Remote Attestation Procedures Architecture
draft-birkholz-rats-architecture-02

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.

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 March 15, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
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 Attesters 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 trustworthiness attributes of other Computing Environments,
- collecting trustworthiness 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. Principals that are Supply Chain Entities). There can be multiple forms of appraisal (e.g., software integrity verification, device composition and configuration verification, device identity and provenance verification). Attestation Results are the output of appraisals. Attestation Results are signed and conveyed to Relying Parties. Attestation Results provide the basis by which the Relying Party may determine a level of confidence to place in the application data or operations that follow.

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 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 Endorsments 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 Endorsments. 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 Attester.

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 Attester.

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.

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
o 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
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 or more TPM 2.0 embedded in the components of a
composite device — sometimes also referred to as an aggregate device
—are required in order to use the YANG module defined in this
document. A TPM 2.0 is used as a root of trust for reporting (RTR)
in order to retrieve attestation evidence from a composite device.
Additionally, it is used as a root of trust for measurement (RTM) in
order to provide event logs — sometimes also referred to as
measurement logs.
1.1. Requirements notation

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

2. The YANG Module for Basic Remote Attestation Procedures

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

2.1. Tree format

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

rpcs:
   +++-x tpm12-challenge-response-attestation
   |   +++-w input
   |   |   |   +++-w tpm1-attestation-challenge
   |   |   |   |   +++-w pcr-indices* uint8
   |   |   |   |   +++-w nonce-value binary
   |   |   |   |   +++-w TPM_SIG_SCHEME-value uint8
</CODE>
++-w (key-identifier)?
    +--w (public-key)
        |    +--w pub-key-id?    binary
        +--w (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

Internet-Draft                    BRAT                         July 2019

+----w (algo-registry-type)
   |   +--:(tcg)
   |   |   |   |   +----w tcg-hash-algo-id?       uint16
   |   |   +--:(ietf)
   |   |       |   +----w ietf-ni-hash-algo-id?   uint8
   |   +----w nonce-value                   binary

+----w (signature-identifier-type)
   |   +--:(TPM_ALG_ID)
   |       |   |   +----w TPM_ALG_ID-value?       uint16
   |   +--:(COSE_Algorithm)
   |       +----w COSE_Algorithm-value?   int32
   +----w (key-identifier)?
       |   |   +--:(public-key)
       |   |       |   +----w pub-key-id?             binary
       |   +--:(uuid)
       |       +----w uuid-value?             binary

+----w tpms* [tpm_name]  
   |   +----w tpm_name              string
   |   +----w tpm-physical-index?   int32 {ietfhw:entity-mib}?

---ro output
   +--ro tpm20-attestation-response* [tpm_name]
       |   +----ro tpm_name              string
       |   +----ro tpm-physical-index?   int32 {ietfhw:entity-mib}?
       |   +----ro up-time?               uint32
       |   +----ro node-name?             string
       +----ro tpms-attest
           |   |   +--ro pcrdigest?                   binary
           |   +--ro tpms-attest-result?          binary
           +--ro tpms-attest-result-length?   uint32
           +--ro tpmt-signature?        binary

---x basic-trust-establishment
   +----w input
       |   +----w nonce-value                   binary
       |   +----w (signature-identifier-type)
       |       +--:(TPM_ALG_ID)
       |           |   |   +----w TPM_ALG_ID-value?       uint16
       |       +--:(COSE_Algorithm)
       |           |   |   +----w COSE_Algorithm-value?   int32
       |       +----w tpm_name?                     string
       |       +----w tpm-physical-index?   int32 {ietfhw:entity-mib}?
       +----w certificate-name?             string

---ro output
   +--ro attestation-certificates* [tpm_name]
       |   +--ro tpm_name              string
       |   +--ro tpm-physical-index?   int32 {ietfhw:entity-mib}?
       |   +--ro up-time?               uint32
       +--ro node-name?             string
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
SDO.
uses hash-algo;
leaf hash-value {
  type binary;
  description
  "The binary representation of the hash value.";
}

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

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.";
      }
    }
  }
}

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.";

case TSS_UUID {

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.";

  container TSS_UUID-value {
    description
    "A detailed structure that is used to create the
    TPM 1.2 native TSS_UUID as defined in the TCG Software
    Stack (TSS) Specification, Version 1.10 Golden,
    August 20, 2003.";

    leaf ulTimeLow {
      type uint32;
      description
      "The low field of the timestamp.";
    }

    leaf usTimeMid {
      type uint16;
      description
      "The middle field of the timestamp.";
    }

    leaf usTimeHigh {
      type uint16;
      description
      "The high field of the timestamp multiplexed with the
      version number.";
    }

    leaf bClockSeqHigh {
      type uint8;
      description
      "The high field of the clock sequence multiplexed with
      the variant.";
    }

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

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

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

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

grouping tpm12-version {
  description
    "This structure provides information relative the version of
the TPM.";
  list version {
    description
      "This indicates the version of the structure
(TPM 1.2 type TPM_STRUCT_VER). This MUST be 1.1.0.0.";
    leaf major {
      type uint8;
      description
        "Indicates the major version of the structure.
MUST be 0x01.";
    }
    leaf minor {
      type uint8;
      description
        "Indicates the minor version of the structure.
MUST be 0x01.";
    }
    leaf revMajor {
      type uint8;
      description
        "Indicates the rev major version of the structure.
MUST be 0x00.";
    }
    leaf revMinor {
      type uint8;
      description
        "Indicates the rev minor version of the structure.
MUST be 0x00.";
    }
  }
}

