Abstract

The Remote ATtestation procedures (RATS) architecture facilitates interoperability of attestation mechanisms by defining a set of participant roles and interactions that reveal information about the trustworthiness attributes of an attester’s computing environment. By making trustworthiness attributes explicit, they can be evaluated dynamically and within an operational context where risk mitigation depends on having a more complete understanding of the possible vulnerabilities germane to the attester’s environment.
1. Introduction

The long-standing Internet Threat Model [RFC3552] focuses on threats to the communication channel, as pioneered by Dolev and Yao [DOLEV-YAO] in 1983. However, threats to the endpoint [RFC5209] and system components [RFC4949] of transited communication gear (i.e. hosts) are increasingly relevant for assessing the trustworthiness properties of a communication channel. Beyond the collection and conveyance of security posture [RFC5209] about an endpoint (host), remote attestation provides believable trustworthiness claims ("Evidence") about an endpoint (host). In general, this document provides normative guidance how to use, create or adopt network protocols that facilitate RATS.
1.1. RATS in a Nutshell

The RATS architecture provides a basis to assess the trustworthiness of endpoints by other parties:

- In remote attestation workflows, trustworthiness Claims are accompanied by a proof of veracity. Typically, this proof is a cryptographic expression such as a digital signature or message digest. Trustworthiness Claims with proof is what makes attestation Evidence believable.

- A corresponding attestation provisioning workflow uses trustworthiness Claims to convey believable Endorsements and Known-Good-Values used by a Verifier to appraise Evidence.

In the RATS architecture, specific content items are identified (and described in more detail below):

- Evidence is provable Claims about a specific Computing Environment made by an Attester.

- Known-Good-Values are reference Claims used to appraise Evidence.

- Endorsements are reference Claims about the environment protecting the Attester's capabilities to create believable Evidence (e.g. the type of protection for an attestation key). It answers the question "why Evidence is believable".

- Attestation Results are the output from the appraisal of Evidence, Known-Good-Values and Endorsements.

Attestation Results are the output of RATS. Assessment of Attestation Results can be multi-faceted, but is out-of-scope for the RATS architecture. If appropriate Endorsements about the Attester are available, Known-Good-Values about the Attester are available, and if the Attester is capable of creating believable Evidence - then the Verifier is able to create Attestation Results that enable Relying Parties to establish a level of confidence in the trustworthiness of the Attester.

2. Terminology

Conveyance: a mechanism for transferring RATS Evidence, Endorsements, Known-Good-Values or Attestation Results.

Entity: a user, organization, device or computing environment.
Principal: an Entity that implements RATS Roles and creates provable Claims or Attestation Results (see [ABLP] and [Lampson2007]).

Trustworthiness: an expectation about a computing environment that it will behave in a way that is intended and nothing more.

Computing Environment: a computing context consisting of system components.


2.1. Requirements Notation

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

3. Conceptual Overview

In network protocol exchanges, it is often the case that one entity (a Relying Party) requires an assessment of the trustworthiness of a remote entity (an Attester or specific system components [RFC4949] thereof). Remote ATtestation procedureS (RATS) enable Relying Parties to establish a level of confidence in the trustworthiness of remote system components through the creation of attestation evidence by remote system components and a processing chain of architectural constituents towards the relying party.

The corresponding trustworthiness attributes processed may not be just a finite set of values. Additionally, the system characteristics of remote components themselves have an impact on the veracity of trustworthiness attributes included in Evidence. Attester environments can vary widely ranging from those highly resistant to attacks to those having little or no resistance to attacks. Configuration options, if set poorly, can result in a highly resistant environment being operationally less resistant. Computing Environments are often malleable being constructed from re-programmable hardware, firmware, software and updatable memory. When a trustworthy environment changes, the question has to be asked whether the change transitioned the environment from a trustworthy state to an untrustworthy state. The RATS architecture provides a framework for anticipating when a relevant change with respect to a
trustworthiness attribute occurs, what changed and how relevant it is. A remote attestation framework also creates a context for enabling an appropriate response by applications, system software and protocol endpoints when changes to trustworthiness attributes do occur.

3.1. Computing Environments

In the RATS context, a Claim is a specific trustworthiness attribute that pertains to a particular Computing Environment of an Attester. The set of possible Claims is expected to follow the possible computing environments that support attestation. In other words, identical (i.e. same type, model, versions, components and composition) Attesting Computing Environments can create different Claim values that still compose valid Evidence due to different computing contexts. Exemplary Claims include flight vectors or learned configuration.

Likely, there are a set of Claims that is widely applicable across most, if not all environments. Conversely, there are Claims that are unique to specific environments. Consequently, the RATS architecture incorporates extensible mechanisms for representing Claims.

Computing Environments can be complex structurally. In general, every Attester consists of multiple components (e.g. memory, CPU, storage, networking, firmware, software). Components are computational elements that can be linked and composed to form computational pipelines, arrays and networks (e.g. a BIOS, a bootloader, or a trusted execution environment).

An Attester includes at least one Computing Environment that is able to create attestation Evidence (the Attesting Computing Environment) about other Computing Environments (the Attested Computing Environments). Not every computational element of an Attester is expected to be a Computing Environment capable of remote attestation. Analogously, remote attestation capable Computing Environments may not be capable of attesting to (creating evidence about) every computational element that interacts with the Computing Environment. A Computing Environment with an attestation capability can only be endorsed by an external entity and cannot create believable evidence about itself by its own.

A Computing Environment with the capability of remote attestation:

- is separate from other Attested Computing Environments (about which attestation evidence is created), and
- is enabling the role of an Attester in the RATS architecture.
A Computing Environment with the capability of remote attestation and taking on the role of an Attester has the following duties in order to create Evidence:

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

### 3.2. Trustworthiness

The trustworthiness of remote attestation capabilities is also a consideration for the RATS architecture. It should be possible to understand the trustworthiness properties of the remote attestation capability for any set of claims of a remote attestation flow via verification operations. The RATS architecture anticipates recursive trustworthiness properties and the need for termination. Ultimately, a portion of a computing environment’s trustworthiness is established via non-automated means. For example, design reviews, manufacturing process audits and physical security. For this reason, trustworthy RATS depend on trustworthy manufacturing and supply chain practices.

### 3.3. RATS Workflow

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

RATS architecture defines attestation Roles (i.e., Attester, Verifier, Asserter and Relying Party), the messages they exchange, their structure and the various legal ways in which Roles may be
hosted, combined and divided (see Principals below). RATS messages are defined by an information model that defines Claims, environment and protocol semantics. Information Model representations are realized as data structure and conveyance protocol binding specifications.

3.4. Interoperability between RATS

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

4. RATS Architecture

4.1. Goals

RATS architecture has the following goals:

- Enable semantic interoperability of attestation semantics through information models about computing environments and trustworthiness.

- Enable data structure interoperability related to claims, endpoint composition / structure, and end-to-end integrity and confidentiality protection mechanisms.

- Enable programmatic assessment of trustworthiness. (Note: Mechanisms that manage risk, justify a level of confidence, or determine a consequence of an attestation result are out of scope).

- Provide the building blocks, including Roles and Principals that enable the composition of service-chains/hierarchies and workflows that can create and appraise evidence about the trustworthiness of devices and services.

- Use-case driven architecture and design (RATS use cases are summarized in [I-D.richardson-rats-usecases]).

- Terminology conventions that are consistently applied across RATS specifications.

- Reinforce trusted computing principles that include attestation.
4.2. Attestation Principles

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

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

4.3. RATS Roles and Messages

The RATS Roles (roles) are performed by RATS Principals.

The RATS Architecture provides the building blocks to compose various RATS roles by leveraging existing and new protocols. It defines architecture for composing RATS roles with principals and models their interactions.

Figure 1 provides an overview of the relationships between RATS Roles and the messages they exchange.
4.3.1. Roles

RATS roles are implemented by principals that possess cryptographic keys used to protect and authenticate Claims or Results.

Attester: An Attestation Function that creates Evidence by collecting, formatting and protecting (e.g., signing) Claims. It presents Evidence to a Verifier using a conveyance mechanism or protocol.

Verifier: An Attestation Function that accepts Evidence from an Attester using a conveyance mechanism or protocol. It also accepts Known-Good-Values and Endorsements from an Asserter using a conveyance mechanism or protocol. It verifies the protection mechanisms, parses and appraises Evidence according to good-known valid (or known-invalid) Claims and Endorsements. It produces Attestation Results that are formatted and protected (e.g., signed). It presents Attestation Results to a Relying Party using a conveyance mechanism or protocol.

Asserter: An Attestation Function that generates reference Claims about both the Attesting Computing Environment and the Attested Computing Environment. The manufacturing and development processes are presumed to be trustworthy processes. In other words the Asserter is presumed, by a Verifier, to produce valid Claims. The function collects, formats and protects (e.g. signs)
valid Claims known as Endorsements and Known-Good-Values. It presents provable Claims to a Verifier using a conveyance mechanism or protocol.

Relying Party: An Attestation Function that accepts Attestation Results from a Verifier using a conveyance mechanism or protocol. It assesses Attestation Results protections, parses and assesses Attestation Results according to an assessment context (Note: definition of the assessment context is out-of-scope).

4.3.2. Role Messages

Claims: Statements about trustworthiness characteristics of an Attested Computing Environment.

The veracity of a Claim is determined by the reputation of the entity making the Claim. (Note: Reputation may involve identifying, authenticating and tracking transactions associated with an entity. RATS may be used to establish entity reputation, but not exclusively. Other reputation mechanisms are out-of-scope).

Evidence: Claims that are formatted and protected by an Attestor.

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

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

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

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

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

Attestation Results: Statements about the output of an appraisal of Evidence that are created, formatted and protected by a Verifier.
Attestation Results provide the basis for a Relying Party to establish a level of confidence in the trustworthiness of an Attester. Attestation Results SHOULD satisfy Relying Party expectations for freshness, identity, context, provenance, validity, relevance and veracity.

4.4. RATS Principals

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

RATS Principals may implement one or more RATS Roles. Role interactions occurring within the same RATS Principal are out-of-scope.

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

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

---

![Diagram of RATS Principals-Role Composition](image)

Figure 2: RATS Principals-Role Composition

RATS Principals have the following properties:

- Multiplicity - Multiple instances of RATS Principals that possess the same RATS Roles can exist.

- Composition - RATS Principals possessing different RATS Roles can be combined into a singleton RATS Principal possessing the union
of RATS Roles. RATS Interactions between combined RATS Principals is uninteresting.

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

RATS Interactions may occur between them.

5. Security Considerations

RATS Evidence, Verifiable Assertions and Results SHOULD use formats that support end-to-end integrity protection and MAY support end-to-end confidentiality protection. Replay attack prevention MAY be supported if a Nonce Claim is included. Nonce Claims often piggyback other information and can convey attestation semantics that are of essence to RATS, e.g. the last four bytes of a challenge nonce could be replaced by the IPv4 address-value of the Attester in its response.

