation (i.e., signing a quote of the contents of the PCRs).

In this case, the device manufacturer should provision an Initial Attestation Key (IAK) and x.509 certificate that parallels the IDevID, with the same device ID information as the IDevID certificate (i.e., the same Subject Name and Subject Alt Name, even though the key pairs are different). This allows a quote from the device, signed by the IAK, to be linked directly to the device that provided it, by examining the corresponding IAK certificate.
Inclusion of an IAK by a vendor does not preclude a mechanism whereby an Administrator can define Local Attestation Keys (LAKs) if desired.

2.3. RIV Information Flow

RIV workflow for networking equipment is organized around a simple use-case, where a network operator wishes to verify the integrity of software installed in specific, fielded devices. This use-case implies several components:

1. A Device (e.g. a router or other embedded device, also known as an Attester) somewhere and the network operator wants to examine its boot state.

2. A Verifier (which might be a network management station) somewhere separate from the Device that will retrieve the information and analyze it to pass judgement on the security posture of the device.

3. A Relying Party, which has access to the Verifier to request attestation and to act on results.

4. This document assumes that signed Reference Integrity Measurements (RIMs) (aka "golden measurements") can either be created by the device manufacturer and shipped along with the device as part of its software image, or alternatively, could be obtained a number of other ways (direct to the verifier from the manufacturer, from a third party, from the owner’s observation of what’s thought to be a "known good system", etc.). Retrieving RIMs from the device itself allows attestation to be done in systems which may not have access to the public internet, or by other devices that are not management stations per-se (e.g., a peer device). If reference measurements are obtained from multiple sources, the Verifier may need to evaluate the relative level of trust to be placed in each source in case of a discrepancy.

These components are illustrated in Figure 2.

A more-detailed taxonomy of terms is given in [I-D.birkholz-rats-architecture]
In Step 0, The Asserter (the device manufacturer) provides a Software Image accompanied by Reference Integrity Measurements (RIMs) to the Attester (the device under attestation) signed by the asserter. In Step 1, the Verifier (Network Management Station), on behalf of a Relying Party, requests Identity, Measurement Values (and possibly RIMs) from the Attester. In Step 2, the Attester responds to the request by providing a DevID, Quotes (measured values), and optionally RIMs, signed by the Attester.

Figure 2: RIV Reference Configuration for Network Equipment

See Section 3.1.1 for more narrowly defined terms related to Attestation

2.4. RIV Simplifying Assumptions

This document makes the following simplifying assumptions to reduce complexity:

- The product to be attested is shipped with an IEEE 802.1AR DevID and an Initial Attestation Key (IAK) with certificate. The IAK cert contains the same identity information as the DevID (specifically, the same Subject Name and Subject Alt Name, signed by the manufacturer), but it’s a type of key that can be used to sign a TPM Quote. This convention is described in TCG Guidance for Securing Network Equipment [NetEq]. For network equipment, which is generally non-privacy-sensitive, shipping a device with both an IDevID and an IAK already provisioned substantially simplifies initial startup. Privacy-sensitive applications may use the TCG Platform Certificate and additional procedures to install identity credentials on the platform after manufacture. (See Section 2.3.1 below for the Platform Certificate alternative)

- The product is equipped with a Root of Trust for Measurement, Root of Trust for Storage and Root of Trust for Reporting that is

Fedorkow & Fitzgerald-McKExpires January 1, 2020 [Page 13]
capable of conforming to the TCG Trusted Attestation Protocol (TAP) Information Model [TAP].

- The vendor will ship Reference Integrity Measurements (i.e., known-good measurements) in the form of signed CoSWID tags [I-D.ietf-sacm-coswid], [SWID], as described in TCG Reference Integrity Measurement Manifest [RIM].

2.4.1. DevID Alternatives

Some situations may have privacy-sensitive requirements that preclude shipping every device with an Initial Device ID installed. In these cases, the IDevID can be installed remotely using the TCG Platform Certificate [Platform-Certificates].

Some security-sensitive administrators may want to install their own identity credentials to certify platform identity and attestation results. IEEE 802.1AR [IEEE-802-1AR] allows for both Initial Device Identity credentials, installed by the manufacturer, or Local Device Identity credentials installed by the administrator of the platform. TCG TPM 2.0 Keys documents [Platform-DevID-TPM-2.0] and [PC-Client-BIOS-TPM-2.0] specifies analogous Initial and Local Attestation Keys (IAK and LAK), and contains figures showing the relationship between IDevID, LDevID, IAK and LAK keys.

Platform administrators are free to use any number of criteria to judge authenticity of a platform before installing local identity keys, as part of an on-boarding process. The TCG TPM 2.0 Keys document [Platform-DevID-TPM-2.0] also outlines procedures for creating Local Attestation Keys and Local Device IDs (LDevIDs) rooted in the manufacturer’s IDevID as a check to reduce the chances that counterfeit devices are installed in the network.

Note that many networking devices are expected to self-configure (aka Zero Touch Provisioning). Current standardized zero-touch mechanisms such as [RFC8572] assume that identity keys are already in place before network on-boarding can start.

2.4.2. Additional Attestation of Platform Characteristics

The Platform Attribute Credential [Platform-Certificates] can also be used to convey additional information about a platform from the manufacturer or other entities in the supply chain. While outside the scope of RIV, the Platform Attribute Credential can deliver information such as lists of serial numbers for components embedded in a device or security assertions related to the platform, signed by the manufacturer, system integrator or value-added-reseller.
2.4.3. Root of Trust for Measurement

The measurements needed for attestation require that the device being attested is equipped with a Root of Trust for Measurement, i.e., some trustworthy mechanism that can take the first measurement in the chain of trust required to attest that each stage of system startup is verified, and a Root of Trust for Reporting to report the results [Roots-of-Trust].

While there are many complex aspects of a Root of Trust, two aspects that are important in the case of attestation are:

- The first measurement taken by the Root of Trust for Measurement, and stored in the Root of Trust for Storage, is presumed to be correct.
- There must not be a way to reset the RTM without re-entering the RTS code.

The first measurement can’t be checked by a code that’s been previously checked by something further back up the chain (it’s the first, after all); if that first measurement can be subverted, none of the remaining measurements can be trusted. (See [NIST-SP-800-155])

2.4.4. Reference Integrity Measurements (RIMs)

Much of attestation focuses on collecting and transmitting ‘evidence’ in the form of PCR measurements and attestation logs. But the critical part of the process is enabling the verifier to decide whether the measured hashes are "the right ones" or not.

While it must be up to network administrators to decide what they want on their networks, the software supplier should supply the Reference Integrity Measurements, (aka Golden Measurements or "known good" hash digests) that may be used by a verifier to determine if evidence shows known good, known bad or unknown software configurations.

In general, there are two kinds of reference measurements:

1. Measurements of early system startup (e.g., BIOS, boot loader, OS kernel) are essentially single threaded, and executed exactly once, in a known sequence, before any results could be reported. In this case, while the method for computing the hash and extending relevant PCRs may be complicated, the net result is that the software (more likely, firmware) vendor will have one known good PCR value that "should" be present in the PCR after
the box has booted. In this case, the signed reference measurement simply lists the expected hash for the given version.

2. Measurements taken later in operation of the system, once an OS has started (for example, Linux IMA\cite{ima}), may be more complex, with unpredictable "final" PCR values. In this case, the Verifier must have enough information to reconstruct the expected PCR values from logs and signed reference measurements from a trusted authority.

In both cases, the expected values can be expressed as signed CoSWID tags, but the SWID structure in the second case is somewhat more complex. An example of how CoSWIDs could be incorporated into a reference manifest can be found in the IETF Internet-Draft "A SUIT Manifest Extension for Concise Software Identifiers" \cite{suit-coswid-manifest}.

The TCG has done exploratory work in defining formats for reference integrity manifests under the working title TCG Reference Integrity Manifest \cite{rim}.

2.4.5. Attestation Logs

Quotes from a TPM can provide evidence of the state of a device at the time the quote was requested, but to make sense of the quote in most cases an event log of what software modules contributed which values to the quote during startup must also be provided. The log needs not be secured, but it is essential that the logs contain enough information to exactly reconstruct the state of whatever went into the quote (e.g., PCR values).

TCG has defined several event log formats:

- Legacy BIOS event log (TCG PC Client Specific Implementation Specification for Conventional BIOS, Section 11.3\cite{pc-client-bios-tpm-1.2})

- UEFI BIOS event log (TCG EFI Platform Specification for TPM Family 1.1 or 1.2, Section 7 \cite{efi})

- Canonical Event Log \cite{canonical-event-log}

It should be noted that a given device might use more than one event log format (e.g., a UEFI log during initial boot, switching to Canonical Log when the host OS launches).

The TCG SNMP Attestation MIB \cite{snmp-attestation-mib} will support any record-oriented log format, including the three TCG-defined formats,
but it currently leaves figuring out which log(s) are in what format up to the Verifier.

3. Standards Components

3.1. Reference Models

3.1.1. IETF Reference Model for Challenge-Response Remote Attestation

Initial work at IETF defines remote attestation as follows:

- The Reference Interaction Model for Challenge-Response-based Remote Attestation is based on the standard roles defined in [I-D.birkholz-rats-architecture]:

- Attester: The role that designates the subject of the remote attestation. A system entity that is the provider of evidence takes on the role of an Attester.

- Verifier: The role that designates the system entity and that is the appraiser of evidence provided by the Attester. A system entity that is the consumer of evidence takes on the role of a Verifier.

The following diagram illustrates a common information flow between a Verifier and an Attester, specified in [I-D.birkholz-rats-reference-interaction-model]:

```
Attester          Verifier

| <------- requestAttestation(nonce, authSecID, claimSelection) |
| collectAssertions(assertionsSelection) => assertions |
| signAttestationEvidence(authSecID, assertions, nonce) => signedAttestationEvidence |
| signedAttestationEvidence ------------------------------- > |
| verifyAttestationEvidence(signedAttestationEvidence, refassertions) attestationResult <= |
```

Figure 3: IETF Attestation Information Flow
The RIV approach outlined in this document aligns with the RATS reference model.

3.2. RIV Workflow

The overall flow for an attestation session is shown in Figure 4. In this diagram:

- **Step 0**, positioning of the signed reference measurements, may happen in two ways:
  - **Step 0A** below shows a verifier obtaining reference measurements directly from a software configuration authority, whether it’s the vendor or another authority chosen by the system administrator. The reference measurements are signed by the Asserter (i.e., the software configuration authority).
  - Or - **Step 0B**, the reference measurements, signed by the Asserter, may be distributed as part of software installation, long before the attestation session begins. Software installation is usually vendor-dependent, so there are no standards involved in this step. However, the verifier can use the same protocol to obtain the reference measurements from the device as it would have used with an external reference authority.

- **Step 1**, the Verifier initiates an attestation session by opening a TLS session, validated using the DevID to prove that the connection is attesting the right box.

- **Step 2**, measured values are retrieved from the Attester’s TPM using a YANG or SNMP interface that implements the TCG TAP model (e.g., YANG Module for Basic Challenge-Response-based Remote Attestation Procedures [I-D.birkholz-yang-basic-remote-attestation]).