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

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

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

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

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

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

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

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


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

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

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

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

rpc tpm12-challenge-response-attestation {
    description "This RPC accepts the input for TSS TPM 1.2 commands of the managed device. ComponentIndex from the hardware manager YANG module to refer to dedicated TPM in composite devices, e.g. smart NICs, is still a TODO.";
    input {
        container tpm1-attestation-challenge {
            description "This container includes every information element defined in the reference challenge-response interaction model for remote attestation. Corresponding values are based on TPM 1.2 structure definitions";
            uses tpm12-pcr-selection;
            uses nonce;
            uses tpm12-signature-scheme;
            uses tpm12-attestation-key-identifier;
            leaf add-version {
                type boolean;
                description "Whether or not to include TPM_CAP_VERSION_INFO; if true, then TPM_Quote2 must be used to create the response.";
            }
            uses tpm-name;
        }
    }
}
output {
  list tpm12-attestation-response {
    key tpm_name;
    description
      "The binary output of TPM 1.2 TPM_Quote/TPM_Quote2, including
       the PCR selection and other associated attestation evidence
       metadata";
    uses tpm-name;
    uses node-uptime;
    uses compute-node;
    uses tpm12-quote-info-common;
    choice tpm12-quote {
      mandatory true;
      description
        "Either a tpm12-quote-info or tpm12-quote-info2, depending
         on whether TPM_Quote or TPM_Quote2 was used
         (cf. input field add-version).";
      case tpm12-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 nodeuptime;
  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.";
}
rpc basic-trust-establishment {
  description
  "This RPC creates a tpm-resident, non-migratable key to be used
  in TPM_Quote commands, an attestation certificate.";
  input {
    uses nonce;
    uses tpm20-signature-scheme;
    uses tpm-name;
    leaf certificate-name {
      type string;
      description
      "An arbitrary name for the identity certificate chain
      requested.";
    }
  }
  output {
    list attestation-certificates {
      key tpm_name;
      description
      "Attestation Certificate data from a TPM identified by the TPM
      name";
      uses tpm-name;
      uses node-uptime;
      uses compute-node;
      leaf certificate-name {
        type string;
        description
        "An arbitrary name for this identity certificate or
        certificate chain.";
      }
      leaf attestation-certificate {
        type ietfct:end-entity-cert-cms;
        description
        "The binary signed certificate chain data for this identity
        certificate.";
      }
      uses tpm20-attestation-key-identifier;
    }
  }
}

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

```yaml
input {
  list log-selector {
    key node-name;
    description "Selection of log entries to be reported.";
    uses compute-node;
    choice index-type {
      description "Last log entry received, log index number, or timestamp.";
      case last-entry {
        description "The last entry of the log already retrieved.";
        leaf last-entry-value {
          type binary;
          description "Content of an log event which matches 1:1 with a unique event record contained within the log. Log entries subsequent to this will be passed to the requester. Note: if log entry values are not unique, this MUST return an error.";
        }
      }
      case index {
        description "Numeric index of the last log entry retrieved, or zero.";
        leaf index-number {
          type uint64;
          description "The numeric index number of a log entry. Zero means to start at the beginning of the log. Entries subsequent to this will be passed to the requester.";
        }
      }
      case timestamp {
        leaf timestamp {
          type yang:date-and-time;
          description "Timestamp from which to start the extraction. The next log entry subsequent to this timestamp is to be sent.";
        }
        description "Timestamp from which to start the extraction.";
      }
    }
  }
}
```
uses log-identifier;
uses tpm20-pcr-selection;
leaf log-entry-quantity {
  type uint16;
  description
    "The number of log entries to be returned. If omitted, it
      means all of them.";
}

output {
  container system-event-logs {
    description
      "The requested data of the measurement event logs";
    list node-data {
      key "node-name tpm_name";
      description
        "Event logs of a node in a distributed system
          identified by the node name";
      uses compute-node;
      uses node-uptime;
      uses tpm-name;
      container log-result {
        description
          "The requested entries of the corresponding log.";
        uses event-logs;
      }
    }
  }
}

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

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

list compute-nodes {
  key node-name;
  uses compute-node;
  description
  "A list names of hardware components in this composite device that rats can be conducted with.";
}
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
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 a new assertion about properties and characteristics of the Attester, i.e., whether or not it is compliant to policies, or even can be "trusted".

7. Further Context

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

7.1. Confidentiality

Confidentiality of exchanged attestation information may be desirable. This requirement usually is present when communication takes place over insecure channels, such as the public Internet. In such cases, TLS may be 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-
ariculture-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
nenonce: 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
Michael Eckel  
Fraunhofer SIT  
Rheinstrasse 75  
Darmstadt 64295  
Germany  

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

This documents defines the method and bindings used to conduct Time-based Uni-Directional Attestation (TUDA) between two RATS (Remote ATtestation procedureS) Principals over the Internet. TUDA does not require a challenge-response handshake and thereby does not rely on the conveyance of a nonce to prove freshness of remote attestation Evidence. Conversely, TUDA enables the creation of Secure Audit Logs that can constitute Evidence about current and past operational states of an Attester. As a prerequisite for TUDA, every RATS Principal requires access to a trusted and synchronized time-source. Per default, in TUDA this is a Time Stamp Authority (TSA) issuing signed Time Stamp Tokens (TST).

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 March 15, 2020.
Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
  1.1. Requirements Notation .................................. 4
  1.2. Evidence .................................................. 4
  1.3. Creating Evidence about Software Component Integrity ........ 5
    1.3.1. Data Items ........................................... 5
    1.3.2. System Components ................................... 5
    1.3.3. Operations ............................................. 6
  1.4. Remote Attestation Principles ................................ 6
  1.5. System Component Requirements ................................ 7
  1.6. Evidence Appraisal ......................................... 7
  1.7. Activities and Actions ..................................... 8
  1.8. Attestation and Verification ................................ 8
  1.9. Information Elements and Conveyance .......................... 8
  1.10. TUDA Objectives ........................................... 9
  1.11. Hardware Dependencies ...................................... 10
2. TUDA Core Concept ............................................. 10
3. Terminology ................................................... 11
  3.1. Universal Terms ............................................ 11
  3.2. Roles ....................................................... 13
    3.2.1. General Types .......................................... 13
    3.2.2. RoT specific terms ..................................... 13
  3.3. Certificates ................................................ 13
4. Time-Based Uni-Directional Attestation ........................ 13
  4.1. TUDA Information Elements Update Cycles ....................... 15
5. Sync Base Protocol ............................................ 17
6. IANA Considerations ........................................... 18
7. Security Considerations ........................................ 18
8. Change Log .................................................... 18
9. Contributors ................................................... 19
10. References .................................................... 19
    10.1. Normative References .................................... 19
1. Introduction

Remote ATtestation procedures (RATS) describe the attempt to determine and appraise properties, such as integrity and trustworthiness, of a communication partner - the Attester - over the Internet to another communication partner - the Verifier - without direct access. TUDA uses the architectural constituents of the RATS Architecture [I-D.birkholz-rats-architecture] that defines the Roles Attester and Verifier in detail. The RATS Architecture also defines Role Messages. TUDA creates and conveys a specific type of Role Message called Evidence, a composition of trustworthiness Claims provided by an Attester and consumed by a Verifier (potentially relayed by another RATS Role that is a Relying Party). TUDA - in contrast to traditional bi-directional challenge-response protocols
Internet-Draft                    TUDA                    September 2019

[I-D.birkholz-rats-reference-interaction-model] - enables a unidirectional conveyance of attestation Evidence that allows for providing attestation information without solicitation (e.g. as beacons or push data via YANG Push [RFC8641], [RFC8640], [RFC8639]).