All other attacks involving RATS structures are not explicitly addressed by RATS architecture. Additional security protections MAY be required of conveyance mechanisms. For example, additional means of authentication, confidentiality, integrity, replay, denial of service and privacy protection of RATS payloads and Principals may be needed.

6. References

6.1. Normative References


6.2. Informative References

[DOLEV-YAO]

[I-D.richardson-rats-usecases]

[Lampson2007]


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

Ned Smith  
Intel Corporation  
USA  
Email: ned.smith@intel.com
YANG Module for Basic Challenge-Response-based Remote Attestation Procedures
draft-birkholz-rats-basic-yang-module-01

Abstract

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

Status of This Memo

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

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

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

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

```yang
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
```

Internet-Draft                    BRAT                         July 2019

++-w (key-identifier)?
  +++:(public-key)
    |  +++-w pub-key-id? binary
  +++:(TSS_UUID)
    +++-w TSS_UUID-value
      +++-w ulTimeLow? uint32
      +++-w usTimeMid?  uint16
      +++-w usTimeHigh? uint16
      +++-w bClockSeqHigh? uint8
      +++-w bClockSeqLow? uint8
      +++-w rgbNode*    uint8
    +++-w add-version? boolean
    +++-w tpm_name?   string
      +++-w tpm-physical-index? int32 {ietfhw:entity-mib}?
  ---ro output
    +++-ro tpm2-attestation-response* [tpm_name]
      +++-ro tpm_name   string
      +++-ro tpm-physical-index? int32 {ietfhw:entity-mib}?
      +++-ro up-time?   uint32
      +++-ro node-name? string
      +++-ro node-physical-index? int32 {ietfhw:entity-mib}?
      +++-ro fixed?     binary
      +++-ro external-data? binary
      +++-ro signature-size? uint32
      +++-ro signature? binary
      +++-ro (tpm2-quote)
        +++:(tpm2-quote1)
          +++-ro version* []
            |  +++-ro major? uint8
            |  +++-ro minor? uint8
            |  +++-ro revMajor? uint8
            |  +++-ro revMinor? uint8
            +++-ro digest-value? binary
        +++-ro TPM_PCR_COMPOSITE* []
          +++-ro pcr-indices* uint8
          +++-ro value-size? uint32
          +++-ro tpm2-pcr-value* binary
        +++:(tpm2-quote2)
          +++-ro tag? uint8
          +++-ro pcr-indices* uint8
          +++-ro locality-at-release? uint8
          +++-ro digest-at-release? binary
  ---x tpm20-challenge-response-attestation
    +++-w input
      |  +++-w tpm20-attestation-challenge
      |    +++-w pcr-list* []
      |      |  +++-w pcr
      |      |    +++-w pcr-indices* uint8

Birkholz, et al. Expires January 9, 2020
2.2. Raw Format

<CODE BEGINS>
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" {

<CODE ENDS>
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 multiplexed with
                the variant.";
            }
        }
    }
}
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.";
}
}

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."
    }
  }
  list version-info {
    description "An optional output parameter from a TPM 1.2 TPM_Quote2."
    leaf tag {
      type uint16;
      description "The TPM 1.2 version and revision (TPM 1.2 type TPM_STRUCTURE_TAG). This MUST be TPM_CAP_VERSION_INFO (0x0030)"
    }
    uses tpm12-version;
    leaf spec-level {
      type uint16;
      description "A number indicating the level of ordinals supported."
    }
    leaf errata-rev {
      type uint8;
      description "A number indicating the errata version of the specification."
    }
    leaf tpm-vendor-id {
      type binary;
      description "The vendor ID unique to each TPM manufacturer."
    }
    leaf vendor-specific-size {
      type uint16;
    }
}
description
    "The size of the vendor-specific area.";
}
leaf vendor-specific {
    type binary;
    description
    "Vendor specific information.";
}

"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.";
    }
}

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

description
    "The type of logs available.";

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

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

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

grouping boot-event-log {
  description
    "Defines an event log corresponding to the event that extended the PCR";
  leaf event-number {
    type uint32;
    description
      "Unique event number of this event";
  }
  leaf event-type {
    type uint32;
    description
      "log event type";
  }
  leaf pcr-index {
    type uint16;
    description
      "Defines the PCR index that this event extended";
  }
  list digest-list {
    description "Hash of event data";
  }
}
uses hash-algo;
leaf-list digest {
    type binary;
    description
    "The hash of the event data";
}
leaf event-size {
    type uint32;
    description
    "Size of the event data";
}
leaf-list event-data {
    type uint8;
    description
    "the event data size determined by event-size";
}

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

leaf pcr-index {
    type uint16;
    description
    "Defines the PCR index that this event extended";
}

leaf signature {
    type binary;
    description
    "The file signature";
}

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

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

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

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

rpc tpm12-challenge-response-attestation {
    description "This RPC accepts the input for TSS TPM 1.2 commands of the managed device. ComponentIndex from the hardware manager YANG module to refer to dedicated TPM in composite devices, e.g. smart NICs, is still a TODO.";
    input {
        container tpm1-attestation-challenge {
            description "This container includes every information element defined in the reference challenge-response interaction model for remote attestation. Corresponding values are based on TPM 1.2 structure definitions";
            uses tpm12-pcr-selection;
            uses nonce;
            uses tpm12-signature-scheme;
            uses tpm12-attestation-key-identifier;
            leaf add-version {
                type boolean;
                description "Whether or not to include TPM_CAP_VERSION_INFO; if true, then TPM_Quote2 must be used to create the response.";
            }
            uses tpm-name;
        }
    }
}
output {
  list tpm12-attestation-response {
    key tpm_name;
    description "The binary output of TPM 1.2 TPM_Quote/TPM_Quote2, including the PCR selection and other associated attestation evidence metadata";
    uses tpm-name;
    uses node-uptime;
    uses compute-node;
    uses tpm12-quote-info-common;
    choice tpm12-quote {
      mandatory true;
      description "Either a tpm12-quote-info or tpm12-quote-info2, depending on whether TPM_Quote or TPM_Quote2 was used (cf. input field add-verson).";
      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-uptime;
        uses compute-node;
        container tpms-attest {
            leaf pcrdigest {
                type binary;
                description
                    "split out value of TPMS_QUOTE_INFO for convenience";
            }
            leaf tpms-attest-result {
                type binary;
                description
                    "The complete TPM generate structure including signature.";
            }
            leaf tpms-attest-result-length {
                type uint32;
                description
                    "Length of attest result provided by the TPM structure.";
            }
            description
                "A composite of value and length and list of selected 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 leafs.";
        }
    }
}
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;
    }
}

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.

```yang
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 nodeuptime;
            uses tpm-name;
            container log-result {
                description
                "The requested entries of the corresponding log."
                uses event-logs;
            }
        }
    }
}
}
container rats-support-structures {
    config false;
    description
    "The datastore definition enabling verifiers or relying
    parties to discover the information necessary to use the
    remote attestation RPCs appropriately.";
    leaf-list supported-algos {
        type uint16;
        description
        "Supported TPM_ALG_ID values for the TPM in question.
        Will include ComponentIndex soon.";
    }
    list tpms {
        key tpm_name;
        uses tpm-name;
        description
        "A list of TPMS in this composite
device that rats can be conducted with.

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

list compute-nodes {
    key node-name;
    uses compute-node;
    description "A list names of hardware components in this composite device that rats can be conducted with.";
}
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.Xialiang@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 RFC 8243.
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ................................................. 2
   1.1. Document Structure ................................... 3
2. RATS Information Elements ................................. 4
3. Security Considerations .................................... 8
4. Acknowledgments ............................................... 8
5. Change Log .................................................. 8
6. Contributors .................................................. 8
7. References .................................................... 8
   7.1. Normative References ................................... 8
   7.2. Informative References ................................. 9
Authors’ Addresses ............................................... 9

1. Introduction

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.

Attester 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 Attester Identifier.

Attester 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 Attester 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]


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-psa-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-psy-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]

[I-D.birkholz-rats-basic-yang-module]

[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]
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
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.

```
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 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 used as a suitable communication protocol that preserves confidentiality. In private networks, such as carrier management networks, it must be evaluated whether or not the transport medium is considered confidential.

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

  o Initial draft -00

  o Changes from version 00 to version 01:
    * Added details to the flow diagram

  o Changes from version 01 to version 02:
    * Integrated comments from Ned Smith (Intel)
    * Reorganized sections and
    * Updated interaction model

  o 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,
"Architecture and Reference Terminology for Remote
Attestation Procedures", draft-birkholz-rats-
ar-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 Quotes) (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
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[16.]), 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" [I-D.birkholz-suit-coswid-manifest].

The TCG has done exploratory work in defining formats for reference integrity manifests under the working title TCG Reference Integrity Manifest [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[PC-Client-BIOS-TPM-1.2])
- UEFI BIOS event log (TCG EFI Platform Specification for TPM Family 1.1 or 1.2, Section 7 [EFI])
- Canonical Event Log [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 [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]:

```
<table>
<thead>
<tr>
<th>Attester</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;-------- requestAttestation(nonce, authSecID, claimSelection)</td>
<td></td>
</tr>
<tr>
<td>collectAssertions(assertionsSelection)</td>
<td>assertions</td>
</tr>
<tr>
<td>=&gt; assertions</td>
<td></td>
</tr>
<tr>
<td>signAttestationEvidence(authSecID, assertions, nonce)</td>
<td>=&gt; signedAttestationEvidence</td>
</tr>
<tr>
<td></td>
<td>signedAttestationEvidence</td>
</tr>
<tr>
<td></td>
<td>verifyAttestationEvidence(signedAttestationEvidence, refassertions)</td>
</tr>
</tbody>
</table>
```

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.

- In 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.

- In 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]).

- In 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.ieft-sacm-coswid]. Reference measurements are signed by the device manufacturer.

Fedorkow & Fitzgerald-McKExpires January 1, 2020 [Page 19]
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.

```
+-----------------------+              +-------------------------+
|                       |              |                         |
|       Attester        |<-------------|        Verifier         |
|       (Device)        |------------->|   (Management Station)  |
|                       |      |       |                         |
+-----------------------+      |       +-------------------------+
|Reference Integrity Measurements| | Attestation |
```

********************************************************************
* IETF Attestation Reference Interaction Diagram *
********************************************************************

. Reference Integrity .  . TAPS (PTS2.0) Info .  