- **Step 3**, the Attester also delivers a copy of the signed reference measurements, using Software Inventory YANG module based on Software Identifiers [I-D.birkholz-yang-swid].
Either CoSWID-encoded reference measurements are signed by a trusted authority and retrieved directly prior to attestation (as shown in Step 0A), or CoSWID-encoded reference measurements are signed by the device manufacturer, installed on the device by a proprietary installer, and delivered during attestation (as shown in Step 0B).

In Step 1, the Verifier initiates a connection for attestation. The Attester’s identity is validated using DevID with TLS. In Step 2, a nonce, quotes (measured values) and measurement log are conveyed via TAP with a protocol-specific binding (e.g. SNMP). Logs are sent in the Canonical Log Format. In Step 3, CoSWID-encoded reference measurements are retrieved from the Attester using the YANG ([I-D.birkholz-yang-swid].

Figure 4: RIV Protocol and Encoding Summary

The following components are used:

1. TPM Keys are configured according to [Platform-DevID-TPM-2.0], [PC-Client-BIOS-TPM-1.2], or [Platform-ID-TPM-1.2]

2. Measurements of bootable modules are taken according to TCG PC Client [PC-Client-BIOS-TPM-2.0] and Linux IMA [IMA]

3. Device Identity is managed by IEEE 802.1AR certificates [IEEE-802-1AR], with keys protected by TPMs.

4. Quotes are retrieved according to TCG TAP Information Model [TAP]

5. Reference Integrity Measurements are encoded as CoSWID tags, as defined in the TCG RIM document [RIM], compatible with NIST IR 8060 [NIST-IR-8060] and the IETF CoSWID draft [I-D.ietf-sacm-coswid]. Reference measurements are signed by the device manufacturer.
3.3. Layering Model for Network Equipment Attester and Verifier

Retrieval of identity and attestation state uses one protocol stack, while retrieval of Reference Measurements uses a different set of protocols. Figure 5 shows the components involved.

---

* IETF Attestation Reference Interaction Diagram *
---
4. Conclusion

TCG technologies can play an important part in the implementation of Remote Integrity Verification. Standards for many of the components needed for implementation of RIV already exist:

- Platform identity can be based on IEEE 802.1AR Device identity, coupled with careful supply-chain management by the manufacturer.
- Complex supply chains can be certified using TCG Platform Certificates [Platform-Certificates]
- The TCG TAP mechanism can be used to retrieve attestation evidence. Work is needed on a YANG model for this protocol.
- Reference Measurements must be conveyed from the software authority (e.g., the manufacturer) to the system in which verification will take place. IETF CoSWID work forms the basis for this, but new work is needed to create an information model and YANG implementation.

Gaps still exist for implementation in Network Equipment (as of May 2019):

- Coordination of YANG model development with the IETF is still needed
- Specifications for management of signed Reference Integrity Measurements must still be completed

5. Appendix
5.1. Implementation Notes

Table 1 summarizes many of the actions needed to complete an Attestation system, with links to relevant documents. While documents are controlled by a number of standards organizations, the implied actions required for implementation are all the responsibility of the manufacturer of the device, unless otherwise noted.

<table>
<thead>
<tr>
<th>Component</th>
<th>Controlling Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make a Secure execution environment</td>
<td></td>
</tr>
<tr>
<td>o Attestation depends entirely on a secure</td>
<td></td>
</tr>
<tr>
<td>root of trust for measurement.</td>
<td>TCG RoT UEFI.org</td>
</tr>
<tr>
<td>o Refer to TCG Root of Trust for Measurement.</td>
<td></td>
</tr>
<tr>
<td>o NIST SP 800-193 also provides guidelines</td>
<td></td>
</tr>
<tr>
<td>on Roots of Trust</td>
<td></td>
</tr>
<tr>
<td>Get a TPM properly provisioned as described in</td>
<td></td>
</tr>
<tr>
<td>TCG documents.</td>
<td></td>
</tr>
<tr>
<td>Put a DevID or Platform Cert in the TPM</td>
<td></td>
</tr>
<tr>
<td>o Install an Initial Attestiation Key at the</td>
<td></td>
</tr>
<tr>
<td>same time so that Attestation can work out</td>
<td></td>
</tr>
<tr>
<td>of the box</td>
<td>TCG TPM DevID TCG Platform</td>
</tr>
<tr>
<td>o Equipment suppliers and owners may want to</td>
<td>Certificate</td>
</tr>
<tr>
<td>implement Local Device ID as well as</td>
<td></td>
</tr>
<tr>
<td>Initial Device ID</td>
<td>IEEE 802.1AR</td>
</tr>
<tr>
<td>Connect the TPM to the TLS stack</td>
<td></td>
</tr>
<tr>
<td>o Use the DevID in the TPM to authenticate</td>
<td>Vendor TLS</td>
</tr>
<tr>
<td>TAP connections, identifying the device</td>
<td>stack (This action is</td>
</tr>
<tr>
<td>Make CoSWID tags for BIOS/LoaderLKernel objects</td>
<td></td>
</tr>
<tr>
<td>o Add reference measurements into SWID tags</td>
<td></td>
</tr>
<tr>
<td>o Manufacturer should sign the SWID tags</td>
<td>IETF CoSWID ISO/IEC 19770-2</td>
</tr>
<tr>
<td>o This should be covered in a new TCG</td>
<td></td>
</tr>
<tr>
<td>Reference Integrity Manifest document</td>
<td></td>
</tr>
<tr>
<td>– IWG should define the literal SWID</td>
<td>NIST IR 8060 TagVault SWID</td>
</tr>
<tr>
<td>format</td>
<td>Tag Signing Guidance</td>
</tr>
</tbody>
</table>
- IWG should evaluate whether IETF SUIT is a suitable manifest when multiple SWID tags are involved
- There could be a proof-of-concept project to actually make sample SWID tags (a gap might appear in the process)

Package the SWID tags with a vendor software release
  - A tag-generator plugin could help (i.e., a plugin for common development environments. NIST has something that plugs into Maven Build Environment)

BIOS SWIDs might be hard to manage on an OS disk-- maybe keep them in the BIOS flash?
  - Maybe a UEFI Var? Would its name have to be specified by UEFI.org?
  - How big is a BIOS SWID tag? Do we need to use a tag ID instead of an actual tag?
  - Note that the presence of Option ROMs turns the BIOS reference measurements into a multi-vendor interoperability problem

Use PC Client measurement definitions as a starting point to define the use of PCRs (although Windows OS is rare on Networking Equipment)

Use TAP to retrieve measurements
  - Map TAP to SNMP
  - Map to YANG
  - Complete Canonical Log Format
Posture Collection Server (as described in IETF SACMs ECP) would have to request the attestation and analyze the result. The Management application might be broken down to several more components:

- A Posture Manager Server, which collects reports and stores them in a database.
- One or more Analyzers that can look at the results and figure out what it means.

---

**Figure 6: Component Status**

5.2. Comparison with TCG PTS / IETF NEA

Some components of an Attestation system have been implemented for end-user machines such as PCs and laptops. Figure 7 shows the corresponding protocol stacks.
6. IANA Considerations

This memo includes no request to IANA.
7. Security Considerations

TBD

8. Informative References

[Canonical-Event-Log]

[EFI]

[Firmware-Profile]

[I-D.birkholz-rats-architecture]

[I-D.birkholz-rats-reference-interaction-model]

[I-D.birkholz-suit-coswid-manifest]

[I-D.birkholz-yang-basic-remote-attestation]
Birkholz, H., "Software Inventory YANG module based on Software Identifiers", draft-birkholz-yang-swid-02 (work in progress), October 2018.


[PC-Client-BIOS-TPM-2.0]

[Platform-Certificates]

[Platform-DevID-TPM-2.0]

[Platform-ID-TPM-1.2]


[SNMP-Attestation-MIB] Trusted Computing Group, "DRAFT: SNMP MIB for TPM-Based Attestation, Specification Version 0.8, Revision 0.02", May 2018.


[TAP] Trusted Computing Group, "DRAFT: TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0, Version 1.0, Revision 0.29", October 2018.

Authors’ Addresses

Guy Fedorkow (editor)
Juniper Networks, Inc.
US

Email: gfedorkow@juniper.net

Jessica Fitzgerald-McKay
National Security Agency
US

Email: jmfitz2@nsa.gov
The Entity Attestation Token (EAT)  

draft-ietf-rats-eat-01

Abstract

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

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

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 January 5, 2020.
Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
   1.1. CDDL, CWT and JWT ....................................... 4
   1.2. Entity Overview ........................................... 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
2. Terminology .................................................... 7
3. The Claims Information Model ................................. 8
   3.1. Nonce Claim (cti and jti) ............................... 8
   3.2. Timestamp claim (iat) ................................... 8
   3.3. Universal Entity ID Claim (ueid) ....................... 8
   3.4. Origination Claim (origination) ....................... 11
       3.4.1. CDDL ................................................. 12
   3.5. OEM identification by IEEE OUI (oemid) ................ 12
       3.5.1. CDDL ................................................. 12
   3.6. The Security Level Claim (security_level) ........... 12
       3.6.1. CDDL ................................................. 13
   3.7. Secure Boot and Debug Enable State Claims (boot_state) . 13
       3.7.1. Secure Boot Enabled ................................ 13
       3.7.2. Debug Disabled ..................................... 13
       3.7.3. Debug Disabled Since Boot ......................... 14
       3.7.4. Debug Permanent Disable ......................... 14
       3.7.5. Debug Full Permanent Disable .................... 14
       3.7.6. CDDL ................................................. 14
   3.8. The Location Claim (location) ......................... 14
       3.8.1. CDDL ................................................. 15
   3.9. The Age Claim (age) .................................... 15
   3.10. The Uptime Claim (uptime) ............................. 15
       3.10.1. CDDL ............................................... 15
   3.11. Nested EATs, the EAT Claim (nested_eat) ............. 15
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 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)
- 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

1.1. CDDL, CWT and JWT

An EAT token is either a CWT as defined in [RFC8392] or a JWT as defined in [RFC7519]. This specification defines additional claims for entity attestation.

This specification uses CDDL, [RFC8610], as the primary formalism to define each claim. The implementor then interprets the CDDL to come to either the CBOR [RFC7049] or JSON [ECMAScript] representation. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 4.3.2 of this document where Appendix E is insufficient. (Note that this is not to define a general means to translate between CBOR and JSON, but only to define enough such that the claims defined in this document can be rendered unambiguously in JSON).

1.2. 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.

1.3. 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).

All these operating models are supported and there is no preference of 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

The following is not standardized for EAT, just the same they are not standardized for CWT or JWT.

1.4.1. Transmission Protocol

EATs may be transmitted by any protocol the same as CWTs and JWTs. 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 and JOSE 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.

Over time to facilitate interoperability, some signing schemes may be
defined in EAT profiles or other documents either in the IETF or
outside.

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].

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

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

Claim Value. The CBOR map or JSON object value representing the
value of the claim.

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

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 Information Model

This section describes new claims defined for attestation. It also mentions several claims defined by CWT and JWT are particularly important for EAT.