As a result, this document introduces the term Forward Authenticity.

Forward Authenticity (FA): A property of secure communication protocols, in which later compromise of the long-term keys of a data origin does not compromise past authentication of data from that origin. FA is achieved by timely recording of assessments of the authenticity from system components (via "audit logs" during "audit sessions") that are authorized for this purpose and trustworthy (e.g via endorsed roots of trust), in a time frame much shorter than that expected for the compromise of the long-term keys.

Forward Authenticity enables new levels of assurance and can be included in basically every protocol, such as ssh, YANG Push, router advertisements, link layer neighbor discovery, or even ICMP echo.

1.1. Requirements Notation

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

1.2. Evidence

Remote attestation Evidence is basically a set of trustworthiness claims (assertions about the Attester and its system characteristics including security posture and protection characteristics) that are accompanied by a proof of their veracity - typically a signature based on shielded, private and potentially restricted key material. As key material alone is typically not self-descriptive with respect to its intended use (its semantics), the remote attestation Evidence created via TUDA is accompanied by two kinds of certificates that are cryptographically associated with a Trust Anchor (TA) [RFC4949] via a certification path:

- an Attestation Key (AK) Certificate (AK-Cert) that represents the attestation provenance of the created Evidence, and
o an Endorsement Key (EK) Certificate (EK-Cert) that represents the protection characteristics of the system components the AK is stored in.

If a Verifier decides to trust both the TA of an AK-Cert and an EK-Cert presented by an Attester - and the included assertions about the system characteristics describing the Attester, the attestation Evidence created via TUDA by the Attester is considered believable. Ultimately, believable Evidence is appraised by a Verifier in order to assess the trustworthiness of the corresponding Attester.

1.3. Creating Evidence about Software Component Integrity

The TUDA protocol mechanism uses hash values of all started software components as a basis to provide and create Evidence about the integrity of the software components of an Attester. This section defines the processed data items, the required system components, and corresponding operations to enable the creation of Evidence about software component integrity for TUDA.

1.3.1. Data Items

The hash value of a software component created before it is executed is referred to as a "measurement" in the remainder of this document. Measurements are chained using a rolling hash function. Each measurement added to the sequence of all measurements results in a new current hash value that is referred to as a "digest" in the remainder of this document.

1.3.2. System Components

The function to store these measurements via a rolling hash function is provided by a root of trust for storage - a system component that MUST be a component of the attester.

With respect to the boot sequence of an Attester, the very first measurements of software components (e.g. the BIOS, or a sometimes a bootloader) have to be conducted by a root of trust for measurement that is implemented in hardware and MUST be a system component of the Attester.

All measurements retained in the root of trust for measurements are handed over to the root of trust for storage when it becomes available during the boot procedure of the Attester. During that hand-over the sequence of measurements retained in the root of trust for measurement are processed by the rolling hash function of the root of trust for storage.
The function of retrieving the current output value of the rolling hash function, including a signature to provide a proof of veracity, is provided by a root of trust for reporting and MUST be a system component of the Attester.

Typically, a root of trust for storage and a root of trust for reporting are tightly coupled. Analogously, a root of trust for measurement is typically independent from the root of trust for storage, but has to be able to interact with root of trust for storage at some point of the boot sequence of the Attester to hand over the retained measurements.

1.3.3. Operations

The operation of processing a measurement and adding it to the sequence of measurements via the rolling hash function is called "extend" and is provided by the root of trust for storage.

The operation of retrieving the current available hash value that is the result of the rolling hash function including a signature based on an Attestation Key is called "quote" and is provided by the corresponding root of trust for reporting.

1.4. Remote Attestation Principles

In essence, RATS are composed of three base activities. The following definitions are derived from the definitions presented in [PRIRA] and [TCGGLOSS], and are a simplified summary of the RATS Architecture relevant for TUDA. The complete RATS Architecture and every corresponding constituent, message and interaction is defined in [I-D.birkholz-rats-architecture].

Attestation: The creation of one ore more claims about the trustworthiness properties of an Attester, such that the claims can be used as Evidence.

Conveyance: The transfer of Evidence from the Attester to the Verifier via an interconnect.

Verification: The appraisal of Evidence by evaluating it against known-good-values (a type of declarative guidance).

With TUDA, the claims that compose the evidence are signatures over trustworthy integrity measurements created by leveraging roots of trust. The evidence is appraised via corresponding signatures over reference integrity measurements (RIM, represented, for example via [I-D.ietf-sacm-coswid]).
Protocols that facilitate Trust-Anchor based signatures in order to provide RATS are usually bi-directional challenge/response protocols, such as the Platform Trust Service protocol [PTS] or CAVES [PRIRA], where one entity sends a challenge that is included inside the response to prove the recentness - the freshness (see fresh in [RFC4949]) - of the attestation information. The corresponding interaction model tightly couples the three activities of creating, transferring and appraising evidence.

The Time-Based Uni-directional Attestation family of protocols - TUDA - described in this document can decouple the three activities RATS are composed of. As a result, TUDA provides additional capabilities, such as:

- remote attestation for Attesters that might not always be able to reach the Internet by enabling the verification of past states,
- secure audit logs by combining the evidence created via TUDA with integrity measurement logs that represent a detailed record of corresponding past states,
- an uni-directional interaction model that can traverse "diode-like" network security functions (NSF) or can be leveraged in RESTful architectures (e.g. CoAP [RFC7252]), analogously.