*****************************************************************************
* YANG SWID Module .  . TCG .  . YANG Attestation *  
* I-D.birkholz-yang-swid* . Attestation. * Module *  
* . . MIB .  . I-D.birkholz-yang-*  
* . . . . . . * basic-remote-*  
* . . . . . . * attestation *  
*****************************************************************************

*****************************************************************************
* XML, JSON, CBOR (etc) *  * UDF *  * XML, JSON, CBOR (etc)*  
*****************************************************************************

*****************************************************************************
* RESTCONF/NETCONF *  * RESTCONF/NETCONF *  
*****************************************************************************

*****************************************************************************
* TLS, SSH *  * TLS, SSH *  
*****************************************************************************
IETF documents are captured in boxes surrounded by asterisks. TCG documents are shown in boxes surrounded by dots. The IETF Attestation Reference Interaction Diagram, Reference Integrity Measurement Manifest, TAPS Information Model and Canonical Log Format, and both YANG modules are works in progress. Information Model layers describe abstract data objects that can be requested, and the corresponding response SNMP is still widely used, but the industry is transitioning to YANG, so in some cases, both will be required. TLS Authentication with TPM has been shown to work; SSH authentication using TPM-protected keys is not as easily done [as of 2019]

Figure 5: RIV Protocol Stacks

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>TCG RoT</td>
</tr>
<tr>
<td>o Attestation depends entirely on a secure root of trust for measurement.</td>
<td>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 on Roots of Trust</td>
<td></td>
</tr>
<tr>
<td>Get a TPM properly provisioned as described in TCG documents.</td>
<td>TCG TPM DevID TCG Platform Certificate</td>
</tr>
<tr>
<td>Put a DevID or Platform Cert in the TPM</td>
<td>TCG TPM DevID TCG Platform Certificate</td>
</tr>
<tr>
<td>o Install an Initial Attestation Key at the same time so that Attestation can work out of the box</td>
<td>IEEE 802.1AR</td>
</tr>
<tr>
<td>o Equipment suppliers and owners may want to implement Local Device ID as well as Initial Device ID</td>
<td></td>
</tr>
<tr>
<td>Connect the TPM to the TLS stack</td>
<td>Vendor TLS stack (This action is simply configuring TLS to use the DevID as its trust anchor.)</td>
</tr>
<tr>
<td>o Use the DevID in the TPM to authenticate TAP connections, identifying the device</td>
<td></td>
</tr>
<tr>
<td>Make CoSWID tags for BIOS/LoaderLKernel objects</td>
<td>IETF CoSWID ISO/IEC 19770-2</td>
</tr>
<tr>
<td>o Add reference measurements into SWID tags</td>
<td>NIST IR 8060</td>
</tr>
<tr>
<td>o Manufacturer should sign the SWID tags</td>
<td>TagVault SWID</td>
</tr>
<tr>
<td>o This should be covered in a new TCG Reference Integrity Manifest document</td>
<td>Tag Signing Guidance</td>
</tr>
<tr>
<td>- IWG should define the literal SWID format</td>
<td></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)
  - There is no need to specify where the tags are stored in a vendor OS, as long as there is a standards-based mechanism to retrieve them.

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

TCG PC Client BIOS
There have been proposals for non-PC-Client allocation of PCRs, although no specific document exists yet.

TCG SNMP MIB
YANG Module for Basic Attestation
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.
Figure 7: Attestation for End User Computers

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

[I-D.ietf-sacm-coswid]  

[IEEE-802-1AR]  

[IMA]  
and, "Integrity Measurement Architecture", June 2019,  

[NetEq]  
Trusted Computing Group, "TCG Guidance for Securing Network Equipment", January 2018,  

[NIST-IR-8060]  
National Institute for Standards and Technology, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags", April 2016,  

[NIST-SP-800-155]  
National Institute for Standards and Technology, "BIOS Integrity Measurement Guidelines (Draft)", December 2011,  

[PC-Client-BIOS-TPM-1.2]  
Trusted Computing Group, "TCG PC Client Specific Implementation Specification for Conventional BIOS, Specification Version 1.21 Errata, Revision 1.00", February 2012,  
[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.
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 ............................................. 14
  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

Mandyam, et al. Expires January 5, 2020
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 ECDAE (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 ECDAE.
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 generate 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)</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 content.
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

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 enabled 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

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

```c
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
}
```

```plaintext
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

```
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_atClaim = (  
    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
ueid = "ueid"
origination = "origination"
oemid = "oemid"
security_level = "security_level"
boot_state = "boot_state"
location = "location"
age = "age"
uptime = "uptime"
nested_eat = "nested_eat"
submods = "submods"

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 section 2 of [RFC7519].
- string_or_uri - must be encoded as StringOrURI as described section 2 of [RFC7519].

4.4. CBOR

4.4.1. Labels

ueid = 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.