Note also: * Any claim defined for CWT or JWT may be used in an EAT including those in the CWT [IANA.CWT.Claims] and JWT IANA [IANA.JWT.Claims] claims registries. * All claims are optional * No claims are mandatory * All claims that are not understood by implementations MUST be ignored

CDDL along with text descriptions is used to define the information model. Each claim is defined as a CDDL group (the group is a general aggregation and type definition feature of CDDL). In the data model, described in the Section 4, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

3.1. Nonce Claim (cti and jti)

All EATs should have a nonce to prevent replay attacks. The nonce is generated by the relying party, sent to the entity by any protocol, and included in the token. Note that intrinsically by the nature of a nonce no security is needed for its transport.

CWT defines the "cti" claim. JWT defines the "jti" claim. These carry the nonce in an EAT.

TODO: what about the JWT claim "nonce"?

3.2. Timestamp claim (iat)

The "iat" claim defined in CWT and JWT is used to indicate the date-of-creation of the token.

3.3. Universal Entity ID Claim (ueid)

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

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

There are privacy considerations for UEID’s. See Section 6.1.

The UEID should be permanent. It should never change for a given
device / entity. In addition, it should not be reprogrammable.
UEID’s are variable length. The recommended maximum is 33 bytes (1
type byte and 256 bits). The recommended minimum is 17 bytes (1 type
and 128 bits) because fewer bytes endanger the universal uniqueness.

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

Creation of new types requires a Standards Action [RFC8126].
<table>
<thead>
<tr>
<th>Type Byte</th>
<th>Type Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>RAND</td>
<td>This is a 128- to 256-bit random number generated once and stored in the device. This may be constructed by concatenating enough identifiers to be universally unique and then feeding the concatenation through a cryptographic hash function. It may also be a cryptographic quality random number generated once at the beginning of the life of the device and stored.</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>This is a 14-digit identifier consisting of an 8-digit Type Allocation Code and a 6-digit serial number allocated by the manufacturer, which SHALL be encoded as a binary integer over 48 bits. The IMEI value encoded SHALL NOT include Luhn checksum or SVN information.</td>
</tr>
<tr>
<td>0x04</td>
<td>EUI-48</td>
<td>This is a 48-bit identifier formed by concatenating the 24-bit OUI with a 24-bit identifier assigned by the organisation that purchased the OUI.</td>
</tr>
<tr>
<td>0x05</td>
<td>EUI-60</td>
<td>This is a 60-bit identifier formed by concatenating the 24-bit OUI with a 36-bit identifier assigned by the organisation that purchased the OUI.</td>
</tr>
<tr>
<td>0x06</td>
<td>EUI-64</td>
<td>This is a 64-bit identifier formed by concatenating the 24-bit OUI with a 40-bit identifier assigned by the organisation that purchased the OUI.</td>
</tr>
</tbody>
</table>

Table 1: UEID Composition Types

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

The consumer (the relying party) of a UEID MUST treat a UEID as a completely opaque string of bytes and not make any use of its
internal structure. For example, they should not use the OUI part of a type 0x02 UEID to identify the manufacturer of the device. Instead they should use the 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 0x07 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.

### CDDL

ueid_claim = {
  ueid: bstr
}

3.4. Origination Claim (origination)

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>

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.
3.4.1. CDDL

    origination_claim = (origination: string_or_uri)

3.5. OEM identification by IEEE OUI (oemid)

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.

3.5.1. CDDL

    oemid_claim = (oemid: bstr)

3.6. The Security Level Claim (security_level)

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.

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 Wi-Fi 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.

3.6.1. CDDL

```cddl
security_level_type = {
  unrestricted: 1,
  restricted: 2,
  secure_restricted: 3,
  hardware: 4
}

security_level_claim = {
  security_level: security_level_type
}
```

3.7. Secure Boot and Debug Enable State Claims (boot_state)

This claim is an array of five Boolean values indicating the boot and debug state of the entity.

3.7.1. Secure Boot Enabled

This 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.

3.7.2. Debug Disabled

This 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.
3.7.3. Debug Disabled Since Boot

This claim indicates whether debug capabilities for the entity were not disabled in any way since boot (i.e. value of ‘true’).

3.7.4. Debug Permanent Disable

This claim 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 enable debug, but the end user is not.

3.7.5. Debug Full Permanent Disable

This claim 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.

3.7.6. CDDL

```cddl
boot_state_type = [
  secure_boot_enabled=> bool,
  debug_disabled=> bool,
  debug_disabled_since_boot=> bool,
  debug_permanent_disable=> bool,
  debug_full_permanent_disable=> bool
]

boot_state_claim = (boot_state: boot_state_type)
```

3.8. The Location Claim (location)

The location claim is a CBOR-formatted object that describes the location of the device entity from which the attestation originates. It is comprised of a map of additional sub claims 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 sub claim providing the estimated accuracy of the location measurement is defined.
3.8.1. CDDL

```plaintext
location_type = {
  latitude => number,
  longitude => number,
  altitude => number,
  accuracy => number,
  altitude_accuracy => number,
  heading_claim => number,
  speed_claim => number
}

location_claim = {
  location: location_type
}
```

3.9. The Age Claim (age)

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.

```plaintext
age_claim = {
  age: uint
}
```

3.10. The Uptime Claim (uptime)

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

3.10.1. CDDL

```plaintext
uptime_claim = {
  uptime: uint
}
```

3.11. Nested EATs, the EAT Claim (nested_eat)

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.
3.11.1. CDDL

```cddl
nested_eat_claim = (  
nested_eat: nested_eat_type)
```

A `nested_eat_type` is defined in words rather than CDDL. It is either a full CWT or JWT including the COSE or JOSE signing.

3.12. The Submods Claim (submods)

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., Wi-Fi and cellular). It may have subsystems for low-power audio and video playback. It may have one or more security-oriented subsystems like a TEE or a Secure Element.

The claims for 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.

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.

3.12.1. The submod_name Claim

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

3.12.2. CDDL

In the following a `generic_claim_type` is any CBOR map entry or JSON name/value pair.
submod_name_type = ( 
    submod_name: tstr )

submods_type = [ * submod_claims ]

submod_claims = { 
    submod_name_type,
    * generic_claim_type
}

submods_claim = ( 
    submods: submod_type )

4. Data Model

This makes use of the types defined in CDDL Appendix D, Standard Prelude.

4.1. Common CDDL Types

string_or_uri = #6.32(tstr) / tstr; See JSON section below for JSON encoding of string_or_uri

4.2. CDDL for CWT-defined Claims

This section provides CDDL for the claims defined in CWT. It is non-normative as [RFC8392] is the authoritative definition of these claims.
cwt_claim = {
  issuer_claim //
  subject_claim //
  audience_claim //
  expiration_claim //
  not_before_claim //
  issued_at_claim //
  cwt_id_claim
}

issuer_claim = {
  issuer: string_or_uri 
}

subject_claim = {
  subject: string_or_uri 
}

audience_claim = {
  audience: string_or_uri 
}

expiration_claim = {
  expiration: time 
}

not_before_claim = {
  not_before: time 
}

issued_at_claim = {
  issued_at: time 
}

cwt_id_claim = {
  cwt_id: bstr 
}

issuer = 1
subject = 2
audience = 3
expiration = 4
not_before = 5
issued_at = 6
cwt_id = 7

4.3. JSON

4.3.1. JSON Labels
4.3.2. JSON Interoperability

JSON should be encoded per RFC 8610 Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

- bstr - must be base64url encoded
- time - must be encoded as NumericDate as described in section 2 of [RFC7519].
- string_or_uri - must be encoded as StringOrURI as described in section 2 of [RFC7519].

4.4. CBOR

4.4.1. Labels

uelid = 8
origination = 9
oemid = 10
security_level = 11
boot_state = 12
location = 13
age = 14
uptime = 15
nested_eat = 16
submods = 17
submod_name = 18
latitude 1
longitude 2
altitude 3
accuracy 4
altitude_accuracy 5
heading_claim 6
speed_claim 7
4.4.2. CBOR Interoperability

Variations in the CBOR serializations supported in CBOR encoding and decoding are allowed and suggests that CBOR-based protocols specify how this variation is handled. This section specifies what formats MUST be supported in order to achieve interoperability.

The assumption is that the entity is likely to be a constrained device and relying party is likely to be a very capable server. The approach taken is that the entity generating the token can use whatever encoding it wants, specifically encodings that are easier to implement such as indefinite lengths. The relying party receiving the token must support decoding all encodings.

These rules cover all types used in the claims in this document. They also are recommendations for additional claims.

Canonical CBOR encoding, Preferred Serialization and Deterministically Encoded CBOR are explicitly NOT required as they would place an unnecessary burden on the entity implementation, particularly if the entity implementation is implemented in hardware.

- Integer Encoding (major type 0, 1) - The entity may use any integer encoding allowed by CBOR. The server MUST accept all integer encodings allowed by CBOR.

- 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 to indicate conversion to base64url in case that conversion is performed.

- 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.

- 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. Collected CDDL

A generic_claim is any CBOR map entry or JSON name/value pair.

eat_claims = { ; the top-level payload that is signed using COSE or JOSE
* claim
}

claim = {
  ueid_claim //
  origination_claim //
  oemid_claim //
  security_level_claim //
  boot_state_claim //
  location_claim //
  age_claim //
  uptime_claim //
  nested_eat_claim //
  cwt_claim //
  generic_claim_type //
}

TODO: copy the rest of the CDDL here (wait until the CDDL is more
settled so as to avoid copying multiple times)

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. No new IANA registry is created. All
EAT claims should be registered in the CWT and JWT Claims Registries.
5.1.1. Claims Registered by This Document

- Claim Name: UEID
- Claim Description: The Universal Entity ID
- JWT Claim Name: N/A
- Claim Key: 8
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): *this document*

TODO: add the rest of the claims in here

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 (cti) /         7:h’948f8860d13a463e8e’,
  / UEID /               8:h’0198f50a4ff6c05861c8860d13a638ea4fe2f’,
  / boot_state /         12:{true, true, true, true, false}
  / time stamp (iat) /   6:1526542894
}
```

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