1.5. System Component Requirements

TUDA is a family of protocols that bundles results from specific attestation activities. The attestation activities of TUDA are based on a hardware roots of trust that provides the following capabilities:

- Platform Configuration Registers (PCR) that can extend measurements consecutively and represent the sequence of measurements as a single digest,
- Restricted Signing Keys (RSK) that can only be accessed, if a specific signature about a set of measurements can be provided as authentication, and
- a dedicated source of (relative) time, e.g. a tick counter (a tick being a specific time interval, for example 10 ms).

1.6. Evidence Appraisal

To appraise the evidence created by an Attester, the Verifier requires corresponding Reference Integrity Measurements (RIM). Typical set of RIMs are required to assess the integrity of an
Attester. These sets are called RIM Bundles. The scope of a RIM Bundle encompasses, e.g., a platform, a device, a computing context, or a virtualised function. In order to be comparable, the hashing algorithms used by the Attester to create the integrity measurements have to match the hashing algorithms used to create the corresponding RIM that are used by the Verifier to appraise the attestation Evidence about software component integrity.

1.7. Activities and Actions

Depending on the platform (i.e. one or more computing contexts including a dedicated hardware RoT), a generic RA activity results in platform-specific actions that have to be conducted. In consequence, there are multiple specific operations and data models (defining the input and output of operations). Hence, specific actions are are not covered by this document. Instead, the requirements on operations and the information elements that are the input and output to these operations are illustrated using pseudo code in Appendix C and D.

1.8. Attestation and Verification

Both the attestation and the verification activity of TUDA also require a trusted Time Stamp Authority (TSA) as an additional third party next to the Attester and the Verifier. The protocol uses a Time Stamp Authority based on [RFC3161]. The combination of the local source of time provided by the hardware RoT (located on the Attester) and the Time Stamp Tokens provided by the TSA (to both the Attester and the Verifier) enable the attestation and verification of an appropriate freshness of the evidence conveyed by the Attester -- without requiring a challenge/response interaction model that uses a nonce to ensure the freshness.

Typically, the verification activity requires declarative guidance (representing desired or compliant endpoint characteristics in the form of RIM, see above) to appraise the individual integrity measurements the conveyed evidence is composed on. The acquisition or representation (data models) of declarative guidance as well as the corresponding evaluation methods are out of the scope of this document.

1.9. Information Elements and Conveyance

TUDA defines a set of information elements (IE) that are created and stored on the Attester and are intended to be transferred to the Verifier in order to enable appraisal. Each TUDA IE:

- is encoded in the Concise Binary Object Representation (CBOR [RFC7049]) to minimize the volume of data in motion. In this
document, the composition of the CBOR data items that represent IE is described using the Concise Data Definition Language, CDDL [RFC8610]

- that requires a certain freshness is only created/updated when out-dated, which reduces the overall resources required from the Attester, including the utilization of the hardware root of trust. The IE that have to be created are determined by their age or by specific state changes on the Attester (e.g. state changes due to a reboot-cycle)

- is only transferred when required, which reduces the amount of data in motion necessary to conduct remote attestation significantly. Only IE that have changed since their last conveyance have to be transferred

- that requires a certain freshness can be reused for multiple remote attestation procedures in the limits of its corresponding freshness-window, further reducing the load imposed on the Attester and its corresponding hardware RoT.

1.10. TUDA Objectives

The Time-Based Uni-directional Attestation family of protocols is designed to:

- increase the confidence in authentication and authorization procedures,

- address the requirements of constrained-node networks,

- support interaction models that do not maintain connection-state over time, such as REST architectures [REST],

- be able to leverage existing management interfaces, such as SNMP [RFC3411]. RESTCONF [RFC8040] or CoMI [I-D.ietf-core-comi] -- and corresponding bindings,

- support broadcast and multicast schemes (e.g. [IEEE1609]),

- be able to cope with temporary loss of connectivity, and to

- provide trustworthy audit logs of past endpoint states.
1.11. Hardware Dependencies

The binding of the attestation scheme used by TUDA to generate the TUDA IE is specific to the methods provided by the hardware RoT used (see above). In this document, expositional text and pseudo-code that is provided as a reference to instantiate the TUDA IE is based on TPM 1.2 and TPM 2.0 operations. The corresponding TPM commands are specified in [TPM12] and [TPM2]. The references to TPM commands and corresponding pseudo-code only serve as guidance to enable a better understanding of the attestation scheme and is intended to encourage the use of any appropriate hardware RoT or equivalent set of functions available to a CPU or Trusted Execution Environment [TEE].

2. TUDA Core Concept

There are significant differences between conventional bi-directional attestation and TUDA regarding both the information elements conveyed between Attester and Verifier and the time-frame, in which an attestation can be considered to be fresh (and therefore trustworthy).

In general, remote attestation using a bi-directional communication scheme includes sending a nonce-challenge within a signed attestation token. Using the TPM 1.2 as an example, a corresponding nonce-challenge would be included within the signature created by the TPM_Quote command in order to prove the freshness of the attestation response, see e.g. [PTS].

In contrast, the TUDA protocol uses the combined output of TPM_CertifyInfo and TPM_TickStampBlob. The former provides a proof about the platform’s state by creating evidence that a certain key is bound to that state. The latter provides proof that the platform was in the specified state by using the bound key in a time operation. This combination enables a time-based attestation scheme. The approach is based on the concepts introduced in [SCALE] and [SPKE2008].

Each TUDA IE has an individual time-frame, in which it is considered to be fresh (and therefore trustworthy). In consequence, each TUDA IE that composes data in motion is based on different methods of creation.

The freshness properties of a challenge-response based protocol define the point-of-time of attestation between:

- the time of transmission of the nonce, and
- the reception of the corresponding response.
Given the time-based attestation scheme, the freshness property of TUDA is equivalent to that of bi-directional challenge response attestation, if the point-in-time of attestation lies between:

- the transmission of a TUDA time-synchronization token, and
- the typical round-trip time between the Verifier and the Attester.

The accuracy of this time-frame is defined by two factors:

- the time-synchronization between the Attester and the TSA. The time between the two tickstamps acquired via the hardware RoT define the scope of the maximum drift ("left" and "right" in respect to the timeline) to the TSA timestamp, and
- the drift of clocks included in the hardware RoT.