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

```cddl
callm = {  
  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

[IANA.CWT.Claims]

[IANA.JWT.Claims]


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-05

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 April 8, 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
2. Terminology ............................................. 3
   2.1. Static attestations .................................. 4
   2.2. Session attestations ................................ 4
   2.3. Statements .......................................... 4
   2.4. Hardware Root Of Trust ............................ 4
   2.5. Template for Use cases ............................ 5
3. Requirements Language .................................... 5
4. Overview of Sources of Use Cases ......................... 6
5. Use case summaries ....................................... 6
   5.1. Device Capabilities/Firmware Attestation .......... 6
      5.1.1. Relying on an (third-party) Attestation Server .. 7
      5.1.2. Autonomous Relying Party ....................... 7
      5.1.3. Proxy Root of Trust ............................ 8
      5.1.4. network scaling - small ........................ 8
      5.1.5. network scaling - medium ....................... 9
      5.1.6. network scaling - large ....................... 9
   5.2. Hardware resiliency / watchdogs .................... 10
   5.3. IETF TEEP WG use case ............................. 10
   5.4. Confidential Machine Learning (ML) model .......... 11
   5.5. Critical infrastructure ............................ 11
      5.5.1. Computation characteristics ................... 12
   5.6. Virtualized multi-tenant hosts .................... 13
   5.7. Cryptographic Key Attestation ...................... 13
      5.7.1. Device Type Attestation ....................... 14
      5.7.2. Key storage attestation ....................... 14
      5.7.3. End user authorization ........................ 15
   5.8. Geographic attestation ............................. 15
      5.8.1. I am here ...................................... 16
      5.8.2. I am near ...................................... 16
      5.8.3. You are here .................................... 16
   5.9. Connectivity attestation ........................... 16
   5.10. Component connectivity attestation ................ 17
   5.11. Device provenance attestation ..................... 17
6. Technology users for RATS ................................ 18
   6.1. Trusted Computing Group Remove Integrity Verification
        (TCG-RIV) ........................................... 18
   6.2. Android Keystore system ............................ 20
   6.3. Fast IDentity Online (FIDO) Alliance ................ 21
7. Examples of Existing Attestation Formats. ................. 22
  7.1. Android Keystore ............................. 22
    7.1.1. TEE .................................. 23
    7.1.2. Secure Element .......................... 35
  7.2. Windows 10 TPM ................................ 48
    7.2.1. Attestation statement .................... 50
  7.3. Yubikey ...................................... 54
    7.3.1. Yubikey 4 ................................ 54
    7.3.2. Yubikey 5 ................................ 58
8. Privacy Considerations. .................................. 62
9. Security Considerations .................................. 62
10. IANA Considerations ................................... 62
11. Acknowledgements ..................................... 62
12. References .......................................... 62
    12.1. Normative References ........................ 62
    12.2. Informative References ...................... 62
Appendix A. Changes ....................................... 65
Authors’ Addresses ........................................ 65

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

[SP800-155] offers the following definition for root of trust.

"Roots of Trust are components (software, hardware, or hybrid) and computing engines that constitute a set of unconditionally trusted functions. Reliable and trustworthy BIOS integrity measurement and reporting depend upon software agents; each software agent relies upon Roots of Trust, and the level of trustworthiness in each agent depends on its Roots of Trust. BIOS integrity measurement requires the coordination of a Measurement Agent to harvest measurements, a Storage Agent to protect the measurements from modification until they can be reported, and a Reporting Agent to reliably report the measurements. Each of these agents has a corresponding Root of Trust (Root of Trust for Measurement, etc.) These Roots of Trust must act in concert and build on each other to enable reliable and trustworthy measurement, reporting, and verification of BIOS integrity measurements."

SP800-155 uses the terms RoT for Reporting, Storage and Measurement, but not RoT for Verification - it uses "Verification Agent". Though it is assumed the verifier is trustworthy.

However, [tcgglossary] (page 9) includes a RoT for Verification (RTV) as well.

The TCG Glossary also offers a general definition for Root of Trust: "A component that performs one or more security-specific functions, such as measurement, storage, reporting, verification, and/or update."
It is trusted always to behave in the expected manner, because its misbehavior cannot be detected (such as by measurement) under normal operation. "

[SP800-147B] defines RoT for Update (RoTU) and RoTU verification (RoTU-v).

The TCG definition seems more concise than the NIST, but gets to the same point.

For the purpose of this document, a hardware root of trust refers to security functionality that is trusted to behave in the expected manner, because its misbehavior cannot be detected under normal operation and resists soft exploits by encapsulating the functionality in hardware.

2.5. Template for Use cases

Each use case will consist of a table with a number of constant fields, as illustrated below. The claim names will be loosely synchronized with the EAT draft. The architecture draft (will) describe two classes of attestation flow: the passport type (Attestee sends evidence to Attester, receives signed statement, which is sent to relying party), or the background check type (Attestee sends measurements to Relying party, Relying Party checks with Attester).

Use case name: Twelve Monkeys
Who will use it: Army of the Twelve Monkeys SDO
Attesting Party: James Cole
Relying Party: Dr. Kathryn Reilly
Attestation type: Passport
Claims used: OEM Identity, Age Claim, Location Claim, ptime Claim
Description: James Cole must convince Dr. Reilly he is from the future, and not insane.

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 covered in this document:

- The Trusted Computing Group "Network Device Attestation Workflow" [I-D.fedorkow-rats-network-device-attestation]
- 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]
- Windows Defender System Guard attestation [windowsdefender]
- Windows Device Health Attestation [windowshealth]
- IETF NEA WG [RFC5209]

And any additional sources suggested.

5. Use case summaries

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

5.1. Device Capabilities/Firmware Attestation

This is a category of claims

Use case name: Device Identity

Who will use it: Network Operators

Attesting Party: varies

Attestation type: varies
Relying Party: varies
Claims used: TBD

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

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

5.1.1. Relying on an (third-party) Attestation Server

Use case name: Third Party Attestation Server
Who will use it: Network Operators
Attestation type: background check
Attesting Party: manufacturer of OS or hardware system
Relying Party: network access control systems
Claims used: TBD

Description: 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

Use case name: Autonomous
Who will use it: network operators
Attestation type: passport
Attesting Party: manufacturer of OS or hardware system
Relying Party: peer systems
Claims used: TBD

Description: The signed measurements are sent to a relying party which must validate them directly. They are not sent to a third party. (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 can not be connected until the equipment is validated.

5.1.3. Proxy Root of Trust

Use case name: Proxy Root of Trust

Who will use it: network operators

Attestation type: passport

Attesting Party: manufacturer of OS or hardware system

Relying Party: peer systems

Claims used: TBD

Description: A variety of devices provide measurements via their Root of Trust. A proxy 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

Use case name: Network scaled - small

Who will use it: enterprises

Attestation type: background check

Attesting Party: manufacturer of OS or hardware system

Relying Party: network equipment

Claims used: TBD

Description: 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

Use case name: Network scaled - medium

Who will use it: larger enterprises, including network operators

Attestation type: passport

Attesting Party: manufacturer of OS or hardware system

Relying Party: network equipment

Claims used: TBD

Description: 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

Use case name: Network scaled - medium

Who will use it: telco/LTE operators

Attestation type: passport

Attesting Party: manufacturer of OS or hardware system

Relying Party: malware auditing systems

Claims used: TBD

Description: 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 to maintain identities of connected devices more than to validate correct firmware, but both situations are reasonable.
5.2.  Hardware resiliency / watchdogs

Use case name:  Hardware watchdog
Who will use it:  individual system designers
Attestation type:  passport
Attesting Party:  manufacturer of OS or hardware system
Relying Party:  bootloader or service processor
Claims used:  TBD

Description:  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

Use case name:  TAM validation
Who will use it:  The TAM server
Attestation type:  background check
Attesting Party:  Trusted Execution Environment (TEE)
Relying Party:  end-application
Claims used:  TBD

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

5.4. Confidential Machine Learning (ML) model

Use case name: Machine Learning protection
Who will use it: Machine Learning systems
Attestation type: TBD
Attesting Party: hardware TEE
Relying Party: machine learning model owner
Claims used: TBD
Description: 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

Use case name: Critical Infrastructure
Who will use it: devices
Attestation type: TBD
Attesting Party: plant controller
Relying Party: actuator
Claims used: TBD
Description: 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

Use case name: Shared Block Chain Computational claims

Who will use it: Consortia of Computation systems

Attestation type: TBD

Attesting Party: computer system (physical or virtual)

Relying Party: other computer systems

Claims used: TBD

Description: 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. Virtualized multi-tenant hosts

Use case name: Multi-tenant hosts
Who will use it: Virtual machine systems
Attestation type: TBD
Attesting Party: virtual machine hypervisor
Relying Party: network operators
Claims used: TBD
Description: The host system will do verification as per 5.1. The tenant virtual machines will do verification as per 5.1
The network operator wants to know if the system _as a whole_ is free of malware, but the network operator is not allowed to know who the tenants are.
This is contrasted to the Chassis + Line Cards case (To Be Defined: TBD).
Multiple Line Cards, but a small attestation system on the main card can combine things together. This is a kind of proxy.

5.7. Cryptographic Key Attestation

Use case name: Key Attestation
Who will use it: network authentication systems
Attestation type: TBD
Attesting Party: device platform
Relying Party: internet peers
Claims used: TBD
Description: The relying party wants to know how secure a private key that identifies an entity is. Unlike the network attestation, the relying party is not part of the network infrastructure, nor do they necessarily have a business relationship (such as ownership) over the end device.
5.7.1. Device Type Attestation

Use case name: Device Type Attestation
Who will use it: mobile platforms
Attestation type: TBD
Attesting Party: device platform
Relying Party: internet peers