```
{
  / nonce /              7:h’948f8860d13a463e8e’,
  / UEID /               8:h’0198f50a4ff6c05861c8860d13a638ea4fe2f’,
  / boot_state /         12:{true, true, true, true, false}
  / time stamp (iat) /   6:1526542894,
  / seclevel /           11:3, / secure restricted OS /
  / submods / 17:
    [                  
      / 1st submod, an Android Application / {  
        / submod_name /   18:'Android App "Foo"',
        / seclevel /      11:1, / unrestricted /
        / app data / -70000:'text string'
      },
      / 2nd submod, A nested EAT from a secure element / {  
        / submod_name / 18:'Secure Element EAT',
        / eat /         16:61( 18(  
          / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]  
        )  
      })
    ]
  / 3rd submod, information about Linux Android / {  
    / submod_name/ 18:'Linux Android',
    / seclevel / 11:1, / unrestricted /
    / custom - release / -80000:'8.0.0',
    / custom - version / -80001:'4.9.51+'
  }
}
```
Appendix B. Changes from Previous Drafts

The following is a list of known changes from the previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences.

B.1. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but not new claims have been added.

- Separate information and data model using CDDL.
- Say an EAT is a CWT or JWT
- Use a map to structure the boot_state and location claims

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
Use cases for Remote Attestation common encodings
draft-richardson-rats-usecases-04

Abstract

This document details mechanisms created for performing Remote
Attestation that have been used in a number of industries. The
document initially focuses on existing industry verticals, mapping
terminology used in those specifications to the more abstract
terminology used by the IETF RATS Working Group.

The document aspires to describe possible future use cases that would
be enabled by common formats.

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Richardson, et al. Expires January 9, 2020
Table of Contents

1.  Introduction .................................................. 3
2.  Terminology .................................................. 3
   2.1.  Static attestations ....................................... 4
   2.2.  Session attestations ..................................... 4
   2.3.  Statements ............................................... 4
   2.4.  Hardware Root Of Trust .................................. 4
3.  Requirements Language .......................................... 4
4.  Overview of Sources of Use Cases ............................. 4
5.  Use case summaries ............................................. 5
   5.1.  Device Capabilities/Firmware Attestation ................ 5
      5.1.1.  Relying on an Attestation Server .................. 5
      5.1.2.  Autonomous Relying Party .......................... 5
      5.1.3.  Proxy Root of Trust ............................... 5
      5.1.4.  network scaling - small ............................ 6
      5.1.5.  network scaling - medium .......................... 6
      5.1.6.  network scaling - large ........................... 6
   5.2.  Hardware resiliency / watchdogs .......................... 6
   5.3.  IETF TEEP WG use case ................................... 7
   5.4.  Confidential Machine Learning (ML) model ............... 7
   5.5.  Critical infrastructure .................................. 7
      5.5.1.  Computation characteristics ........................ 7
   5.6.  Cryptographic Key Attestation ............................ 8
      5.6.1.  Device Type Attestation ............................. 8
      5.6.2.  Key storage attestation ............................. 8
      5.6.3.  End user authorization .............................. 9
   5.7.  Geographic attestation .................................... 9
      5.7.1.  I am here .......................................... 9
      5.7.2.  I am near .......................................... 9
      5.7.3.  You are here ...................................... 10
   5.8.  Connectivity attestation ................................ 10
6.  Technology users for RATS ...................................... 10
   6.1.  Trusted Computing Group (TCG) .......................... 10
   6.2.  Android Keystore system ................................. 12
   6.3.  Fast IDentity Online (FIDO) Alliance ................... 12
7.  Examples of Existing Attestation Formats .................... 14
   7.1.  Android Keystore ....................................... 14
      7.1.1.  TEE .............................................. 15
      7.1.2.  Secure Element .................................... 19
   7.2.  Windows 10 TPM .......................................... 24
1. Introduction

The recently chartered IETF RATS WG intends to create a system of attestations that can be shared across a multitude of different users.

This document exists as place to collect use cases for the common RATS technologies in support of the IETF RATS charter point 1. This document is not expected to be published as an RFC, but remain open as a working document. It could become an appendix to provide motivation for a protocol standards document.

End-user use cases that would either directly leverage RATS technology, or would serve to inform technology choices are welcome, however.

2. Terminology

Critical to dealing with and contrasting different technologies is to collect terms which are compatible, to distinguish those terms which are similar but used in different ways.

This section will grow to include forward and external references to terms which have been seen. When terms need to be disambiguated they will be prefixed with their source, such as "TCG(claim)" or "FIDO(relying party)"

Platform attestations generally come in two categories. This document will attempt to indicate for a particular attestation technology falls into this.
2.1. Static attestations

A static attestation says something about the platform on which the code is running.

2.2. Session attestations

A session attestation says something about how a session key used in a connection such as TLS connection was created. It is usually the result of evaluating attestations that are attached to the certificates used to create such a session.

2.3. Statements

The term "statement" is used as the generic term for the semantic content which is being attested to.

2.4. Hardware Root Of Trust

(TBD: Seeking something useful here.)

3. Requirements Language

This document is not a standards track document and does not make any normative protocol requirements using terminology described in [RFC2119].

4. Overview of Sources of Use Cases

The following specifications have been converged in this document:

- The Trusted Computing Group "Network Attestation System" (private document)
- Android Keystore
- Fast Identity Online (FIDO) Alliance attestation,

This document will be expanded to include summaries from:

- Trusted Computing Group (TCG) Trusted Platform Module (TPM)/Trusted Software Stack (TSS)
- ARM "Platform Security Architecture" [I-D.tschofenig-rats-psa-token]
- Intel SGX attestation [intelsgx]
5. Use case summaries

This section lists a series of cases where an attestation is done.

5.1. Device Capabilities/Firmware Attestation

A network operator wants to know the qualities of the hardware and software on the machines attached to their network. The process starts with some kind of Root of Trust, performs a series of measurements, and expresses this with an attestation as to the hardware and firmware/software which is running. This is a general description for which there are many specific use cases.

5.1.1. Relying on an Attestation Server

The measurements from a heterogenous network of devices are provided to device-specific attestation servers. The attestation servers know what the "golden" measurements are, and perform the appropriate evaluations, resulting in attestations that the relying parties can depend upon.

5.1.2. Autonomous Relying Party

The signed measurements are sent to a relying party which must validate them directly. (It may do so with the help of a signed list of golden values, or some other process). The relying party needs to validate the signed statements directly.

This may occur because the network is not connected, or even because it cannot be connected until the equipment is validated.

5.1.3. Proxy Root of Trust

A variety of devices provide measurements via their Root of Trust. A server collects these measurements, and (having applied a local
policy) then creates a device agnostic attestation. The relying party can validate the claims in a standard format.

5.1.4. network scaling — small

An entire network of systems needs to be validated (such as all the desktops in an enterprise’s building). The infrastructure is in control of a single operator and is already trusted. The network can be partitioned so that machines that do not pass attestation can be quarantined. A 1:1 relationship between the device and the relying party can be used to maintain freshness of the attestation.

5.1.5. network scaling — medium

An entire network of systems needs to be validated: such as all the desktops in an enterprise’s building, or all the routers at an ISP. The infrastructure is not necessarily trusted: it could be subverted, and it must also attest. The devices may be under a variety of operators, and may be mutually suspicious: each device may therefore need to process attestations from every other device. An NxM mesh of attestations may be untenable, but a system of N:1:M relationships can be setup via proxy attestations.

5.1.6. network scaling — large

An entire network of systems need to be continuously attested. This could be all of the smartphones on an LTE network, or every desktop system in a worldwide enterprise. The network operator wishes to do this in order maintain identities of connected devices more than to validate correct firmware, but both situations are reasonable.

5.2. Hardware resiliency / watchdogs

One significant problem is malware that holds a device hostage and does not allow it to reboot to prevent updates to be applied. This is a significant problem, because it allows a fleet of devices to be held hostage for ransom. Within CyRes the TCG is defining hardware Attention Triggers that force a periodical reboot in hardware.

This can be implemented by forcing a reboot unless attestation to an Attestation Server succeeds within the period interval, and having a reboot do remediation by bringing a device into compliance, including installation of patches as needed.

This is unlike the previous section on Device Attestation in that the attestation comes from a network operator, as to the device’s need to continue operating, and is evaluated by trusted firmware (the relying party), which resets a watchdog timer.
5.3. IETF TEEP WG use case

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

5.4. Confidential Machine Learning (ML) model

Microsoft talked about this category of use cases at the recent Microsoft //build conference.

An example use case is where a device manufacturer wants to protect its intellectual property in terms of the ML model it developed and that runs in the devices that its customers purchased, and it wants to prevent attackers, potentially including the customer themselves, from seeing the details of the model. This works by having some protected environment (e.g., a hardware TEE) in the device attest to some manufacturer’s service, which if attestation succeeds, then the manufacturer service releases the model, or a key to decrypt the model, to the requester. If a hardware TEE is involved, then this use case overlaps with the TEEP use case.

5.5. Critical infrastructure

When a protocol operation can affect some critical system, the device attached to the critical equipment wants some assurance that the requester has not been compromised. As such, attestation can be used to only accept commands from requesters that are within policy. Hardware attestation in particular, especially in conjunction with a TEE on the requester side, can provide protection against many types of malware.

5.5.1. Computation characteristics

A group of enterprises organized as a consortium seeks to deploy computing nodes as the basis of their shared blockchain system. Each member of the consortium must forward an equal number of computing nodes to participate in the P2P network of nodes that form the basis of the blockchain system. In order to prevent the various issues (e.g., concentration of hash power, anonymous mining nodes) found in other blockchain systems, each computing node must comply to a predefined allowable manifest of system hardware, software and firmware, as agreed to by the membership of the consortium. Thus, a given computing node must be able to report the (pre-boot)
configuration of its system and be able to report at any time the operational status of the various components that make-up its system.

The consortium seeks to have the following things attested: system configuration, group membership, and virtualization status.

This is a peer-to-peer protocol so each device in the consortium is a relying party. The attestation may be requested online by another entity within the consortium, but not by other parties. The attestation needs to be compact and interoperable and may be included in the blockchain itself at the completion of the consensus algorithm.

The attestation will need to start in a hardware RoT in order to validate if the system is running real hardware rather than running a virtual machine.

5.6. Cryptographic Key Attestation

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

5.6.1. Device Type Attestation

This use case convinces the relying party of the characteristics of a device. For privacy reasons, it might not identify the actual device itself, but rather the class of device. The relying party can understand from either in-band (claims) or out-of-band (model numbers, which may be expressed as a claim) whether the device has features such as a hardware TPM, software TPM via TEE, or software TPM without TEE. Other details such as the availability of fingerprint readers or HDMI outputs may also be inferred.

5.6.2. Key storage attestation

This use case convinces the relying party only about the provenance of a private key by providing claims of the storage security of the private key. This can be conceived as a subset of the previous case, but may be apply very specifically to just a keystore. Additional details associated with the private key may be provided as well, including limitations on usage of the key.

Key storage attestations may be consumed by systems provisioning public key certificates for devices or human users. In these cases, attestations may be incorporated into certificate request protocols
5.6.3. End user authorization

This use case convinces the relying party that the digital signatures made by the indicated key pair were done with the approval of the end-user operator. This may also be considered possible subset of the device attestation above, but the attestation may be on a case-by-case basis. The nature of the approval by the end-user would be indicated. Examples include: the user unlocked the device, the user viewed some message and acknowledge it inside an app, the message was displayed to the user via out-of-app control mechanism. The acknowledgements could include selecting options on the screen, pushing physical buttons, scanning fingerprints, proximity to other devices (via bluetooth beacons, chargers, etc).

5.7. Geographic attestation

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

5.7.1. I am here

The simplest use case is the claim of some specific coordinates.

5.7.2. I am near

The second use case is the claim that some other devices are nearby. This may be absolute ("I am near device X, which claims to be at location A"), or just relative, ("I am near device X"). This use could use "I am here" or "I am near" claims from a 1:1 basis with device X, or use some other protocol. The nature of how the proximity was established would be part of this claim. In order to defeat a variety of mechanisms that might attempt to proxy ("wormhole") radio communications, highly precise clocks may be required, and there may also have to be attestations as to the precision of those clocks.

An additional example of being near would be for the case where two smartphones can establish that they are together by recording a common random movement, such as both devices being shaken together. Each device may validate the claim from the other (in a disconnected
fashion), or a third party may validate the claim as the relying party.

This could be used to establish that a medical professional was in proximity of a patient with implanted devices who needs help.

5.7.3. You are here

A third way to establish location is for a third party to communicate directly with the relying party. The nature of how this trust is established (and whether it is done recursively) is outside of the scope here. What is critical is that the identity of "You" can be communicated through the third party in a way that the relying party can use, but other intermediaries can not view.

5.8. Connectivity attestation

The relying party wants to know what devices are connected. A typical situation would be a media owner needing to know what TV device is connected via HDMI and if High-bandwidth Digital Content Protection (HDCP) is intact.

6. Technology users for RATS

6.1. Trusted Computing Group (TCG)

The TCG is trying to solve the problem of knowing if a networking device should be part of a network, if it belongs to the operator, and if it is running appropriate software. The work covers most of the use cases in Section 5.1.

This proposal is a work-in-progress, and is available to TCG members only. The goal is to be multi-vendor, scalable and extensible. The proposal intentionally limits itself to:

- "non-privacy-preserving applications (i.e., networking, Industrial IoT )",

- the firmware is provided by the device manufacturer

- there is a manufacturer installed hardware root of trust (such as a TPM and boot room)

Service providers and enterprises deploy hundreds of routers, many of them in remote locations where they’re difficult to access or secure. The point of remote attestation is to:

- identify a remote box in a way that’s hard to spoof
The use case described is to be able to monitor the authenticity of software versions and configurations running on each device. This allows owners and auditors to detect deviation from approved software and firmware versions and configurations, potentially identifying infected devices. [RFC5209]

Attestation may be performed by network management systems. Networking Equipment is often highly interconnected, so it’s also possible that attestation could be performed by neighboring devices.

Specifically listed to be out of scope for the first generation includes: Linux processes, assemblies of hardware/software created by end-customers, and equipment that is sleepy. There is an intention to cover some of these topics in future versions of the documents.

The TCG Attestation leverages the TPM to make a series of measurements during the boot process, and to have the TPM sign those measurements. The resulting "PCG" hashes are then available to an external verifier.

The TCG uses the following terminology:

- Device Manufacturer
- Attester ("device under attestation")
- Verifier (Network Management Station)
- "Explicit Attestation" is the TCG term for a static (platform) attestation
- "Implicit Attestation" is the TCG term for a session attestation
- Reference Integrity Measurements (RIM), which are signed by device manufacturer and integrated into firmware.
- Quotes: measured values (having been signed), and RIMs
- Reference Integrity Values (RIV)
- devices have a Initial Attestation Key (IAK), which is provisioned at the same time as the IDevID [ieee802-1AR]
- PCR - Platform Configuration Registry (deals with hash chains)
The TCG document builds upon a number of IETF technologies: SNMP (Attestation MIB), YANG, XML, JSON, CBOR, NETCONF, RESTCONF, CoAP, TLS and SSH. The TCG document leverages the 802.1AR IDevID and LDevID processes.

6.2. Android Keystore system

[keystore] describes a system used in smart phones that run the Android operation system. The system is primarily a software container to contain and control access to cryptographic keys, and therefore provides many of the same functions that a hardware Trusted Platform Module might provide.

The uses described in section Section 5.6 are the primary focus.

On hardware which is supported, the Android Keystore will make use of whatever trusted hardware is available, including use of a Trusted Execution Environment (TEE) or Secure Element (SE). The Keystore therefore abstracts the hardware, and guarantees to applications that the same APIs can be used on both more and less capable devices.

A great deal of focus from the Android Keystore seems to be on providing fine-grained authorization of what keys can be used by which applications.

XXX - clearly there must be additional (intended?) use cases that provide some kind of attestation.

Android 9 on Pixel 2 and 3 can provided protected confirmation messages. This uses hardware access from the TPM/TEE to display a message directly to the user, and receives confirmation directly from the user. A hash of the contents of the message can provided in an attestation that the device provides.

In addition, the Android Keystore provides attestation information about itself for use by FIDO.

QUOTE: Finally, the Verified Boot state is included in key attestation certificates (provided by Keymaster/Strongbox) in the deviceLocked and verifiedBootState fields, which can be verified by apps as well as passed onto backend services to remotely verify boot integrity

6.3. Fast IDentity Online (FIDO) Alliance

The FIDO Alliance [fido] has a number of specifications aimed primarily at eliminating the need for passwords for authentication to online services. The goal is to leverage asymmetric cryptographic
operations in common browser and smart-phone platforms so that users can easily authentication.

The use cases of Section 5.6 are primary.

FIDO specifications extend to various hardware second factor authentication devices.

Terminology includes:

- "relying party" validates a claim
- "relying party application" makes FIDO Authn calls
- "browser" provides the Web Authentication JS API
- "platform" is the base system
- "internal authenticator" is some credential built-in to the device
- "external authenticator" may be connected by USB, bluetooth, wifi, and may be an stand-alone device, USB connected key, phone or watch.

FIDO2 had a Key Attestation Format [fidoattestation], and a Signature Format [fidosignature], but these have been combined into the W3C document [fido_w3c] specification.

A FIDO use case involves the relying party receiving a device attestation about the biometric system that performs the identification of the human. It is the state of the biometric system that is being attested to, not the identity of the human!

FIDO does provides a transport in the form of the WebAuthn and FIDO CTAP protocols.

According to [fidotechnote] FIDO uses attestation to make claims about the kind of device which is be used to enroll. Keypairs are generated on a per-device _model_ basis, with a certificate having a trust chain that leads back to a well-known root certificate. It is expected that as many as 100,000 devices in a production run would have the same public and private key pair. One assumes that this is stored in a tamper-proof TPM so it is relatively difficult to get this key out. The use of this key attests to the the device type, and the kind of protections for keys that the relying party may assume, not to the identity of the end user.
7. Examples of Existing Attestation Formats.

This section provides examples of some existing attestation formats.

7.1. Android Keystore

Android Keystore attestations take the form of X.509 certificates. The examples below package the attestation certificate along with intermediate CA certificates required to validate the attestation as a certificates-only SignedData message [RFC5652]. The trust anchor is available here: [keystore_attestation].

The attestations below were generated using the generateKeyPair method from the DevicePolicyManager class using code similar to the following.