Since the conveyance of TUDA evidence does not rely upon a Verifier provided value (i.e. the nonce), the security guarantees of the protocol only incorporate the TSA and the hardware RoT. In consequence, TUDA evidence can even serve as proof of integrity in audit logs with precise point-in-time guarantees, in contrast to classical attestations.

Appendix A contains guidance on how to utilize a REST architecture.

Appendix B contains guidance on how to create an SNMP binding and a corresponding TUDA-MIB.

Appendix C contains a corresponding YANG module that supports both RESTCONF and CoMI.

Appendix D.2 contains a realization of TUDA using TPM 1.2 primitives.

Appendix D.3 contains a realization of TUDA using TPM 2.0 primitives.

3. Terminology

This document introduces roles, information elements and types required to conduct TUDA and uses terminology (e.g. specific certificate names) typically seen in the context of attestation or hardware security modules.

3.1. Universal Terms

**Attestation Identity Key (AIK):** a special purpose signature (therefore asymmetric) key that supports identity related operations. The private portion of the key pair is maintained...
confidential to the entity via appropriate measures (that have an impact on the scope of confidence). The public portion of the key pair may be included in AIK credentials that provide a claim about the entity.

Claim: A piece of information asserted about a subject [RFC4949]. A claim is represented as a name/value pair consisting of a Claim Name and a Claim Value [RFC7519].

In the context of SACM, a claim is also specialized as an attribute/value pair that is intended to be related to a statement [I-D.ietf-sacm-terminology].

Endpoint Attestation: the creation of evidence on the Attester that provides proof of a set of the endpoints’s integrity measurements. This is done by digitally signing a set of PCRs using an AIK shielded by the hardware RoT.

Endpoint Characteristics: the context, composition, configuration, state, and behavior of an endpoint.

Evidence: a trustworthy set of claims about an endpoint’s characteristics.

Identity: a set of claims that is intended to be related to an entity.

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

Reference Integrity Measurements: Signed measurements about the characteristics of an endpoint’s characteristics that are provided by a vendor and are intended to be used as declarative guidance [I-D.ietf-sacm-terminology] (e.g. a signed CoSWID).

Trustworthy: the qualities of an endpoint that guarantee a specific behavior and/or endpoint characteristics defined by declarative guidance. Analogously, trustworthiness is the quality of being trustworthy with respect to declarative guidance. Trustworthiness is not an absolute property but defined with respect to an entity, corresponding declarative guidance, and has a scope of confidence.

Trustworthy Endpoint: an endpoint that guarantees trustworthy behavior and/or composition (with respect to certain declarative guidance and a scope of confidence).
Trustworthy Statement: evidence that is trustworthy conveyed by an endpoint that is not necessarily trustworthy.

3.2. Roles

Attester: the endpoint that is the subject of the attestation to another endpoint.

Verifier: the endpoint that consumes the attestation of another endpoint to conduct a verification.

TSA: a Time Stamp Authority [RFC3161]

3.2.1. General Types

Byte: the now customary synonym for octet

Cert: an X.509 certificate represented as a byte-string

3.2.2. RoT specific terms

PCR: a Platform Configuration Register that is part of a hardware root of trust and is used to securely store and report measurements about security posture

PCR-Hash: a hash value of the security posture measurements stored in a TPM PCR (e.g. regarding running software instances) represented as a byte-string

3.3. Certificates

TSA-CA: the Certificate Authority that provides the certificate for the TSA represented as a Cert

AIK-CA: the Certificate Authority that provides the certificate for the attestation identity key of the TPM. This is the client platform credential for this protocol. It is a placeholder for a specific CA and AIK-Cert is a placeholder for the corresponding certificate, depending on what protocol was used. The specific protocols are out of scope for this document, see also [AIK-Enrollment] and [IEEE802.1AR].

4. Time-Based Uni-Directional Attestation

A Time-Based Uni-Directional Attestation (TUDA) consists of the following seven information elements. They are used to gain assurance of the Attester’s platform configuration at a certain point in time:
TSA Certificate: The certificate of the Time Stamp Authority that is used in a subsequent synchronization protocol token. This certificate is signed by the TSA-CA.

AIK Certificate: A certificate about the Attestation Identity Key (AIK) used. This may or may not also be an [IEEE802.1AR] IDevID or LDevID, depending on their setting of the corresponding identity property. ([AIK-Credential], [AIK-Enrollment]; see Appendix D.2.1.)

Synchronization Token: The reference for attestations are the relative timestamps provided by the hardware RoT. In order to put attestations into relation with a Real Time Clock (RTC), it is necessary to provide a cryptographic synchronization between these trusted relative timestamps and the regular RTC that is a hardware component of the Attester. To do so, a synchronization protocol is run with a Time Stamp Authority (TSA).

Restriction Info: The attestation relies on the capability of the hardware RoT to operate on restricted keys. Whenever the PCR values for the machine to be attested change, a new restricted key is created that can only be operated as long as the PCRs remain in their current state.

In order to prove to the Verifier that this restricted temporary key actually has these properties and also to provide the PCR value that it is restricted, the corresponding signing capabilities of the hardware RoT are used. It creates a signed certificate using the AIK about the newly created restricted key.

Measurement Log: Similarly to regular attestations, the Verifier needs a way to reconstruct the PCRs’ values in order to estimate the trustworthiness of the device. As such, a list of those elements that were extended into the PCRs is reported. Note though that for certain environments, this step may be optional if a list of valid PCR configurations (in the form of RIM available to the Verifier) exists and no measurement log is required.

Implicit Attestation: The actual attestation is then based upon a signed timestamp provided by the hardware RoT using the restricted temporary key that was certified in the steps above. The signed timestamp provides evidence that at this point in time (with respect to the relative time of the hardware RoT) a certain configuration existed (namely the PCR values associated with the restricted key). Together with the synchronization token this timestamp represented in relative time can then be related to the real-time clock.
Concise SWID tags: As an option to better assess the trustworthiness of an Attester, a Verifier can request the reference hashes (RIM, which are often referred to as golden measurements) of all started software components to compare them with the entries in the measurement log. References hashes regarding installed (and therefore running) software can be provided by the manufacturer via SWID tags. SWID tags are provided by the Attester using the Concise SWID representation [I-D.ietf-sacm-coswid] and bundled into a CBOR array (a RIM Manifest). Ideally, the reference hashes include a signature created by the manufacturer of the software to prove their integrity.