Claims used: TBD

Description: 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 trustworthy features such as a hardware TPM, software TPM via TEE, or software TPM without TEE. Other details such as the availability of finger-print readers or HDMI outputs may also be inferred.

5.7.2. Key storage attestation

Use case name: Key storage Attestation
Who will use it: secure key storage subsystems
Attestation type: TBD
Attesting Party: device platform
Relying Party: internet peers
Claims used: TBD

Description: 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.
(e.g., EST {#rfc7030}, CMP {#rfc4210}, ACME {#rfc8555}, SCEP [I-D.gutmann-scep], etc.) and processed by registration authorities or certification authorities prior to determining contents for any issued certificate.

5.7.3. End user authorization

Use case name: End User authorization

Who will use it: authorization systems

Attestation type: TBD

Attesting Party: device platform

Relying Party: internet peers

Claims used: TBD

Description: 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)

5.8. Geographic attestation

Use case name: Location attestation

Who will use it: geo-fenced systems

Attestation type: passport (probably)

Attesting Party: secure GPS system(s)

Relying Party: internet peers

Claims used: TBD

Description: 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.

5.8.1. I am here

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

5.8.2. 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.

5.8.3. 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.

5.9. Connectivity attestation

Use case name: Connectivity attestation

Who will use it: entertainment systems

Attestation type: TBD
Attesting Party: hardware-manufacturer/TEE
Relying Party: connected peer
Claims used: TBD
Description: 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.

5.10. Component connectivity attestation

Use case name: Component connectivity
Who will use it: chassis systems with pluggable components
Attestation type: background check
Attesting Party: line card
Relying Party: management/control plane software
Claims used: TBD
Description: A management controller or similar hardware component wants to know what peripherals, rack scale device or other dynamically configurable components are currently attached to the platform that is under management controller control. The management controller may serve as attestation verifier over a local bus or backplane but may also aggregate local attestation results and act as a platform attester to a remote verifier.

5.11. Device provenance attestation

Use case name: RIV - Device Provenance
Who will use it: Industrial IoT devices
Attestation type: passport
Attesting Party: network management station
Relying Party: a network entity
Claims used: TBD
Description: A newly manufactured device needs to be onboarded into a network where many if not all device management duties are performed by the network owner. The device owner wants to verify the device originated from a legitimate vendor. A cryptographic device identity such as an IEEE802.1AR is embedded during manufacturing and a certificate identifying the device is delivered to the owner onboarding agent. The device authenticates using its 802.1AR IDevID to prove it originated from the expected vendor.

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

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

6. Technology users for RATS

6.1. Trusted Computing Group Remove Integrity Verification (TCG-RIV)

The TCG RIV Reference Document addresses 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 available as [I-D.fedorkow-rats-network-device-attestation]. 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 ROM)

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
report the inventory of software was launched on the box in a way that cannot be spoofed, that is undetectably altered by a "Lying Endpoint."

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, composite assemblies of hardware/software created by end-customers, and equipment that uses Sleep or Hibernate modes. There is an intention to cover some of these are topics in future versions of the documents.

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

A critical component of the RIV is compatibility with existing TPM practice for attestation procedures, as spelled out in the TCG TAP Informational Model [tapinfomodel] and TPM architecture specifications [tpmarchspec].

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 my device manufacturer and integrated into firmware.
- Quotes: measured values (having been signed), and RIMs
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 5.7 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 5.7 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.
KeyGenParameterSpec.Builder builder = null;
if(hasStrongBox) {
    builder = new KeyGenParameterSpec.Builder(
            m_alias,
            KeyProperties.PURPOSE_SIGN | KeyProperties.PURPOSE_VERIFY
Y | KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
            .setKeySize(2048)
            .setDigests(KeyProperties.DIGEST_NONE, KeyProperties.DIG
EST_SHA256)
            .setBlockModes(KeyProperties.BLOCK_MODE_CBC, KeyProperti
es.BLOCK_MODE_GCM)
            .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_R
SA_PKCS1, KeyProperties.ENCRYPTION_PADDING_RSA_OAEP)
            .setSignaturePaddings(KeyProperties.SIGNATURE_PADDING_RS
A_PSS, KeyProperties.SIGNATURE_PADDING_RSA_PKCS1)
            .setUserAuthenticationRequired(false)
            .setIsStrongBoxBacked(true)
            .setUnlockedDeviceRequired(true);
} else {
    builder = new KeyGenParameterSpec.Builder(
            m_alias,
            KeyProperties.PURPOSE_SIGN | KeyProperties.PURPOSE_VERIFY
Y | KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
            .setKeySize(2048)
            .setDigests(KeyProperties.DIGEST_NONE, KeyProperties.DIG
EST_SHA256, KeyProperties.DIGEST_SHA384, KeyProperties.DIGEST_SHA512)
            .setBlockModes(KeyProperties.BLOCK_MODE_CBC, KeyProperti
es.BLOCK_MODE_CTR, KeyProperties.BLOCK_MODE_GCM)
            .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_R
SA_PKCS1, KeyProperties.ENCRYPTION_PADDING_RSA_OAEP)
            .setSignaturePaddings(KeyProperties.SIGNATURE_PADDING_RS
A_PSS, KeyProperties.SIGNATURE_PADDING_RSA_PKCS1)
            .setUserAuthenticationRequired(false)
            .setIsStrongBoxBacked(false)
            .setUnlockedDeviceRequired(true);
}
builder.setAttestationChallenge(challenge_bytes);

KeyGenParameterSpec keySpec = builder.build();
AttestedKeyPair apk = dpm.generateKeyPair(componentName, algorithm, keySpec, idA
ttestationFlags);

7.1.1.  TEE

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

0 1172: SEQUENCE {
4 764:   SEQUENCE {
8  3:       [0] {
10  1:           INTEGER 2
   :   }
13  1:           INTEGER 1
16 13:   SEQUENCE {
18  9:       OBJECT IDENTIFIER
          : sha256WithRSAEncryption (1 2 840 113549 1 1 11)
29  0:         NULL
  :       )
31  27:       SEQUENCE {
33  25:       SET {
35  23:         SEQUENCE {
37    3:           OBJECT IDENTIFIER serialNumber (2 5 4 5)
42    16:           PrintableString 'c6047571d8f0d17c'
        :         }
        :       }
60  32:       SEQUENCE {
62  13:         UTCTime 01/01/1970 00:00:00 GMT
77  15:         GeneralizedTime 07/02/2106 06:28:15 GMT
      :       }
94  31:       SEQUENCE {
96  29:       SET {
98  27:         SEQUENCE {
100    3:           OBJECT IDENTIFIER commonName (2 5 4 3)
105   20:           UTF8String 'Android Keystore Key'
      :       }
      :       }
127 290:       SEQUENCE {
131 13:       SEQUENCE {
133  9:           OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
144  0:           NULL
      :       }
146 271:       BIT STRING, encapsulates {
151 266:       SEQUENCE {
155 257:         INTEGER
      :         00 B5 3A 83 61 A2 85 CC D2 D6 25 7F 07 0B B4 A0
      :         F6 FE 05 01 C9 55 CB 0D 18 D2 C6 79 BA 82 12 67
      :         75 8D 5B F3 24 D3 F8 EA 99 82 7D 1F 5E CD 77 D6
      :         99 11 13 FF 18 C9 3D 4D 01 C5 8E E9 04 E7 17 E2
      :         88 12 2B B9 A1 77 2F C2 4F 57 78 98 4E E3 DE 7A
      :         1B 18 BE D3 ED C9 59 A0 24 50 E1 FA AC 81 B6 DA
      :         80 B0 BD 48 AD 26 9C 4A 4E CE 54 17 58 C1 F4 F8
      :         7F 3C 5D 8F C8 2C 2A 7B 18 95 B3 D4 E0 3A C8 9D
      :         [ Another 129 bytes skipped ]
416  3:         INTEGER 65537
      :         }
      :       }
421 347:       [3] {
425 343:       SEQUENCE {
429 14:       SEQUENCE {
431  3:         OBJECT IDENTIFIER keyUsage (2 5 29 15)
436  1:         BOOLEAN TRUE

Richardson, et al.    Expires April 8, 2020
439 4:           OCTET STRING, encapsulates {
441 2:             BIT STRING 4 unused bits
442 :               '1100'B
443 :             }
445 323:           SEQUENCE {
449 10:             OBJECT IDENTIFIER '1 3 6 1 4 1 11129 2 1 17'
461 307:             OCTET STRING, encapsulates { -- Attestation Extension
465 303:               SEQUENCE {
469 1:                 INTEGER 2 -- attestationVersion (KM3)
472 1:                 ENUMERATED 1 -- attestationSecurityLevel (Trusted
474 358:                 Env.)
475 1:                 INTEGER 3 -- keymasterVersion
478 1:                 ENUMERATED 1 -- keymasterSecurityLevel (TrustedEn
480 358:                 v.)
481 9:                 OCTET STRING 'challenge' -- attestationChallenge
492 0:                 OCTET STRING -- reserved
494 44:               SEQUENCE { -- softwareEnforced
496 8:                 [701] {
498 6:                   INTEGER 01 64 47 2A 4B 64
500 6:                 }
508 28:                 [709] {
512 26:                   OCTET STRING, encapsulates {
514 24:                     SEQUENCE { -- AttestationApplicationId
516 20:                       SET {
518 18:                         SEQUENCE { -- AttestationPackageInfo
520 13:                           OCTET STRING 'AndroidSystem' -- package_na
523 358:                           me
525 1:                             INTEGER 1 -- version
527 1:                             }
535 1:                             }
538 0:                           SET {} -- signature_digests
540 229:               SEQUENCE { -- hardwareEnforced
543 14:                 [1] {
545 12:                   SET {
547 1:                     INTEGER 0 -- KeyPurpose.ENCRYPT
550 1:                     INTEGER 1 -- KeyPurpose.DECRYPT
553 1:                     INTEGER 2 -- KeyPurpose.SIGN
556 1:                     INTEGER 3 -- KeyPurposeVERIFY
559 3:                     [2] {
561 1:                       INTEGER 1 -- Algorithm.RSA
564 4:                     [3] {
566 2:                       INTEGER 2048

570 11: [5] { -- digest
572  9: SET {
574  1: INTEGER 4   -- Digest.SHA256
577  1: INTEGER 5   -- Digest.SHA384
580  1: INTEGER 6   -- Digest.SHA512
  :  : }
583 14: [6] { -- padding
585 12: SET {
587  1: INTEGER 4   -- PaddingMode.RSA_PKCS1_1_5_ENCRYPT
590  1: INTEGER 2   -- PaddingMode.RSA_OAEP
593  1: INTEGER 3   -- PaddingMode.RSA_PKCS1_1_5_SIGN
596  1: INTEGER 5   -- PaddingMode.RSA_PSS
  :  : }
599  5: [200] { -- rsaPublicExponent
603  3: INTEGER 65537
  :  }
608  2: [503] { -- noAuthRequired
612  0: NULL           -- documentation indicates this is a Boolean
  :  }
614  3: [702] { -- origin
618  1: INTEGER 0   -- KeyOrigin.GENERATED
  :  }
621  2: [703] { -- rollbackResistant
625  0: NULL           -- documentation indicates this is a Boolean
  :  }
627 42: [704] { -- rootOfTrust
631 40: SEQUENCE { -- verifiedBootKey
633 32: OCTET STRING :
       19 62 B0 53 85 79 FF CE 9A C9 F5 07 C4 6A FE 3B
       92 05 5B AC 71 46 46 22 83 C8 5C 50 0B E7 8D 82
667  1: BOOLEAN TRUE   -- deviceLocked
670  1: ENUMERATED 0   -- verifiedBootState (verified)
  :  : }
673  5: [705] { -- osVersion
677  3: INTEGER 90000   -- Android P
  :  : }
682  5: [706] { -- osPatchLevel
686  3: INTEGER 201806   -- June 2018
  :  : }
691  8: [710] { -- attestationIdBrand
695  6: OCTET STRING 'google'
  :  : }
703  9: [711] { -- attestationIdDevice
707  7: OCTET STRING 'walleye'
716  9:    [712] {                     -- attestationIdProduct
720  7:      OCTET STRING 'walleye'
    :                     }
729 14:    [713] {                     -- attestationIdSerial
733 12:      OCTET STRING 'HT83K1A03849'
    :                     }
747  8:    [716] {                     -- attestationIdManufacturer
751  6:      OCTET STRING 'Google'
    :                     }
759  9:    [717] {                     -- attestationIdModel