```java
```
Annotations included below are delimited by ASN.1 comments, i.e., -. Annotations should be consistent with structures described here: [keystore_attestation].


20 3D 68 30 A5 86 26 14 77 : AD 7E 93 F5 1D 38 A9 DB 5B FE B2 B8 1A
7B CD 22 : 3B 17 98 FC 1F 4F 77 2D 92 E9 DE 5F 6B 02 09 4E : 99 86 53
98 1C SE 23 B6 A4 61 53 A5 FB D1 37 09 : DB C0 0A 40 E9 28 E6 BE E2
8E 57 94 A9 F2 13 3A : 11 40 D2 34 99 A6 B4 F3 99 F2 5D 4A 5E 6C
4B : [ Another 257 bytes skipped ] 547 3: INTEGER 65537 : ) ) ) )
552 221: {3} 555 218: SEQUENCE ( 558 29: SEQUENCE ( 560 3: OBJECT
IDENTIFIER subjectKeyIdentifier (2 5 29 14) 565 22: OCTET STRING,
encapsulates ( 567 20: OCTET STRING : 7B 7B F8 43 CA 1F 0F 96 27 0F
10 6F 7D OC 23 14 : 72 8F 1D 80 : ) ) ) ) ) 589 31: SEQUENCE ( 591 3:
OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29 35) 596 24: OCTET
STRING, encapsulates ( 598 22: SEQUENCE ( 600 20: [0] : 0E 55 6F 46
F5 3B 77 67 E1 B9 73 DC 55 E6 AE EA : B4 F2 7D DD : ) ) ) 622 12:
SEQUENCE ( 624 3: OBJECT IDENTIFIER basicConstraints (2 5 29 19) 629
1: BOOLEAN TRUE 632 1: OCTET STRING, encapsulates ( 634 0: SEQUENCE
() : : ) ) : 636 14: SEQUENCE ( 638 3: OBJECT IDENTIFIER keyUsage (2 5
29 15) 643 1: BOOLEAN TRUE 646 4: OCTET STRING, encapsulates ( 648 2:
BIT STRING 7 unused bits : '1'B (bit 0) : ) : 652 36: SEQUENCE ( 654 3:
OBJECT IDENTIFIER nameConstraints (2 5 29 30) 659 29: OCTET
STRING, encapsulates ( 661 27: SEQUENCE ( 663 25: [0] (665 23:
SEQUENCE ( 667 21: [2] 'invalid@email:invalid': : ) ) ) ) ) )
690 84: SEQUENCE ( 692 3: OBJECT IDENTIFIER cRLDistributionPoints (2
5 29 31) 697 77: OCTET STRING, encapsulates ( 699 75: SEQUENCE ( 701
73: SEQUENCE ( 703 71: [0] (705 69: [0] (707 67: [6]:
'https://android.googleapis.com/attestation/crl/1':
776 13: SEQUENCE ( 778 9: OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840
113549 1 1 11) 789 0: NULL : ) ) 791 513: BIT STRING : 69 13 A7 56 B3
9F E1 2B CE A2 09 89 E5 DC 03 B4 : B6 FF FG 1E 96 C7 62 C2 31 B3
D6 1A 9E 36 CF : C2 FC 0E 6F FA 0E CF B5 2D F8 19 D6 13 96 0B 56 : B0
EE 86 3B B1 B8 38 70 4E 57 EB D9 60 DC 58 74 : FE C8 EB A5 78 9F B7
19 5C F0 80 CF 29 16 6B 04 : 3A 5D 7C 2E 5F 11 12 36 BE 46 29 45 04
41 8F B5 : AB C6 31 5F 23 28 0C F2 7C 48 4A F6 43 AA 50 D0 : 53 96 1E
AD 7C A3 89 96 BB 8B BF 2D 9A OC 16 35 : [ Another 384 bytes skipped
] : ) 0 1393: SEQUENCE ( 4 857: SEQUENCE ( 8 3: [0] (10 1: INTEGER 2
: ) 13 10: INTEGER 03 88 26 67 60 65 89 96 85 74 25 13: SEQUENCE ( 27
9: OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1
11) 38 0: NULL : ) ) 40 27: SEQUENCE ( 42 25: SET ( 44 23: SEQUENCE ( 46
3: OBJECT IDENTIFIER serialNumber (2 5 4 5) 51 16: PrintableString
'f92009e853b6b045': : ) ) ) ) ) 69 30: SEQUENCE ( 71 13: UTCtime
26/05/2016 17:01:32 GMT 86 13: UTCtime 24/05/2026 17:01:32 GMT : )
101 27: SEQUENCE ( 103 25: SET ( 105 23: SEQUENCE ( 107 3: OBJECT
IDENTIFIER serialNumber (2 5 4 5) 112 16: PrintableString
'87f4514475ba0a2b': : ) ) ) ) 130 546: SEQUENCE ( 134 13: SEQUENCE ( 136
9: OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1) 147 0:
NULL : ) 149 527: BIT STRING, encapsulates ( 154 522: SEQUENCE ( 159
513: INTEGER : 00 D2 60 D6 45 85 E3 E2 23 79 5A 45 7D 6D : 5B
AF BD 9A 37 CB FA 97 C0 65 44 9D 3A C6 47 F6 : 0D 0B A2 74 12 CA F7
4B B9 5F FB B4 EC 5A 2B D0 : 16 01 DE BE E2 FE D2 76 0D 75 C4 B1 6A

CB 3A 67 : 07 21 E0 D5 19 68 C8 1B 01 A2 24 02 FE AD 40 D6 : A7 98 16
OF A2 98 2E A7 AD 75 34 84 6F F8 CF 8A : A1 0E 90 33 40 9E D0 86 26
57 71 CE FF CF 52 E1 : F0 F9 2B 7E 68 62 03 D8 FD FD 02 53 03 19 AC
IDENTIFIER subjectKeyIdentifier (2 5 29 14) 693 22: OCTET STRING,
encapsulates ( 695 20: OCTET STRING : 0E 55 6F 46 F5 3B 77 67 E1 B9
73 DC 55 E6 AE EA : B4 FD 27 DD : ) ) ) : ) 717 31: SEQUENCE ( 719 3:
OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29 35) 724 24: OCTET
STRING, encapsulates ( 726 22: SEQUENCE ( 728 20: [0] : 36 61 E1 00
7C 88 05 09 51 8B 44 6C 47 FF 1A 4C : C9 EA 4F 12 : ) ) : ) 750 15:
SEQUENCE ( 752 3: OBJECT IDENTIFIER basicConstraints (2 5 29 19) 757
1: BOOLEAN TRUE 760 5: OCTET STRING, encapsulates ( 762 3: SEQUENCE {
764 1: BOOLEAN TRUE : ) ) : ) 767 14: SEQUENCE ( 769 3: OBJECT
IDENTIFIER keyUsage (2 5 29 15) 774 1: BOOLEAN TRUE 777 4: OCTET
STRING, encapsulates ( 779 2: BIT STRING 1 unused bit : ‘1100001’B : )
: } ) : ) 783 80: SEQUENCE ( 785 3: OBJECT IDENTIFIER
cRLDistributionPoints (2 5 29 31) 790 73: OCTET STRING, encapsulates
( 792 71: SEQUENCE ( 794 69: SEQUENCE ( 796 67: [0] ) 798 65: [0] [800
865 13: SEQUENCE ( 867 9: OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840
113549 1 1 11) 878 0: NULL : ) 880 513: BIT STRING : 0E 0D 71 4A 88
0A 58 53 B6 31 14 7D DA 22 31 C6 : 06 D6 EF 3B 22 4D D7 A5 C0 3F BF
C6 B4 64 A3 FB : 92 C2 CC 67 F4 6C 24 25 49 6E F6 CB 08 D6 A8 0D : 94
06 7F 8C 8C 3C B1 77 CD C2 3F C7 5E A3 85 6D : F7 A5 94 13 CD 5A 5C
F3 9B 0A 0D E1 82 42 F4 C9 : 3F AD FC FB 7C AA 27 04 CC 1C 12 45 15
EB E6 70 : A0 6C DE 77 77 54 9B 1F 02 05 76 03 A4 FC 6C 07 : F4 CB BB
59 F5 CB ED 58 D8 30 9B 6E 3C F7 76 C1 : [ Another 384 bytes skipped ]
: ) ) 0 1376: SEQUENCE ( 4 840: SEQUENCE ( 8 3: [0] ) 10 1: INTEGER 2
: ) 13 9: INTEGER 00 E8 FA 19 63 14 D2 FA 18 24 13: SEQUENCE ( 26 9:
OBJECT IDENTIFIER : sha256WithRSAEncryption (1 2 840 113549 1 1 11)
37 0: NULL : ) 39 27: SEQUENCE ( 41 25: SET ( 43 23: SEQUENCE ( 45 3:
OBJECT IDENTIFIER serialNumber (2 5 4 5) 50 16: PrintableString
’f92009e853b6b045’ : ) ) : ) 68 30: SEQUENCE ( 70 13: UTCTime
26/05/2016 16:28:52 GMT 85 13: UTCTime 24/05/2026 16:28:52 GMT : )
100 27: SEQUENCE ( 102 25: SET ( 104 23: SEQUENCE ( 106 3: OBJECT
IDENTIFIER serialNumber (2 5 4 5) 111 16: PrintableString
’f92009e853b6b045’ : ) ) : ) 129 546: SEQUENCE ( 133 13: SEQUENCE {
135 9: OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1) 146 0:
NULL : ) 148 527: BIT STRING, encapsulates ( 153 522: SEQUENCE ( 157
513: INTEGER : 00 AF B6 C7 B2 2B B1 A7 01 EC 2B B4 2E BB CC 54 : 16
63 AB EF 98 2F 32 C7 7F 75 31 03 0C 97 52 4B : 1B 5F E8 09 FB C7 2A
A9 45 1F 74 3C BD 9A 6F 13 : 35 74 4A A5 5E 77 F6 B6 AC 35 35 EE 17
C2 5E 63 : 95 17 DD 9C 92 E6 37 4A 53 CB FE 25 8F 8F FB B6 : FD 12 93
7B A2 2A 4C A9 9C 45 2D 47 A5 9F 32 01 : F4 41 97 CA 1C CD 7E 76 2F
B2 F5 31 51 B6 FE B2 : FF FD 2B 6F E4 FE 5B C6 BD 9E C3 4B FE 08 23
9D : [ Another 385 bytes skipped ] 674 3: INTEGER 65537 : ) : ) : )

7.1.2. Secure Element

The structures below are not annotated except where the difference is specific to the difference between the TEE structure shown above and artifacts emitted by StrongBox.