These information elements could be sent en bloc, but it is recommended to retrieve them separately to save bandwidth, since these elements have different update cycles. In most cases, retransmitting all seven information elements would result in unnecessary redundancy.

Furthermore, in some scenarios it might be feasible not to store all elements on the Attester endpoint, but instead they could be retrieved from another location or be pre-deployed to the Verifier. It is also feasible to only store public keys on the Verifier and skip the whole certificate provisioning completely in order to save bandwidth and computation time for certificate verification.

4.1. TUDA Information Elements Update Cycles

An endpoint can be in various states and have various information associated with it during its life cycle. For TUDA, a subset of the states (which can include associated information) that an endpoint and its hardware root of trust can be in, is important to the attestation process. States can be:

- persistent, even after a hard reboot. This includes certificates that are associated with the endpoint itself or with services it relies on.
- volatile to a degree, because they change at the beginning of each boot cycle. This includes the capability of a hardware RoT to provide relative time which provides the basis for the synchronization token and implicit attestation—and which can reset after an endpoint is powered off.
- very volatile, because they change during an uptime cycle (the period of time an endpoint is powered on, starting with its boot). This includes the content of PCRs of a hardware RoT and thereby also the PCR-restricted signing keys used for attestation.
Depending on this "lifetime of state", data has to be transported over the wire, or not. E.g. information that does not change due to a reboot typically has to be transported only once between the Attester and the Verifier.

There are three kinds of events that require a renewed attestation:

- The Attester completes a boot-cycle
- A relevant PCR changes
- Too much time has passed since the last attestation statement

The third event listed above is variable per application use case and also depends on the precision of the clock included in the hardware RoT. For usage scenarios, in which the device would periodically push information to be used in an audit-log, a time-frame of approximately one update per minute should be sufficient in most cases. For those usage scenarios, where Verifiers request (pull) a fresh attestation statement, an implementation could use the hardware RoT continuously to always present the most freshly created results. To save some utilization of the hardware RoT for other purposes, however, a time-frame of once per ten seconds is recommended, which would typically leave about 80% of utilization for other applications.

```
Attester
  |                                            Verifier
  Boot
  Create Sync-Token
  Create Restricted Key
  Certify Restricted Key
  AIK-Cert --------------------------------------------->
  Sync-Token ------------------------------------------>
  Certify-Info ---------------------------------------->
  Measurement Log -------------------------------------->
  Attestation ------------------------------------------>
  Verify Attestation
  <Time Passed>
  Attestation ------------------------------------------>
  Verify Attestation
  <Time Passed>
```
5. Sync Base Protocol

The uni-directional approach of TUDA requires evidence on how the TPM time represented in ticks (relative time since boot of the TPM) relates to the standard time provided by the TSA. The Sync Base Protocol (SBP) creates evidence that binds the TPM tick time to the TSA timestamp. The binding information is used by and conveyed via the Sync Token (TUDA IE). There are three actions required to create the content of a Sync Token:

- At a given point in time (called "left"), a signed tickstamp counter value is acquired from the hardware RoT. The hash of counter and signature is used as a nonce in the request directed at the TSA.

Figure 1: Example sequence of events
The corresponding response includes a data-structure incorporating the trusted timestamp token and its signature created by the TSA.

At the point-in-time the response arrives (called "right"), a signed tickstamp counter value is acquired from the hardware RoT again, using a hash of the signed TSA timestamp as a nonce.

The three time-related values -- the relative timestamps provided by the hardware RoT ("left" and "right") and the TSA timestamp -- and their corresponding signatures are aggregated in order to create a corresponding Sync Token to be used as a TUDA Information Element that can be conveyed as evidence to a Verifier.

The drift of a clock incorporated in the hardware RoT that drives the increments of the tick counter constitutes one of the triggers that can initiate a TUDA Information Element Update Cycle in respect to the freshness of the available Sync Token.

content TBD

6. IANA Considerations

This memo includes requests to IANA, including registrations for media type definitions.

TBD

7. Security Considerations

There are Security Considerations. TBD

8. Change Log

Changes from version 04 to I2NSF related document version 00: *
Refactored main document to be more technology agnostic * Added first draft of procedures for TPM 2.0 * Improved content consistency and structure of all sections

Changes from version 03 to version 04:

  o Refactoring of Introduction, intend, scope and audience

  o Added first draft of Sync Base Protocoll section illustrated background for interaction with TSA

  o Added YANG module

  o Added missing changelog entry
Changes from version 02 to version 03:

- Moved base concept out of Introduction
- First refactoring of Introduction and Concept
- First restructuring of Appendices and improved references

Changes from version 01 to version 02:

- Restructuring of Introduction, highlighting conceptual prerequisites
- Restructuring of Concept to better illustrate differences to handshake based attestation and deciding factors regarding freshness properties
- Subsection structure added to Terminology
- Clarification of descriptions of approach (these were the FIXMEs)
- Correction of RestrictionInfo structure: Added missing signature member

Changes from version 00 to version 01:

Major update to the SNMP MIB and added a table for the Concise SWID profile Reference Hashes that provides additional information to be compared with the measurement logs.

9. Contributors

TBD

10. References

10.1. Normative References

[I-D.birkholz-rats-architecture]


Fuchs, et al. Expires March 15, 2020


10.2. Informative References


[I-D.ietf-sacm-terminology]

[IEEE1609]

[IEEE802.1AR]

[PRIRA]

[PTS]

[REST]


Fuchs, et al. Expires March 15, 2020


Appendix A. REST Realization

Each of the seven data items is defined as a media type (Section 6). Representations of resources for each of these media types can be retrieved from URIs that are defined by the respective servers [RFC7320]. As can be derived from the URI, the actual retrieval is via one of the HTTPs ([RFC7230], [RFC7540]) or CoAP [RFC7252]. How a client obtains these URIs is dependent on the application; e.g., CoRE Web links [RFC6690] can be used to obtain the relevant URIs from the self-description of a server, or they could be prescribed by a RESTCONF data model [RFC8040].