763  7:      OCTET STRING 'Pixel 2'
    :                     }
772 13:  SEQUENCE {
774  9:    OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1 11)
785  0:    NULL
    :                     }
787 385:  BIT STRING
    :  05 41 B9 13 11 53 93 A2 02 62 1F 15 35 8E D9 7C
    :  A1 D5 2E ED 13 AC 24 26 B2 A1 2F EE B4 0C 4D 71
    :  DC 9F 55 EC A1 F6 64 62 F2 73 A8 7E FC 48 63 29
    :  1E F5 0D 48 F3 73 43 0C 00 E0 D4 07 B6 A6 A4 38
    :  0E A8 47 0F 27 01 01 31 52 F6 62 8A 4B 80 BE 72
    :  FB 02 E7 56 84 CA CA 4D C3 6C 7C B2 BA C7 D7 9B
    :  C5 9D 90 65 4E F5 54 8F 25 CC 11 7F BE 77 10 6A
    :  6E 9F 80 89 48 8B 1D 51 AA 3B B7 C5 24 3C 28 B1
    :  [ Another 256 bytes skipped ]
    :                     }
0 1304:  SEQUENCE {
4  768:    SEQUENCE {
8  3:      [0] {
10  1:        INTEGER 2
    :                     }
13 10:    INTEGER 10 34 53 32 94 08 68 79 38 72
25 13:    SEQUENCE {
27  9:      OBJECT IDENTIFIER
    :      sha256WithRSAEncryption (1 2 840 113549 1 1 11)
38  0:      NULL
    :                     }
40  27:    SEQUENCE {
42  25:      SET {

Richardson, et al.          Expires April 8, 2020
44  23:    SEQUENCE {  
46   3:      OBJECT IDENTIFIER serialNumber (2 5 4 5)  
51  16:      PrintableString '87f4514475ba0a2b'  
 :  
 :  
 :  
 :  
69  30:    SEQUENCE {  
71  13:      UTCTime 26/05/2016 17:14:51 GMT  
86  13:      UTCTime 24/05/2026 17:14:51 GMT  
 :  
101  27:    SEQUENCE {  
103  25:      SET {  
107  23:        SEQUENCE {  
112  16:          PrintableString 'c6047571d8f0d17c'  
 :  
 :  
 :  
 :  
130  418:    SEQUENCE {  
134  13:      SEQUENCE {  
136   9:        OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)  
147   0:        NULL  
 :  
149  399:      BIT STRING, encapsulates {  
154  394:        SEQUENCE {  
158  385:          INTEGER : 00 B3 01 0D 78 BC 06 33 25 CA D6 A7 2C EF 49 05  
 : 4C C1 77 36 F2 E5 7B E8 4C 0A 87 8F 77 6A 09 45  
 : 9B AC E8 72 DA E2 0E 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 9B FC 1F 4F 77 2D 92 E9 DE 5F 6B 02 09 4E  
 : 99 86 53 98 1C 5E 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 5D 6A 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 0C 23 14  
 : 72 8F 1D 80  
 :  
Richardson, et al. Expires April 8, 2020 [Page 28]
589 31:  SEQUENCE {
591  3:    OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29 35)
596  24:    OCTET STRING, encapsulates {
598  22:      SEQUENCE {
600  20:        [0]
602   :          0E 55 6F 46 F5 3B 77 67 E1 B9 73 DC 55 E6 AE EA
604   :            B4 FD 27 DD
606   :
608   :
610   :
612   :
614   :
616   :
618   :
620   :
622  12:    SEQUENCE {
624   3:      OBJECT IDENTIFIER basicConstraints (2 5 29 19)
629   1:      BOOLEAN TRUE
632   2:      OCTET STRING, encapsulates {
634   0:        SEQUENCE {}
636   :
638   14:    SEQUENCE {
640   3:      OBJECT IDENTIFIER keyUsage (2 5 29 15)
643   1:      BOOLEAN TRUE
646   4:      OCTET STRING, encapsulates {
648   2:        BIT STRING 7 unused bits
650   :          '1'B (bit 0)
652   :
654  36:    SEQUENCE {
656  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'
669   :
671   :
673   :
675   :
677   :
679  84:    SEQUENCE {
681  3:      OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
684  77:      OCTET STRING, encapsulates {
686  75:      SEQUENCE {
688  73:        SEQUENCE {
690  71:          [0] {
692  69:            [0] {
694  67:              'https://android.googleapis.com/attestation/crl/1'
696   :                '0345332940868793872'
776 13:  SEQUENCE {
778  9:    OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1 11)
780  0:    NULL
782  1393:  SEQUENCE {
784  4  857:    SEQUENCE {
787  8  3:      [0] {
789  10  1:        INTEGER 2
791  13:      }
794  25  13:      INTEGER 03 88 26 67 60 65 89 96 85 74
798  27  9:      SEQUENCE {
800  38  0:        OBJECT IDENTIFIER
802  40  13:        sha256WithRSAEncryption (1 2 840 113549 1 1 11)
805  50  0:        NULL
807  40  27:      SEQUENCE {
809  42  25:        SET {
812  46  3:          OBJECT IDENTIFIER serialNumber (2 5 4 5)
815  51  16:          PrintableString 'f92009e853b6b045'
818  69  30:      SEQUENCE {
820  71  13:        UTCTime 26/05/2016 17:01:32 GMT
823  86  13:        UTCTime 24/05/2026 17:01:32 GMT
826  101  27:      SEQUENCE {
828  101  27:      }

103  25:       SET {
105  23:         SEQUENCE {
107   3:           OBJECT IDENTIFIER serialNumber (2 5 4 5)
112  16:           PrintableString '87f4514475ba0a2b'
130  546:       SEQUENCE {
134   9:           OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
147  0:           NULL
149  527:       BIT STRING, encapsulates {
154  522:         SEQUENCE {
158  513:           INTEGER 65537
168  182:         [3] {
173  179:           SEQUENCE {
178  29:             SEQUENCE {
188   3:             OCTET STRING, encapsulates {
198  20:               OCTET STRING
208   20:                 0E 55 6F 46 F5 3B 77 67 E1 B9 73 DC 55 E6 AE EA
218   20:                 B4 FD 27 DD
228   20:               [0]
233   20:                 36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
243   20:                 C9 EA 4F 12
253   20:               [0]
263   20:             SEQUENCE {
273  31:               SEQUENCE {
283  35:               OCTET STRING, encapsulates {
293  32:               SEQUENCE {
303  20:                 36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
313   20:                 C9 EA 4F 12
323   20:               [0]
SEQUENCE { 
  OBJECT IDENTIFIER basicConstraints (2 5 29 19) 
  BOOLEAN TRUE 
  OCTET STRING, encapsulates { 
    SEQUENCE { 
      BOOLEAN TRUE 
    } 
  } 
} 

SEQUENCE { 
  OBJECT IDENTIFIER keyUsage (2 5 29 15) 
  BOOLEAN TRUE 
  OCTET STRING, encapsulates { 
    BIT STRING 1 unused bit 
    '1100001'B 
  } 
} 

SEQUENCE { 
  OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31) 
  OCTET STRING, encapsulates { 
    SEQUENCE { 
      [0] { 
        [0] { 
          [6] 
          'https://android.googleapis.com/attestation/crl/E' 
          '8FA196314D2FA18' 
        } 
      } 
    } 
  } 
} 

SEQUENCE { 
  OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1 11) 
  NULL 
} 

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
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 82 2B B1 A7 01 EC 2B B4 2E 8B CC 54
   :     16 63 AB EF 90 2F 32 C7 7F 75 31 03 0C 97 52 4B
   :     1B 5F E9 09 FB C7 2A A9 45 1F 74 3C BD 9A 6E 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
   :  }
Richardson, et al. Expires April 8, 2020 [Page 33]
:     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
:     }
:     }
679 166:     [3] {
682 163:     SEQUENCE {
685  29:         SEQUENCE {
687  3:             OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
692  22:             OCTET STRING, encapsulates {
694  20:                 OCTET STRING
696  18:                 36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
698  12:                 C9 EA 4F 12
716  31:         SEQUENCE {
718  3:             OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29 35)
723  24:             OCTET STRING, encapsulates {
725  22:                 SEQUENCE {
727  20:                     [0]
729  18:                     36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
731  12:                     C9 EA 4F 12
749  15:         SEQUENCE {
751  3:             OBJECT IDENTIFIER basicConstraints (2 5 29 19)
756  1:             BOOLEAN TRUE
759  5:             OCTET STRING, encapsulates {
761  3:                 SEQUENCE {
763  1:                     BOOLEAN TRUE
766  14:         SEQUENCE {
768  3:             OBJECT IDENTIFIER keyUsage (2 5 29 15)
773  1:             BOOLEAN TRUE
776  4:             OCTET STRING, encapsulates {
778  2:                 BIT STRING 1 unused bit
781  3:                     '1100001'B
792  64:         SEQUENCE {
784  3:             OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
789  57:             OCTET STRING, encapsulates {
791  55:             SEQUENCE {
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.

0 5143: SEQUENCE {
4 9:  OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)
15 5128: [0] {
19 5124:  SEQUENCE {
23 1:   INTEGER 1
26 0:   SET {
28 11:    SEQUENCE {
30 9:     OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
    }
41 5100: [0] {
45 1114:   SEQUENCE {
49 834:    SEQUENCE {

Richardson, et al. Expires April 8, 2020
53  3:          [0] {  
55  1:            INTEGER 2  
    :  
58  1:            INTEGER 1  
61  13:          SEQUENCE {  
63  9:            OBJECT IDENTIFIER  
    :            sha256WithRSAEncryption (1 2 840 113549 1 1 11)  
74  0:            NULL  
    :  
76  47:          SEQUENCE {  
78  25:          SET {  
80  23:          SEQUENCE {  
82  3:            OBJECT IDENTIFIER serialNumber (2 5 4 5)  
87  16:            PrintableString ‘90e8da3cadfc7820’  
    :  
    :  
105  18:          SET {  
107  16:          SEQUENCE {  
109  3:            OBJECT IDENTIFIER title (2 5 4 12)  
114  9:            UTF8String ‘StrongBox’  
    :  
    :  
125  30:          SEQUENCE {  
127  13:          UTCTime 01/01/1970 00:00:00 GMT  
142  13:          UTCTime 23/05/2028 23:59:59 GMT  
    :  
157  31:          SEQUENCE {  
159  29:          SET {  
161  27:          SEQUENCE {  
163  3:            OBJECT IDENTIFIER commonName (2 5 4 3)  
168  20:            UTF8String ‘Android Keystore Key’  
    :  
    :  
190  290:          SEQUENCE {  
194  13:          SEQUENCE {  
196  9:            OBJECT IDENTIFIER  
    :            rsaEncryption (1 2 840 113549 1 1 1)  
207  0:            NULL  
    :  
209  271:          BIT STRING, encapsulates {  
214  266:          SEQUENCE {  
218  257:            INTEGER  
    :            00 DE 98 94 D5 E5 05 98 E8 FC 73 4D 26 FB 48 6A  
    :            CA 06 A0 24 FA 05 D1 D2 32 10 46 F8 50 DD 3E 0D  
    :            DF 4F 95 53 D2 CB 10 1F 00 B2 62 15 1E 21 7E 05  
    :            C6 10 AC EE 7A D8 69 F1 1F 32 C3 17 CA D7 07 BE  

Richardson, et al. Expires April 8, 2020
479 3:  INTEGER 65537

484 399:  [3] {  
488 395:   SEQUENCE {  
492 14:     SEQUENCE {  
494 3:       OBJECT IDENTIFIER keyUsage (2 5 29 15)  
499 1:         BOOLEAN TRUE  
502 4:         OCTET STRING, encapsulates {  
504 2:          BIT STRING 7 unused bits
506 0:          '1'B (bit 0)  
508 375:       SEQUENCE {  
512 10:         OBJECT IDENTIFIER ’1 3 6 1 4 1 11129 2 1 17’  
516 359:         OCTET STRING, encapsulates {  
520 355:           SEQUENCE {  
524 1:             INTEGER 3  
527 1:             ENUMERATED 2 -- attestationSecurityLevel (StrongBox)  
530 1:             INTEGER 4  
533 1:             ENUMERATED 2 -- attestationSecurityLevel (StrongBox)  
536 9:             OCTET STRING ’challenge’  
540 0:             OCTET STRING  
544 53:               SEQUENCE {  
548 2:                 [509] {  
552 0:                   NULL  
554 11:                     [701] {  
558 9:                       INTEGER 00 FF FF FF FF E5 99 78  
562 0:                        [709] {  
566 28:                          OCTET STRING, encapsulates {  
570 24:                            SEQUENCE {  
574 20:                              SET {  
578 18:                                OCTET STRING ’AndroidSystem’  
582 1:                                  INTEGER 1  
585 0:                                    SET {}  
588 0:                                      }  
592 18:                                    }  
595 0:                                   }  
599 0:                                  }  
603 0:                                 }  
607 0:                               }  
611 0:                             }  

Richardson, et al. Expires April 8, 2020 [Page 37]
612  271:       SEQUENCE {
616   14:          [1] {
618   12:             SET {
620    1:               INTEGER 0
623    1:               INTEGER 1
626    1:               INTEGER 2
629    1:               INTEGER 3

632    3:
634    1:
637    4:
639    2:               INTEGER 2048

643    8:
645    6:             SET {
647    1:               INTEGER 2
650    1:               INTEGER 32

653    8:
655    6:
657    1:
660    1:

663   14:
665   12:             SET {
667    1:               INTEGER 2
670    1:               INTEGER 3
673    1:               INTEGER 4
676    1:               INTEGER 5

679    2:
683    0:
685    3:
689    1:

692    76:
696    74:          SEQUENCE {
698    32:             OCTET STRING
61 FD A1 2B 32 ED 84 21 4A 9C F1 3D 1A FF B7 AA

Richardson, et al. Expires April 8, 2020 [Page 38]
80 BD 8A 26 8A 86 1E D4 BB 7A 15 17 0F 1A B0 0C

732 1: BOOLEAN TRUE
735 1: ENUMERATED 0