```"```

Richardson, et al. Expires January 9, 2020
7F 21 6C EE 82 : C1 3F BD C1 41 4B DD 1A 6F 6C AF 4A 52 FC 19 19 : 17
AC 29 0C 5E D7 57 90 D5 B1 2B 36 2D 1F 45 33 : [ Another 384 bytes
skipped ] : ) 2348 1376: SEQUENCE { 2352 840: SEQUENCE { 2356 3: [0]
( 2358 1: INTEGER 2 : ) 2361 9: INTEGER 00 E8 FA 19 63 14 D2 FA 18
2372 13: SEQUENCE { 2374 9: OBJECT IDENTIFIER :
sha256WithRSAEncryption (1 2 840 113549 1 1 11) 2385 0: NULL : } 2387
27: SEQUENCE { 2389 25: SET { 2391 23: SEQUENCE { 2393 3: OBJECT
IDENTIFIER serialNumber (2 5 4 5) 2398 16: PrintableString
'f92009e853b6b045' : ) : } : } 2416 30: SEQUENCE { 2418 13: UTCTime
26/05/2016 16:28:52 GMT 2433 13: UTCTime 24/05/2026 16:28:52 GMT : }
2448 27: SEQUENCE { 2450 25: SET { 2452 23: SEQUENCE { 2454 3: OBJECT
IDENTIFIER serialNumber (2 5 4 5) 2459 16: PrintableString
'f92009e853b6b045' : ) : } : } 2477 546: SEQUENCE { 2481 13: SEQUENCE
( 2483 9: OBJECT IDENTIFIER : rsaEncryption (1 2 840 113549 1 1 1)
2494 0: NULL : ) 2496 527: BIT STRING, encapsulates { 2501 522:
SEQUENCE { 2505 513: INTEGER : 00 AF B6 C7 82 2B B1 A7 01 EC 2B B4 2E
8B CC 54 : 16 63 AB EF 09 7F 5F 32 7F 75 31 03 9C 77 52 4B : 1B 5F E8
09 FB C7 2A A9 45 1F 74 3C BD 9A 6F 13 : 35 74 4A A5 5E 77 F6 B6 AC
35 35 EE 17 C2 5E 63 : 95 17 DD 9C 92 E6 37 4A 53 CB FE 25 8F 8F FB
B6 : FD 12 93 78 A2 2A 4C A9 9C 45 2D 47 A5 9F 32 01 : F4 41 97 CA 1C
CD 7E 76 2F B5 31 51 B6 FE B2 : FF FF 2B 6F E4 FE 5B C6 BD 9E C3
4B FE 08 23 9D : [ Another 385 bytes skipped ] 3022 3: INTEGER 65537
( 3035 3: OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14) 3040 22:
OCET STRING, encapsulates ( 3042 20: OCTET STRING : 36 61 E1 00 7C
88 05 09 51 8B 44 6C 47 FF 1A 4C : C9 EA 4F 12 : ) : } : } 3064 31:
SEQUENCE { 3066 3: OBJECT IDENTIFIER : authorityKeyIdentifier (2 5 29
35) 3071 24: OCTET STRING, encapsulates ( 3073 22: SEQUENCE { 3075
20: [0] : 36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C : C9 EA 4F
12 : ) : } : } 3097 15: SEQUENCE { 3099 3: OBJECT IDENTIFIER
basicConstraints (2 5 29 15) 3104 1: BOOLEAN TRUE 3107 5: OCTET
STRING, encapsulates { 3109 3: SEQUENCE { 3111 1: BOOLEAN TRUE : ) :
} : } : } 3114 14: SEQUENCE { 3116 3: OBJECT IDENTIFIER keyUsage (2 5 29
15) 3121 1: BOOLEAN TRUE 3124 4: OCTET STRING, encapsulates { 3126 2:
BIT STRING 1 unused bit : '1100001'B : ) : } : } 3130 64: SEQUENCE { 3132
3: OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31) 3137 57: OCTET
STRING, encapsulates { 3139 55: SEQUENCE { 3141 53: SEQUENCE { 3143
51: [0] { 3145 49: [0] { 3147 47: [6] :
sha256WithRSAEncryption (1 2 840 113549 1 1 11) 3209 0: NULL : ) 3211
513: BIT STRING : 20 C8 C3 8D 4B DC A5 0B 00 7E 9C 69 44 FF 72 AA :
C6 F8 44 A1 1D 41 A8 F0 73 6C C3 7D 16 D6 42 6D : 8E 7E 94 07 04 4C
EA 39 E6 8B 07 C1 3D BF 15 03 : DD 5C 85 BD AF B2 C0 2D 5F 6C DB 4E
FA 81 2F DF : 8B 04 F1 82 77 OF C4 E7 74 5B 7F CE AA 87 12 9A :
88 01 CE 8E 9B C0 CB 96 37 9B 4D 26 A8 2D 30 FD : 9C 2F 8E ED 6D C1 BE 2F
84 B6 89 E4 D9 14 25 8B : 14 4B BA E6 24 A1 C7 06 71 13 2E 2F 06 16
A8 84 : [ Another 384 bytes skipped ] : ) 3728 1413: SEQUENCE { 3732
7.2. Windows 10 TPM