Appendix B. SNMP Realization

SNMPv3 [STD62] [RFC3411] is widely available on computers and also constrained devices. To transport the TUDA information elements, an SNMP MIB is defined below which encodes each of the seven TUDA information elements into a table. Each row in a table contains a single read-only columnar SNMP object of datatype OCTET-STRING. The values of a set of rows in each table can be concatenated to reconstitute a CBOR-encoded TUDA information element. The Verifier can retrieve the values for each CBOR fragment by using SNMP GetNext requests to "walk" each table and can decode each of the CBOR-encoded data items based on the corresponding CDDL [RFC8610] definition.

Design Principles:

1. Over time, TUDA attestation values age and should no longer be used. Every table in the TUDA MIB has a primary index with the value of a separate scalar cycle counter object that disambiguates the transition from one attestation cycle to the next.

2. Over time, the measurement log information (for example) may grow large. Therefore, read-only cycle counter scalar objects in all TUDA MIB object groups facilitate more efficient access with SNMP GetNext requests.

3. Notifications are supported by an SNMP trap definition with all of the cycle counters as bindings, to alert a Verifier that a new attestation cycle has occurred (e.g., synchronization data, measurement log, etc. have been updated by adding new rows and possibly deleting old rows).
B.1. Structure of TUDA MIB

The following table summarizes the object groups, tables and their indexes, and conformance requirements for the TUDA MIB:

<table>
<thead>
<tr>
<th>Group/Table</th>
<th>Cycle</th>
<th>Instance</th>
<th>Fragment</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>AIKCert</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSACert</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SyncToken</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Restrict</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measure</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerifyToken</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWIDTag</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B.1.1. Cycle Index

A tudaV1<Group>CycleIndex is the:

1. first index of a row (element instance or element fragment) in the tudaV1<Group>Table;
2. identifier of an update cycle on the table, when rows were added and/or deleted from the table (bounded by tudaV1<Group>Cycles); and
3. binding in the tudaV1TrapV2Cycles notification for directed polling.

B.1.2. Instance Index

A tudaV1<Group>InstanceIndex is the:

1. second index of a row (element instance or element fragment) in the tudaV1<Group>Table; except for
2. a row in the tudaV1SyncTokenTable (that has only one instance per cycle).

B.1.3. Fragment Index

A tudaV1<Group>FragmentIndex is the:

1. last index of a row (always an element fragment) in the tudaV1<Group>Table; and
2. accommodation for SNMP transport mapping restrictions for large string elements that require fragmentation.

B.2. Relationship to Host Resources MIB

The General group in the TUDA MIB is analogous to the System group in the Host Resources MIB [RFC2790] and provides context information for the TUDA attestation process.

The Verify Token group in the TUDA MIB is analogous to the Device group in the Host MIB and represents the verifiable state of a TPM device and its associated system.

The SWID Tag group (containing a Concise SWID reference hash profile [I-D.ietf-sacm-coswid]) in the TUDA MIB is analogous to the Software Installed and Software Running groups in the Host Resources MIB [RFC2790].

B.3. Relationship to Entity MIB

The General group in the TUDA MIB is analogous to the Entity General group in the Entity MIB v4 [RFC6933] and provides context information for the TUDA attestation process.

The SWID Tag group in the TUDA MIB is analogous to the Entity Logical group in the Entity MIB v4 [RFC6933].

B.4. Relationship to Other MIBs

The General group in the TUDA MIB is analogous to the System group in MIB-II [RFC1213] and the System group in the SNMPv2 MIB [RFC3418] and provides context information for the TUDA attestation process.

B.5. Definition of TUDA MIB

<CODE BEGINS>
TUDA-V1-ATTESTATION-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, Integer32, Counter32,
    enterprises, NOTIFICATION-TYPE
    FROM SNMPv2-SMI -- RFC 2578
    MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
    FROM SNMPv2-CONF -- RFC 2580
    SnmpAdminString
    FROM SNMP-FRAMEWORK-MIB; -- RFC 3411

    tudaV1MIB MODULE-IDENTITY

The MIB module for monitoring of time-based unidirectional attestation information from a network endpoint system, based on the Trusted Computing Group TPM 1.2 definition.

Copyright (C) High North Inc (2019)."
DESCRIPTION
"Third version, published as draft-birkholz-tuda-02."

REVISION "201603210000Z" -- 21 March 2016
DESCRIPTION
"Second version, published as draft-birkholz-tuda-01."

REVISION "201510180000Z" -- 18 October 2015
DESCRIPTION
"Initial version, published as draft-birkholz-tuda-00."

::= { enterprises fraunhofersit(21616) mibs(1) tudaV1MIB(1) }

tudaV1MIBNotifications OBJECT IDENTIFIER ::= { tudaV1MIB 0 }
tudaV1MIBObjects OBJECT IDENTIFIER ::= { tudaV1MIB 1 }
tudaV1MIBConformance OBJECT IDENTIFIER ::= { tudaV1MIB 2 }

--
-- General
--
tudaV1General OBJECT IDENTIFIER ::= { tudaV1MIBObjects 1 }

tudaV1GeneralCycles OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of TUDA update cycles that have occurred, i.e.,
sum of all the individual group cycle counters.
DEFVAL intentionally omitted - counter object."
::= { tudaV1General 1 }

tudaV1GeneralVersionInfo OBJECT-TYPE
SYNTAX SnmpAdminString (SIZE(0..255))
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Version information for TUDA MIB, e.g., specific release
version of TPM 1.2 base specification and release version
of TPM 1.2 errata specification and manufacturer and model
TPM module itself."
DEFVAL { "" }
::= { tudaV1General 2 }

--
-- AIK Cert
--
tudaV1AIKCert OBJECT IDENTIFIER ::= { tudaV1MIBObjects 2 }

tudaV1AIKCertCycles OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION "Count of AIK Certificate chain update cycles that have occurred.
DEFVAL intentionally omitted - counter object."
 ::= { tudaV1AIKCert 1 }

tudaV1AIKCertTable OBJECT-TYPE
SYNTAX SEQUENCE OF TudaV1AIKCertEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "A table of fragments of AIK Certificate data."
 ::= { tudaV1AIKCertTable 1 }

tudaV1AIKCertEntry OBJECT-TYPE
SYNTAX TudaV1AIKCertEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "An entry for one fragment of AIK Certificate data."
INDEX { tudaV1AIKCertCycleIndex, tudaV1AIKCertInstanceIndex, tudaV1AIKCertFragmentIndex }
 ::= { tudaV1AIKCertTable 1 }