738 32: OCTET STRING

772 3: [705] {
776 1: INTEGER 0
    }
779 5: [706] {
783 3: INTEGER 201903
    }
788 8: [710] {
792 6: OCTET STRING 'google'
    }
800 10: [711] {
804 8: OCTET STRING 'blueline'
    }
814 10: [712] {
818 8: OCTET STRING 'blueline'
    }
828 11: [713] {
832 9: OCTET STRING '8A2X0KLUU'
    }
843 8: [716] {
847 6: OCTET STRING 'Google'
    }
855 9: [717] {
859 7: OCTET STRING 'Pixel 3'
    }
868 6: [718] {
872 4: INTEGER 20180905
    }
878 5: [719] {
882 3: INTEGER 201903
    }
887 13: SEQUENCE {
889 9: OBJECT IDENTIFIER
     : sha256WithRSAEncryption (1 2 840 113549 1 1 11)
83  EA 59  8D  BE 37  4A  D5  C0  FC  F8  FB  AC  8B  72  1E
      :    A5  C2  3B  0C  C0  04  1B  C0  5A  18  A5  DF  D4  67  1D  B9
      :    08  42  4B  E2  2C  AC  07  0F  D8  0E  24  97  56  9E  14  F2
      :    D0  AC  DD  1E  FC  DD  68  20  11  DF  88  B8  B6  22  AD  2B
      :    DB  9C  2E  5C  3F  AF  0B  8F  02  68  AA  34  4B  5E  C8  75
      :    B1  1A  09  D2  19  41  24  51  18  A5  DF  D4  67  1D  B9
      :    08  42  4B  E2  2C  AC  07  0F  D8  0E  24  97  56  9E  14  F2
      :    B1  1A  09  D2  19  41  24  51  18  A5  DF  D4  67  1D  B9
      :    D8  18  C7  25  F3  3F  C0  6A  37  AB  49  B6  96  51  61  72
      : [ Another 128 bytes skipped ]

1163 1181:     SEQUENCE {
1167  645:       SEQUENCE {
1171    3:         [0] {
1173    1:           INTEGER 2
      :
1176   10:           INTEGER 17 10 24 68 40 71 02 97 78 50
1188   13:       SEQUENCE {
1190    9:         OBJECT IDENTIFIER
      :           sha256WithRSAEncryption (1 2 840 113549 1 1 11)
1201   0:         NULL
      :
1203   47:       SEQUENCE {
1205   25:         SET {
1207   23:           SEQUENCE {
1209    3:             OBJECT IDENTIFIER serialNumber (2 5 4 5)
1214   16:             PrintableString ‘ccd18b9b608d658e’
      :
      :
1232   18:         SET {
1234   16:           SEQUENCE {
1236    3:             OBJECT IDENTIFIER title (2 5 4 12)
1241    9:             UTF8String ‘StrongBox’
      :
      :
1252   30:         SEQUENCE {
1254   13:           UTCTime 25/05/2018 23:28:47 GMT
1269   13:           UTCTime 22/05/2028 23:28:47 GMT
      :
      :
1284   47:         SEQUENCE {
1286   25:         SET {
1288   23:           SEQUENCE {
1290    3:             OBJECT IDENTIFIER serialNumber (2 5 4 5)
1295   16:             PrintableString ‘90e8da3cadfc7820’
      :
      :}
Internet-Draft           useful RATS                  October 2019

: )
1313  18:     SET {
1315  16:       SEQUENCE {
1317    3:         OBJECT IDENTIFIER title (2 5 4 12)
1322    9:         UTF8String 'StrongBox'
    : )
    : )
1333  290:    SEQUENCE {
1337    13:     SEQUENCE {
1339    9:       OBJECT IDENTIFIER
    : rsaEncryption (1 2 840 113549 1 1 1)
1350    0:       NULL
    : )
1352  271:    BIT STRING, encapsulates {
1357  266:      SEQUENCE {
1361  257:        INTEGER 65537
    : )
    : )
1622    3:      INTEGER 65537
    : )
    : )
1627  186:     [3] {
1630  183:     SEQUENCE {
1633    29:      SEQUENCE {
1635    3:        OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
1640    22:      OCTET STRING, encapsulates {
1642    20:        OCTET STRING
    : 77 A4 AD DF 1D 29 89 CA 92 E3 BA DE 27 3C 70 DF
    : 36 03 7C 0C
    : )
1664    31:      SEQUENCE {
1666    3:        OBJECT IDENTIFIER
    : authorityKeyIdentifier (2 5 29 35)
1671    24:      OCTET STRING, encapsulates {
1673    22:        SEQUENCE {
1675    20:          [0]
    : 1B 17 70 C6 97 DC 84 54 75 7C 3C 98 5C E6 1D 1D
    : 08 59 5D 53

Richardson, et al. Expires April 8, 2020 [Page 41]
::
::
::
1697 15:
SEQUENCE {
1699 3:
  OBJECT IDENTIFIER basicConstraints (2 5 29 19)
1704 1:
  BOOLEAN TRUE
1707 5:
  OCTET STRING, encapsulates {
1709 3:
    SEQUENCE {
1711 1:
      BOOLEAN TRUE
    }
  }
1714 14:
SEQUENCE {
1716 3:
  OBJECT IDENTIFIER keyUsage (2 5 29 15)
1721 1:
  BOOLEAN TRUE
1724 4:
  OCTET STRING, encapsulates {
1726 2:
    BIT STRING 2 unused bits
  
1730 84:
SEQUENCE {
1732 3:
  OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
1737 77:
  OCTET STRING, encapsulates {
1741 73:
    SEQUENCE {
1743 71:
      [0] {
1745 69:
        [0] {
1747 67:
          [6]
            'https://android.googleapis.com/attestation/crl/1'
            '7102468407102977850'
          }
        }
      }
    }
1816 13:
SEQUENCE {
1818 9:
  OBJECT IDENTIFIER
  sha256WithRSAEncryption (1 2 840 113549 1 1 11)
1829 0:
  NULL
1831 513:
BIT STRING
  13 22 DA F2 92 93 CE C0 9F 70 40 C9 DA 85 6B 61
  6F 8F BE E0 A4 04 55 C1 63 84 61 37 F5 4B 71 6D
  62 AA 6F BF 6C E8 48 03 AD 28 85 21 9E 3C 1C 91

Richardson, et al. Expires April 8, 2020 [Page 42]
: 48 EE 65 28 65 70 D0 BD 5B CC DB CE B1 F5 B5 C3
: CA 7A A9 C8 8A 68 12 8A CA 6A 85 A6 BC DA 36 E9
: B9 94 35 82 5B CA BC B6 9F 83 03 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 29 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
2377 7:         sha256WithRSAEncryption (1 2 840 113549 1 1 11)
2380 0:        NULL
2383 3:      }}
2387 27:     SEQUENCE {
2390 25:       SET {
2393 23:         SEQUENCE {
2396 3:           OBJECT IDENTIFIER serialNumber (2 5 4 5)
2399 16:           PrintableString ‘f92009e853b6b045’
2398 16:           }
2401 25:       }}
2416 30:     SEQUENCE {
2420 13:       UTCTime 26/05/2016 16:28:52 GMT
2423 13:       UTCTime 24/05/2026 16:28:52 GMT
2428 27:     }}
2448 27:   SEQUENCE {
2455 25:     SET {
2458 23:       SEQUENCE {
2461 3:         OBJECT IDENTIFIER serialNumber (2 5 4 5)
2464 16:         PrintableString ‘f92009e853b6b045’
2463 16:       }}
2477 546:  SEQUENCE {
2481 13:   SEQUENCE {
2483 9:    OBJECT IDENTIFIER
2486 7:      rsaEncryption (1 2 840 113549 1 1 1)
2490 0:    NULL
2493 3:   }}
2496 527:  BIT STRING, encapsulates {
2504 513:   SEQUENCE {
2505 513:     INTEGER
2508 16:     00 AF B6 C7 82 2B B1 A7 01 EC 2B 4E 2E 8B CC 54

Richardson, et al. Expires April 8, 2020 [Page 43]
3022  3:                      INTEGER 65537
          
3027 166:                   [3] {  
3030 163:                    SEQUENCE {  
3033  29:                       OCTET STRING, encapsulates {  
3035   3:                         OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)  
3040  22:                         OCTET STRING  
3042  20:                         OCTET STRING  
3044  18:                     [0]  
3046  16:                        OCTET STRING  
3048  14:                         OCTET STRING  
3050  12:                       OCTET STRING  
3052  10:                         OCTET STRING  
3054   8:                       OCTET STRING  
3056   6:                        OCTET STRING  
3058   4:                       OCTET STRING  
3060   2:                        OCTET STRING  
3062   0:                          OCTET STRING  
3064  31:                    SEQUENCE {  
3066   3:                        OBJECT IDENTIFIER  
3071  24:                         OCTET STRING, encapsulates {  
3073  22:                         OCTET STRING  
3075  20:                         OCTET STRING  
3077  18:                     [0]  
3079  16:                        OCTET STRING  
3081  14:                         OCTET STRING  
3083  12:                       OCTET STRING  
3085  10:                         OCTET STRING  
3087   8:                       OCTET STRING  
3089   6:                        OCTET STRING  
3091   4:                       OCTET STRING  
3093   2:                        OCTET STRING  
3095   0:                          OCTET STRING  
3097  15:                    SEQUENCE {  
3099  3:                        OBJECT IDENTIFIER basicConstraints (2 5 29 19)  
3104   1:                       BOOLEAN TRUE  
3107  5:                         OCTET STRING, encapsulates {  
3109  3:                         OCTET STRING  
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

Richardson, et al.          Expires April 8, 2020         [Page 44]
SEQUENCE {
  OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
  OCTET STRING, encapsulates {
    SEQUENCE {
      [0] {
        [0] {
          [6] :
            'https://android.googleapis.com/attestation/crl/

        
      } :
    } :
  } :
}

SEQUENCE {
  OBJECT IDENTIFIER
    sha256WithRSAEncryption (1 2 840 113549 1 1 11)
  NULL :
}

BIT STRING :
  20 C8 C3 8D 4B DC A9 57 1B 46 8C 89 2F 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 27 DF :
  8B 04 F1 82 77 0F 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 ]
  }

SEQUENCE {
  [0] {
    INTEGER 2 :
  } :
}

SEQUENCE {
  INTEGER 03 88 26 67 60 65 89 96 85 99 :
  OBJECT IDENTIFIER
    sha256WithRSAEncryption (1 2 840 113549 1 1 11)
  NULL :
  }

SEQUENCE {

SET {
  SEQUENCE {
    OBJECT IDENTIFIER serialNumber (2 5 4 5)
    PrintableString 'f92009e853b6b045'
  }
}

SET {
  SEQUENCE {
    OBJECT IDENTIFIER serialNumber (2 5 4 5)
    PrintableString 'ccd18b9b608d658e'
  }
}

SET {
  SEQUENCE {
    OBJECT IDENTIFIER title (2 5 4 12)
    UTF8String 'StrongBox'
  }
}

SEQUENCE {
  SEQUENCE {
    OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
    NULL
  }
  BIT STRING, encapsulates {
    INTEGER 65537
  }
}
SEQUENCE {
  OCTET STRING, encapsulates {
    1B 17 70 C6 97 DC 84 54 75 7C 3C 98 5C E6 1D 1D
    08 59 5D 53
  }
}

SEQUENCE {
  OCTET STRING, encapsulates {
    [0]
    36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
    C9 EA 4F 12
  }
}

SEQUENCE {
  OCTET STRING, encapsulates {
    BOOLEAN TRUE
    [0]
  }
}

SEQUENCE {
  OCTET STRING, encapsulates {
    BOOLEAN TRUE
  }
}

SEQUENCE {
  OCTET STRING, encapsulates {
    BOOLEAN TRUE
    [0]
  }
}

SEQUENCE {
  OCTET STRING, encapsulates {
    [6]
    'https://android.googleapis.com/attestation/crl/8'
    'F67349FA504789'
  }
}
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:
CertificateRequestProperties request = new CertificateRequestProperties();
request.FriendlyName = "Self-Signed Device Certificate";
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 = "MIIDezCCAmOgAwIBAgIBATANBgkqhkiG9w0BAqsFADBUMQswCQYDVQQGEwJVUzEY" + "MBYGA1UEChMPVS5TliBHb3Z1cm5tZW50MQ0wCwYDVQQLEwRESVNBMRwwGgYDVQQD" + "EXNQxJ1YNj1ZCBQcm12YWN5IENMBM4XDTExMDQwMzE0NTQwMFoXDTI4MDQwMzE0" + "NTQwMFowVDELMAkGA1UEBhMCVVMxGDAWBgNVBAoTD1UuUy4gR292ZXJpdWVuY29t" + "MAwGA1UdDwEB/wQEAwIBAgIADB5MEEYDQYJKoZIhvcNAQEFBQADggEPADCCAQoC" + "G777BuS/EXmuoHiVctA0n58u4SZb6i9Jvw1gI3qIryGM2oxDSKPr36c7R2tFmAqo4m9" + "Nh5r5r2YW0YAhZdhIy5F9BIOZEN/BpYrvKziupf3OVTQaMjMWoI5DQ+C+Dp4OA/0x" + "8qUy4c8m9MoJ7uNzkdHbdagou1Gsj5t2y0lW37IbRo6HRz5D1l18IAx7s7n9k/6" + "M7bK4r/gF7TMvI5bBpN/Pp4ay13f+oyQbS+FPQwFbWGLLukTuzYcDVfUCAwEY" + "AaNYMFYwHQYDVR0OBBYEFAFy9PrSM65GyC0EVDPU91W0B5MAsGA1UdDwQEAwIC+" + "pDAobGnuNSUEItAFBAmgQEBFQdDyAQYKMwYWBUHUawEGCSsGAQQBjxcVJDANBkgqk" + "hkIG9w0BAQsFAOAAEJQEG777BuS/EXmuJHvCva0n58u4S2b619Jw1q13IryGM/" + "2oxSKPr36c7R2tFmAqo4m9N97wh4FxFebkkYHg2WFsp0vR79vEw+MCwz2OBB88" + "ir4a2z/oTDMW9uf3r+BaZrJKpVoaKY9e3wmpxe6D3A3wtdEUE2N4Gv5XIdiS/" + "pfVd4eYEvPNy0yP9ZDBB9v9Vc5d7VfG8rzQoaDcerwrsXJ9/WLDz76A6d2/syHN/" + "74CRuXyGhpB7YI1jHhgVi16Rb4Dbq3d9D1kmTqUecEknuX730ddr/hpgqMOrWVUB" + "1XrHJbPU++nuPbShhJ0vPRw13TX3deqjzTsj8EXc==";
byte[] privacyCaBytes = Convert.FromBase64String(privacyCa);
IBuffer buffer = privacyCaBytes.AsBuffer();
request.AttestationCredentialCertificate = new Certificate(buffer); ;

csrToDiscard = await CertificateEnrollmentManager.UserCertificateEnrollmentManager.
CreateRequestAsync(request);


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

600 1256:                         SEQUENCE {
604  9:                           OBJECT IDENTIFIER '1 3 6 1 4 1 311 21 24'
615 1241:                         SET {
619 1237:                           OCTET STRING
     4B 41 53 54 01 00 00 00 02 00 00 00 01 00 00 00
     00 00 00 00 B9 04 00 00 00 00 00 00 00 00 4B 41 44 53
     02 00 00 00 18 00 00 00 00 A1 00 00 00 00 00 01 00 00
     00 03 00 00 FF 54 43 47 80 17 00 22 00 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 89 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 30 BF 82 73 9C 00
     0B 2B E6 2C AD 8D E8 9A 85 04 D7 F3 7B 07 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 08 17 CF 31 08 F1 55 F2 9B 4E
     82 C8 5B 49 7B 1A F1 4B 12 A1 C5 D1 A4 C5 A4 59
     C4 0A 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 0B 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 36 6A
     01 B0 7D 35 D0 99 BA A1 77 35 76 7E 83 90 AB 8B
     86 27 B8 3D 47 75 2D 98 D0 23 4E 09 D8 26 6B 32
     3C AB AC 50 A2 EF 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 8C 80 32 2D 5C A3 60
     48 F5 5E 8E 65 6B 5E B5 0E A4 ED B9 8B F9 C3 D9
The format is structured as follows:

```c
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;
```

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 89 4D BF 7A
54 26 36 2F 10 D6 00 00 00 00 00 51 4F CB E5 AD 8C BC 60 E6 C2 70 80 00
D4 2C 65 4C 6B 95 ED 95 00 22 00 0B 2B E6 2C AD BD E8 9A 85 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

Richardson, et al.        Expires April 8, 2020                [Page 53]
Signature (256 bytes) - generated using the AIK private key

------------- 1A F1 4B 12 A1 C5 D1 A4 C5 A4 59 C4 0A 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
0B 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
8C 80 32 2D 5C A3 60 4B F5 5E 8E 6B B5 B0 E4 A4 ED B9 8B F9 C3 D9
A8 CE C0 64 71 F6 E3 81 F7 9D 79 E5 73 7B F3 A4 66 65 8D 72 B4 0A 3E
5E 70 5F AB 2B 8B 99 B5 65 4B 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 {
28  11:       SEQUENCE {
30  9:         OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
        }
41 1533:       [0] {
45  742:         SEQUENCE {
49  462:           SEQUENCE {

Richardson, et al. Expires April 8, 2020 [Page 54]
[0] {
  INTEGER 2

INTEGER 00 A4 85 22 AA 34 AF AE 4F

SEQUENCE {
  OBJECT IDENTIFIER
  sha256WithRSAEncryption (1 2 840 113549 1 1 11)
  NULL
}

SEQUENCE {
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER commonName (2 5 4 3)
      UTF8String 'Yubico PIV Root CA Serial 263751'
    }
  }
}

SEQUENCE {
  UTCTime 14/03/2016 00:00:00 GMT
  GeneralizedTime 17/04/2052 00:00:00 GMT
}

SEQUENCE {
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER commonName (2 5 4 3)
      UTF8String 'Yubico PIV Attestation'
    }
  }
}

SEQUENCE {
  OBJECT IDENTIFIER
  rsaEncryption (1 2 840 113549 1 1 1)
  NULL
}

BIT STRING
  30 82 01 0A 02 82 01 01 00 AB A9 0B 16 9B EF 31
  CC 3E AC 18 5A 02 45 80 75 70 0C 06 B0 6C 3F 1B
  39 0D 49 B9 59 E8 6F CE BB 27 6F D8 3C 08 3A 85
  00 EF 5C BC 40 99 3D 41 EE EA C0 81 7F 76 48 E4
  A9 4C BC 0A 02 6B E1 1F 0A 60 93 C6 0E AA D2 8D 8E
  E2 BD CD 8B 1B 0B FD 9B 2F CF B9 0E 54 CE
  EC 8D F5 5E D7 7B 91 C3 A7 56 9C DC C1 06 86 76
  4E 44 53 0B 08 25 D8 06 B9 06 8C 01 8D 63 67 CA
  [ Another 142 bytes skipped ]
}

[3] {
  SEQUENCE {

SEQUENCE {
  OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3'
  OCTET STRING 04 03 03
  }

SEQUENCE {
  OBJECT IDENTIFIER
    sha256WithRSAEncryption (1 2 840 113549 1 1 11)
  NULL
  }

BIT STRING
  52 80 5A 6D C3 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 B9 D4
  B3 02 1A 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 6B 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 ]

SEQUENCE {
  [0] {
    INTEGER 2
    }
  }

INTEGER
  00 FE B9 AF 03 3B 0B A7 79 04 02 F5 67 AE DF 72
  ED

SEQUENCE {
  OBJECT IDENTIFIER
    sha256WithRSAEncryption (1 2 840 113549 1 1 11)
  NULL
  }

SEQUENCE {
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER commonName (2 5 4 3)
      UTF8String 'Yubico PIV Attestation'
    }
  }
}

SEQUENCE {
  UTCTime 14/03/2016 00:00:00 GMT
  GeneralizedTime 17/04/2052 00:00:00 GMT
  }

Richardson, et al. Expires April 8, 2020 [Page 56]
907  37:    SEQUENCE {
909  35:      SET {
911  33:        SEQUENCE {
913  31:          OBJECT IDENTIFIER commonName (2 5 4 3)
918  26:          UTF8String 'YubiKey PIV Attestation 9e'
933  22:          };
946  290:      SEQUENCE {
950  13:        SEQUENCE {
952  9:          OBJECT IDENTIFIER
963  0:            rsaEncryption (1 2 840 113549 1 1 1)
965  271:          NULL
988  216:          };
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:            };
1265  10:            SEQUENCE {
1277  5:              OCTET STRING 02 03 4F 9B B5  -- serial number
1284  16:            };
1286  10:            SEQUENCE {
1298  2:              OCTET STRING 01 01  -- PIN and touch policy
1302  13:            };
1304  9:            OBJECT IDENTIFIER
1315  0:              sha256WithRSAEncryption (1 2 840 113549 1 1 11)
1317  257:          BIT STRING
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)
36 :    }
41 1570:  [0] {
45  762:    SEQUENCE {
49 482:      SEQUENCE {
53  3:        [0] {
55  1:          INTEGER 2
58 :        }
60  9:          INTEGER 00 86 77 17 E0 1D 19 2B 26
69 13:        SEQUENCE {
71  9:          OBJECT IDENTIFIER
74 :            sha256WithRSAEncryption (1 2 840 113549 1 1 11)
78  0:            NULL
82 :        }
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'
99 :            }
103 :          }
129 32:        SEQUENCE {
133 :      }
135 :    }
141 :  }
143 :}

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
: 6B 82 88 FE DA 70 FC 5C B8 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 6B F8 63 5C 83 CB 83 38 62 B7 64 AE 83 37
: 37 8E C8 60 80 E6 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
: 0D F7 9F 20 A0 28 A0 9F B3 B1 22 5F AF 38 FB 73
: 46 F4 C7 93 30 DD FA D0 B6 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
SEQUENCE {
  [0] {
    INTEGER 2
  }
  INTEGER
    17 7D 2D F7 D6 6D 97 CC D6 CF 69 33 87 5B F1 5E
  SEQUENCE {
    OBJECT IDENTIFIER
      sha256WithRSAEncryption (1 2 840 113549 1 1 11)
    NULL
  }
  SEQUENCE {
    SET {
      commonName (2 5 4 3)
      UTF8String 'Yubico PIV Attestation'
    }
  }
  SEQUENCE {
    UTCTime 14/03/2016 00:00:00 GMT
    GeneralizedTime 17/04/2052 00:00:00 GMT
  }
  SEQUENCE {
    SET {
      commonName (2 5 4 3)
      UTF8String 'YubiKey PIV Attestation 9e'
    }
  }
  SEQUENCE {
    rsaEncryption (1 2 840 113549 1 1 1)
    NULL
  }
}
BIT STRING
984 271:  30 82 01 0A 02 82 01 02 00 A9 02 2D 7A 4C 0B 1C
:  0C 02 F9 E5 9C E5 6F 20 D1 9D F9 CE B3 B3 4D 1B
:  61 B0 B4 E0 3F 44 19 72 88 8B 8D 9F 86 4A 5E C7
:  38 F0 AF C9 28 5C D8 A2 B0 C9 43 93 2D FA 39 7F
:  E9 39 2D 1B 7A A2 67 76 88 8B 8D 9F 86 4A 5E C7
:  06 37 9D 90 D5 71 00 6E FB 82 D1 5B 2A 7C 3B 62
:  9E AB 15 81 B8 AD 7F 3D 30 1C 4B 9D C4 D5 64
:  32 9A 54 D6 23 B1 65 92 A3 D7 57 E2 62 10 2B 93
:                    [ Another 142 bytes skipped ]

Richardson, et al.          Expires April 8, 2020              [Page 61]
1615  0:  SET {
  :  
  :  
  :  

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

[android_security]

[azureattestation]
[keystore]

[keystore_attestation]


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
Carl Wallace
Red Hound Software
Email: carl@redhoundsoftware.com

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