The next two sections provide two views of a CSR generated via invocation of the Certificate Enrollment Manager API similar to the below:

```java
request.KeyAlgorithmName = KeyAlgorithmNames.Rsa; request.KeyStorageProviderName = "Microsoft Smart Card Key Storage Provider";
request.UseExistingKey = true; request.Exportable = ExportOption.NotExportable; request.ContainerName = prj.GetContainerName();

request.Subject = subject_name; request.KeyUsages = keyUsages; request.SmartcardReaderName = smartCardReaderName;

string privacyCa = "MIIDezCCAmOgAwIBAgIBATANBgkqhkiG9w0BAQsFADBUMQswCQYDVQQGEwJVUzEyMB0GA1UEChMKVS5TLiBHb3Zlcm5tZW50MQ0wCwYDVQQDExNQdXJlYnJlZCBQcml2YWN5IENBMB4XDTE4MDQwMzE0NTQwMDQwMzE0MB4XDTE4MDQwMzE0NTQwMDQwMzE0IjAwHQYDVR0OBBYEFAFy9PrSM65GYyC0EVDPU91WJ0Bb1xIgEvAgEDAOCAQEAAG777Bu5SExmuoHiVctA0n5u84SZb619Jwv1I3gYpIyGM" + "Mp7GbK4rj/1F7Wv5BbpN/Pp4syi3f+o+yQbSz+zFQwfw/BWGLukTUZyPc DoyleUCwEA" + "AnYMYFWYHQDYVR0BEFyAFy9PrSM65GYyC0EVDFU91WJ0Bb1xIgEvAgEDAOCAQEAAG777Bu5SExmuoHiVctA0n5u84SZb619Jwv1I3gYpIyGM" + "2oxDSKPr36c7R2tFmAg0q4m9997wh4xFebkkYHgZWFsp0hRFy9veE+wMCwz2B8B" + "r14a22/0TDmW9u3r+baJ2KrPmVo9W9eztmz6DJA3wEdvEdvUEZnq4GIV5yXidc8U" + "pfvd4eYFVPWYVN0yP9ZDDB9vVcd5x7VfG8rQoaRcerwrxJ9/WD7z6A6d2/syHn" + "74CRuXYGhpbB7Yl1JhgV16Rb4Dq5dIG1kmVuceEnku73Qddr/phgqOMWvbU" + "1XrHJbPulC+nuPbShhJ0YPRw13TX3degjzTsj8XECa==";

byte[] privacyCaBytes = Convert.FromBase64String(privacyCa); IBuffer buffer = privacyCaBytes.AsBuffer(); request.AttestationCredentialCertificate = new Certificate(buffer); 
```

The structure is essentially a Full PKI Request as described in RFC 5272.

- ContentInfo
  - SignedData
    - PKIData
      - Empty controlSequence
      - One TaggestRequest
        - PKCS 10
          - Basic request details along with encrypted attestation extension
            - Empty cmsSequence
            - Empty otherMsgSequence
          + Certificates bag with two certs (one of which is revoked)

7.2.1. Attestation statement

This section provides an annotation attestation statement as extracted from an encrypted attestation extension. The structure of the attestation statement is defined here: https://msdn.microsoft.com/en-us/library/dn408990.aspx.
The format is structured as follows:

typedef struct { UINT32 Magic; UINT32 Version; UINT32 Platform;
UINT32 HeaderSize; UINT32 cbIdBinding; UINT32 cbKeyAttestation;
UINT32 cbAIKOpaque; BYTE idBinding[cbIdBinding]; BYTE
keyAttestation[cbKeyAttestation]; BYTE aikOpaque[cbAIKOpaque]; } KeyAttestationStatement;

4B 41 53 54 - Magic 01 00 00 00 - Version 02 00 00 00 - Platform 1C
00 00 00 - HeaderSize 00 00 00 00 - cbIdBinding B9 04 00 00 -
cbKeyAttestation 00 00 00 00 - cbAIKOpaque

The remainder is the keyAttestation, which is structured as follows:

typedef struct { UINT32 Magic; UINT32 Platform; UINT32
HeaderSize; UINT32 cbKeyAttest; UINT32 cbSignature; UINT32 cbKeyBlob;
BYTE keyAttest[cbKeyAttest]; BYTE signature[cbSignature]; BYTE
keyBlob[cbKeyBlob]; } keyAttestation;

4B 41 44 53 - Magic 02 00 00 00 - Platform 18 00 00 00 - HeaderSize
A1 00 00 00 - cbKeyAttest (161) 00 01 00 00 - cbSignature (256) 00 03
00 00 - cbKeyBlob

keyAttest (161 bytes) "FF 54 43 47 80 17 00 22 00 0B 9A FD AB 8A 0B
E9 0B BB 3F 7F E6 B6 77 91 EF A9 15 8A 03 B2 2B 8C BE 3F EC 56 B6 30
BF 82 73 9C 00 14 13 6E 2F 14 DD AF 30 72 A6 E3 B9 4D BF 7A 54 26 36
2F 10 D6 00 00 00 00 51 4F CB E5 AD 8C 8C 60 E6 C2 70 80 00 D4 2C 65
4C 6B 95 ED 95 00 22 00 0B 2B E6 2C AD 8D E8 9A 85 04 04 D7 F3 7B B7 4C
F8 32 CD B4 F1 80 CA A6 35 B9 2C 39 87 B7 96 03 C3 A3 00 22 00 0B 6C
88 60 B2 80 E3 BE 7D 34 F2 85 DC 26 9D 1B 72 A8 0A 17 CF 31 08 F1 55
F2 9B 4E 82 C8 5B 49 7B "

The keyAttest field is of type TPMS_ATTEST. The TPMS_ATTEST
structure is defined in section 10.11.8 of
https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-
Part-2-Structures-00.99.pdf. "" FF 54 43 47 - magic 80 17 - type
(TPM_ST_ATTEST_CERTIFY) 00 22 - name - TPM2B_NAME.size (34 bytes) 00
0B 9A FD AB 8A 0B E9 0B BB - TPM2B_NAME.name 3F 7F E6 B6 77 91 EF A9
15 8A 03 B2 2B 8C BE 3F EC 56 B6 30 BF 82 73 9C

00 14 - extraData - TPM2B_DATA.size (20 bytes) 13 6E 2F 14 DD AF 30
72 A6 E3 - TPM2B_DATA.buffer 89 4D BF 7A 54 26 36 2F 10 D6

00 00 00 00 51 4F CB E5 - clockInfo - TPMS_CLOCK_INFO.clock AD 8C 8C
60 - TPMS_CLOCK_INFO.resetCount E6 C2 70 80 -
TPMS_CLOCK_INFO.restartCount 00 -- TPMS_CLOCK_INFO.safe

D4 2C 65 4C 6B 95 ED 95 - firmwareVersion
00 22 - attested - TPMS_CERTIFY_INFO.name.size 00 0B 2B E6 2C AD 8D
E8 9A 85 - TPM2B_NAME.name 04 D7 F3 7B B7 4C F8 32 CD B4 F1 80 CA A6
35 B9 2C 39 87 B7 96 03 C3 A3

00 22 - TPMS_CERTIFY_INFO.qualifiedName.size 00 0B 6C 88 60 B2 80 E3
BE 7D - TPM2B_NAME.name 34 F2 85 DC 26 9D 1B 72 A8 0A 17 CF 31 08 F1
55 F2 9B 4E 82 C8 5B 49 7B ""

Signature (256 bytes) - generated using the AIK private key "1A F1 4B
12 A1 C5 D1 A4 C5 A4 59 C4 9A 97 E0 88 ED 1C D3 B6 38 4A 5D 6C 27 F5
69 7D 17 AD F6 C0 03 27 09 5D 93 B5 13 EA 50 B5 05 27 7B A0 51 4D 1B
17 52 87 7D B8 A6 05 4A 4F 39 CA 36 5C A1 19 19 OB 73 B4 0E 7F D3 91
DA 91 EE 37 C6 CE 78 AF 15 21 5D EB 5E 5F 23 A7 08 E9 85 D4 6B A0 95
6D D7 E0 3A D1 92 72 B7 D4 E5 35 6A 01 B0 7D 35 D0 99 BA A1 77 35 76
75 E3 90 A8 8B 86 27 B8 3D 47 75 2D 98 D0 23 4E 09 D8 26 6B 32 3C AB
AC 50 A2 E8 FF 70 21 85 C5 5E B1 F5 9C B9 6E 21 27 C7 2A CD 84 61 02
47 6A A0 E1 9A 9F AF 02 43 08 D8 BF 9F 69 14 C4 BC 80 32 2D 5C A3 60
48 F5 5E 8E 65 6B 5E B5 0E A4 ED B9 8B F9 C3 D9 A8 CE C0 64 71 F6 E3
81 F7 9D 79 E5 73 7B F3 A4 6E 65 8D 72 B4 0A 3E 5E 70 5F AB 2B 89 B9
5E 65 44 BF 44 7B FB 2E 29 39 64 36 85 63 46 62 AF 25 A5 8B 19 30 AF
"

The remainder is the keyBlob, which is defined here:
https://github.com/Microsoft/TSS.MSR/blob/master/PCPTool.v11/inc/
TpmAtt.h.

7.3. Yubikey

As with the Android Keystore attestations, Yubikey attestations take
the form of an X.509 certificate. As above, the certificate is
presented here packaged along with an intermediate CA certificate as
a certificates-only SignedData message.

The attestations below were generated using code similar to that
found in the yubico-piv-tool (https://github.com/Yubico/yubico-piv-
tool). Details regarding attestations are here:
https://developers.yubico.com/PIV/Introduction/PIV_attestation.html

7.3.1. Yubikey 4

"0 1576: SEQUENCE { 4 9: OBJECT IDENTIFIER signedData (1 2 840 113549
1 7 2) 15 1561: [0] ( 19 1557: SEQUENCE { 23 1: INTEGER 1 26 0: SET
() 2B 11: SEQUENCE { 30 9: OBJECT IDENTIFIER data (1 2 840 113549 1 7
1) : ) 41 1533: [0] ( 45 742: SEQUENCE { 49 462: SEQUENCE ( 53 3: [0]
( 55 1: INTEGER 2 : ) 58 9: INTEGER 00 A4 85 22 AA 34 AF AE 4F 69 13:
SEQUENCE ( 71 9: OBJECT IDENTIFIER : sha256WithRSAEncryption (1 2 840
113549 1 1 11) 82 0: NULL : ) 84 43: SEQUENCE { 86 41: SET ( 88 39:
SEQUENCE ( 90 3: OBJECT IDENTIFIER commonName (2 5 4 3) 95 32:

UTF8String 'Yubico PIV Root CA Serial 263751' : } : } : } 129 32:
SEQUENCE { 131 13: UTCTime 14/03/2016 00:00:00 GMT 146 15:
GeneralizedTime 17/04/2052 00:00:00 GMT : } 163 33: SEQUENCE { 165
31: SET { 167 29: SEQUENCE ( 169 3: OBJECT IDENTIFIER commonName (2 5
4 3) 174 22: UTF8String 'Yubico PIV Attestation' : } : } : } 198 290:
SEQUENCE { 202 13: SEQUENCE { 204 9: OBJECT IDENTIFIER :
rsaEncryption (1 2 840 113549 1 1 1) 215 0: NULL : } 217 271: BIT
STRING : 30 82 01 0A 02 82 01 01 00 AB A9 0B 16 9B EF 31 : CC 3E AC
18 5A 2D 45 80 75 70 C7 58 06 6C 3F 1B : 59 0D 49 B9 89 E8 6F CE BB
27 6F D8 3C 60 3A 85 : 00 EF 5C BC 40 99 3D 41 EE EA C0 81 7F 6E 48
E4 : A9 4C BC D5 6B E1 1F 0A 60 93 C6 FE AA D2 8D 8E : E2 B7 CD 8B 2B
F7 9B DD 5A AB 2F CF B9 0E 54 CE : EC 8D F5 5E D7 97 B1 C3 A7 56 9C
DC C1 06 86 76 : 36 44 53 FB 08 25 D8 06 B9 06 8C 81 FD 63 67 CA : 
17: SEQUENCE { 498 10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3' 510
3: OCTET STRING 04 03 03 : } : } : } 515 13: SEQUENCE { 517 9:
OBJECT IDENTIFIER : sha256WithRSAEncryption (1 2 840 113549 1 1 11)
528 0: NULL : } 530 257: BIT STRING : 52 80 5A 6D 3C 9E DF 47 A8 F1
B2 A5 9C A3 80 81 : 3B 1D 6A EB 6A 12 62 4B 11 FD 8D 30 F1 7B FC 71:
10 C9 B2 08 FC D1 4E 35 7F 45 F2 10 A2 52 D4 : B3 02 01 56 07
6B FA 64 A7 08 F0 03 FB 27 A9 : 60 8D 0D D3 AC 5A 10 CF 20 96 4E 82
BC 9D E3 37 : DA C1 4C 50 E1 3D 16 B4 CA F4 1B FF 08 64 C9 74 : 4F 2A
3A 43 E0 DE 42 79 F2 13 AE 77 A1 E2 AE 6B : DF 72 A5 B6 CE D7 4C 90
13 DF DE DB F2 8B 34 45 : [ Another 128 bytes skipped ] : } 791 783:
SEQUENCE { 795 503: SEQUENCE { 799 3: [0] { 801 1: INTEGER 2 : } 804
17: INTEGER : 00 FE B9 AF 03 3B 0B A7 79 04 02 F5 67 AE DF 72 : ED
823 13: SEQUENCE { 825 9: OBJECT IDENTIFIER : sha256WithRSAEncryption (1 2 840 113549 1 1 11) 836 0: NULL : } 838 33: SEQUENCE { 840 31:
SET { 842 29: SEQUENCE { 844 3: OBJECT IDENTIFIER commonName (2 5 4
3) 849 22: UTF8String 'YubiKey PIV Attestation' : } : } : } 873 32:
SEQUENCE { 875 13: UTCTime 14/03/2016 00:00:00 GMT 890 15:
GeneralizedTime 17/04/2052 00:00:00 GMT : } 907 37: SEQUENCE { 909
35: SET { 911 33: SEQUENCE { 913 3: OBJECT IDENTIFIER commonName (2 5 4
3) 918 26: UTF8String 'YubiKey PIV Attestation 9e' : } : } : } 946
290: SEQUENCE { 950 13: SEQUENCE { 952 9: OBJECT IDENTIFIER :
rsaEncryption (1 2 840 113549 1 1 1) 963 0: NULL : } 965 271: BIT
STRING : 30 82 01 0A 02 82 01 01 00 93 C4 C0 35 95 7E 26 : 2A 7E A5
D0 29 C4 D7 E9 39 67 22 B1 09 45 46 4D : DB A4 77 CB 0B A3 F1 D0 69
3C 24 8D A2 72 72 27 : E1 7F DE CB 67 A4 1D D2 E5 43 44 6F 21 39 F8
57 : 34 01 0E 7E C3 81 63 63 6A 6D D7 40 20 7B AF 35 : 61 9C 8D C1 D1
2B 25 48 EE 52 FC F3 72 6A 74 96 : 01 CB 1C 1A B2 AD F9 18 96 EB 59
EF E3 3A CA BC : AA 9B 42 FE FF 60 6E 28 89 49 0D C1 B1 B0 25 AE : 
[ Another 142 bytes skipped ] : } 1240 60: [3] { 1242 58: SEQUENCE { 1244
17: SEQUENCE { 1246 10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3' 1258
3: OCTET STRING 04 03 03 -- firmware version : } 1263 19:
SEQUENCE { 1265 10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 7' 1277 5:
OCTET STRING 02 03 4F 9B B5 -- serial number : } 1284 16: SEQUENCE { 1286
10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 B' 1298 2: OCTET
STRING 01 01 -- PIN and touch policy : } : } : } : } 1302 13:
SEQUENCE { 1304 9: OBJECT IDENTIFIER : sha256WithRSAEncryption (1 2
840 113549 1 1 11) 1315 0: NULL : } 1317 257: BIT STRING : 1F 2B B8
1C 95 A1 01 74 3F 87 27 F6 B3 A6 A9 9D : 11 B9 ED 68 92 B9 05 2D 22
36 51 28 23 3D B0 2F : 7A 17 D5 8C 0C F4 3A 68 FD 2A 34 0D 80 3C F7
8F : B8 79 B0 76 E5 61 94 C5 72 D6 9F 6E 26 76 5F : 03 94 55 40 93
5C 04 EF CC 58 41 EB 7C 86 64 23 : 5F 23 5E 94 78 73 2E 77 8C 58 C5
45 87 22 CF BA : 69 06 B8 C7 06 37 10 21 8C 74 AD 08 B9 85 F2 7B : 99
02 4A 3E E8 96 09 D3 F4 C6 AB FA 49 68 E2 E0 : [ Another 128 bytes
skipped ] : } : } 1578 0: SET {} : } : } : } "

7.3.2.  Yubikey 5

"0 1613: SEQUENCE { 4 9: OBJECT IDENTIFIER signedData (1 2 840 113549
1 7 2) 15 1598: [0] ( 19 1594: SEQUENCE ( 23 1: INTEGER 1 26 0: SET
( ) 28 11: SEQUENCE ( 30 9: OBJECT IDENTIFIER data (1 2 840 113549 1 7
1) : ) 41 1570: [0] ( 45 762: SEQUENCE ( 49 482: SEQUENCE ( 53 3: [0]
( 55 1: INTEGER 2 : ) ) 58 9: INTEGER 00 86 77 17 E0 1D 19 2B 26 69 13:
SEQUENCE ( 71 9: OBJECT IDENTIFIER : sha256WithRSAEncryption (1 2 840
113549 1 1 11) 82 0: NULL : ) 84 43: SEQUENCE ( 86 41: SET { 88 39:
SEQUENCE ( 90 3: OBJECT IDENTIFIER commonName (2 5 4 3) 95 32:
UTF8String 'Yubico PIV Root CA Serial 263751' : ) : ) : ) 129 32:
SEQUENCE ( 131 13: UTCTime 14/03/2016 00:00:00 GMT 146 15:
GeneralizedTime 17/04/2052 00:00:00 GMT : ) 163 33: SEQUENCE { 165
31: SET ( 167 29: SEQUENCE ( 169 3: OBJECT IDENTIFIER commonName (2 5
4 3) 174 22: UTF8String 'Yubico PIV Attestation' : ) : ) : ) 198 290:
SEQUENCE ( 202 13: SEQUENCE ( 204 9: OBJECT IDENTIFIER :
rsaEncryption (1 2 840 113549 1 1 1) 215 0: NULL : ) 217 271: BIT
STRING : 30 82 01 0A 02 82 01 01 00 C5 5B 8D E9 B9 3C 53 : 69 82 88
FE DA 70 FC 5C 88 78 41 25 A2 1D 7B 84 : 8E 93 36 AD 67 2B 4C AB 45
BE B2 E0 D5 9C 1B A1 : 68 D5 68 F8 63 5C 83 CB 83 38 62 B7 64 AE 83
37 : 37 8E C8 60 80 86 01 F8 75 AA AE F6 6E A7 D5 76 : C5 C1 25 AD AA
9E 9D DC B5 7E E9 8E 2A B4 3F 99 : OD F7 9F 20 A0 28 A0 9F B3 B1 22
5F AF 38 FB 73 : 46 F4 C7 93 30 DD FA D0 86 E0 C9 C6 72 99 AF FB : [ Another 142 bytes skipped ] : ) 492 41: [3] ( 494 39: SEQUENCE ( 496
17: SEQUENCE ( 498 10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3* 510
3: OCTET STRING 05 01 02 : ) 515 18: SEQUENCE { 517 3: OBJECT
IDENTIFIER basicConstraints (2 5 29 19) 522 1: BOOLEAN TRUE 525 8:
OCTET STRING 30 06 01 01 FF 02 01 00 : ) : ) : ) : ) : ) 535 13: SEQUENCE
( 537 9: OBJECT IDENTIFIER : sha256WithRSAEncryption (1 2 840 113549
1 1 11) 548 0: NULL : ) 550 257: BIT STRING : 05 57 B7 BF 5A 41 74 F9
5F EC 2E D2 B8 78 26 E5 : EF 4F EA BF 5A 64 C9 CF 06 7F CA 8C 0A FC
1A 47 : 1C D6 AC ED C8 5B 54 72 00 9F B8 59 AB 73 25 B2 : D6 02 A3 59
83 31 69 EE C1 5F 3D F2 2B 1B 22 CA : B6 FC F9 FB 21 32 9E 08 F3 08
54 6D C9 26 10 42 : 08 1D 3C B5 F0 5A B1 98 D4 6B DC 91 F1 D3 91 54 :
7A A0 34 8B F6 65 EB 13 9F 3A 1C BF 43 C5 D1 D0 : 33 23 C6 25 A0 4C
E4 E9 AA 59 80 D8 02 1E B0 10 : [ Another 128 bytes skipped ] : ) 811
800: SEQUENCE { 815 520: SEQUENCE { 819 3: [0] ( 821 1: INTEGER 2 : )

8. Privacy Considerations.

TBD

9. Security Considerations

TBD.
10. IANA Considerations

   TBD.

11. Acknowledgements

    Thomas Hardjono provided the text on blockchain system. Dave Thaler suggested many small variations. Frank Xiaoliang suggested the scaling scenarios that might preclude a 1:1 protocol between attesters and relying parties. Henk Birkholz provided many reviews. Kathleen Moriarty provided many useful edits. Ned Smith, Anders Rundgren and Steve Hanna provided many useful pointers to TCG terms and concepts. Thomas Fossati and Shawn Willden elucidated the Android Keystore goals and limitations.

12. References

12.1. Normative References


12.2. Informative References


[fidosignature]

[fidotechnote]

[I-D.gutmann-scep]

[I-D.tschofenig-rats-psa-token]

[ieee802-1AR]

[intelsgx]

[keystore]

[keystore_attestation]

Internet-Draft                 useful RATS                     July 2019


Appendix A. Changes

- created new section for target use cases
- added comments from Guy, Jessica, Henk and Ned on TCG description.

Authors’ Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Richardson, et al.         Expires January 9, 2020
Carl Wallace
Red Hound Software

Email: carl@redhoundsoftware.com

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com