TudaV1AIKCertEntry ::= SEQUENCE {
    tudaV1AIKCertCycleIndex Integer32, 
    tudaV1AIKCertInstanceIndex Integer32, 
    tudaV1AIKCertFragmentIndex Integer32, 
    tudaV1AIKCertData OCTET STRING 
}

tudaV1AIKCertCycleIndex OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "High-order index of this AIK Certificate fragment. Index of an AIK Certificate chain update cycle that has occurred (bounded by the value of tudaV1AIKCertCycles)."
DEFVAL intentionally omitted - index object.
::= { tudaV1AIKCertEntry 1 }

tudaV1AIKCertInstanceIndex OBJECT-TYPE
SYNTAX      Integer32 (1..2147483647)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"Middle index of this AIK Certificate fragment.
Ordinal of this AIK Certificate in this chain, where the AIK
Certificate itself has an ordinal of '1' and higher ordinals
go "up" the certificate chain to the Root CA.

DEFVAL intentionally omitted - index object."
::= { tudaV1AIKCertEntry 2 }

tudaV1AIKCertFragmentIndex OBJECT-TYPE
SYNTAX      Integer32 (1..2147483647)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"Low-order index of this AIK Certificate fragment.
DEFVAL intentionally omitted - index object."
::= { tudaV1AIKCertEntry 3 }

tudaV1AIKCertData OBJECT-TYPE
SYNTAX      OCTET STRING (SIZE(0..1024))
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"A fragment of CBOR encoded AIK Certificate data."
DEFVAL      { "" }
::= { tudaV1AIKCertEntry 4 }

--
--  TSA Cert
--
tudaV1TSACert OBJECT IDENTIFIER ::= { tudaV1MIBObjects 3 }

tudaV1TSACertCycles OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"Count of TSA Certificate chain update cycles that have
occurred."
DEFVAL intentionally omitted - counter object.

::= { tudaV1TSACert 1 }

tudaV1TSACertTable OBJECT-TYPE
SYNTAX      SEQUENCE OF TudaV1TSACertEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION  "A table of fragments of TSA Certificate data."
 ::= { tudaV1TSACert 2 }

TudaV1TSACertEntry OBJECT-TYPE
SYNTAX      TudaV1TSACertEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION  "An entry for one fragment of TSA Certificate data."
INDEX       { tudaV1TSACertCycleIndex,
                   tudaV1TSACertInstanceIndex,
                   tudaV1TSACertFragmentIndex }
 ::= { tudaV1TSACertTable 1 }

TudaV1TSACertEntry ::= SEQUENCE {
   tudaV1TSACertCycleIndex         Integer32,
   tudaV1TSACertInstanceIndex      Integer32,
   tudaV1TSACertFragmentIndex      Integer32,
   tudaV1TSACertData               OCTET STRING
}

TudaV1TSACertCycleIndex OBJECT-TYPE
SYNTAX      Integer32 (1..2147483647)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION  "High-order index of this TSA Certificate fragment.  
Index of a TSA Certificate chain update cycle that has 
ocurred (bounded by the value of tudaV1TSACertCycles)."
DEFVAL intentionally omitted - index object.
 ::= { tudaV1TSACertEntry 1 }

TudaV1TSACertInstanceIndex OBJECT-TYPE
SYNTAX      Integer32 (1..2147483647)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION  "Middle index of this TSA Certificate fragment."
Ordinal of this TSA Certificate in this chain, where the TSA Certificate itself has an ordinal of ‘1’ and higher ordinals go *up* the certificate chain to the Root CA.

::= { tudaV1TSACertEntry 2 }

tudaV1TSACertFragmentIndex OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Low-order index of this TSA Certificate fragment.

DEFVAL intentionally omitted - index object."
::= { tudaV1TSACertEntry 3 }

tudaV1TSACertData OBJECT-TYPE
SYNTAX OCTET STRING (SIZE(0..1024))
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"A fragment of CBOR encoded TSA Certificate data."
DEFVAL { "" }
::= { tudaV1TSACertEntry 4 }

--
-- Sync Token
--

tudaV1SyncToken OBJECT IDENTIFIER ::= { tudaV1MIBObjects 4 }

tudaV1SyncTokenCycles OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of Sync Token update cycles that have occurred.

DEFVAL intentionally omitted - counter object."
::= { tudaV1SyncToken 1 }

tudaV1SyncTokenInstances OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of Sync Token instance entries that have
been recorded (some entries MAY have been pruned).

DEFVAL intentionally omitted - counter object."
::= { tudaV1SyncToken 2 }

**tudaV1SyncTokenTable** OBJECT-TYPE
SYNTAX SEQUENCE OF TudaV1SyncTokenEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "A table of fragments of Sync Token data."
::= { tudaV1SyncToken 3 }

**tudaV1SyncTokenEntry** OBJECT-TYPE
SYNTAX TudaV1SyncTokenEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "An entry for one fragment of Sync Token data."
INDEX { tudaV1SyncTokenCycleIndex,
tudaV1SyncTokenInstanceIndex,
tudaV1SyncTokenFragmentIndex }
::= { tudaV1SyncTokenTable 1 }

TudaV1SyncTokenEntry ::= SEQUENCE {
  tudaV1SyncTokenCycleIndex       Integer32,
tudaV1SyncTokenInstanceIndex    Integer32,
tudaV1SyncTokenFragmentIndex    Integer32,
tudaV1SyncTokenData             OCTET STRING
}

**tudaV1SyncTokenCycleIndex** OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "High-order index of this Sync Token fragment.
Index of a Sync Token update cycle that has occurred (bounded by the value of tudaV1SyncTokenCycles)."
DEFVAL intentionally omitted - index object."
::= { tudaV1SyncTokenEntry 1 }

**tudaV1SyncTokenInstanceIndex** OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION

"Middle index of this Sync Token fragment. Ordinal of this instance of Sync Token data (NOT bounded by the value of tudaV1SyncTokenInstances).

DEFVAL intentionally omitted - index object."

::= { tudaV1SyncTokenEntry 2 }

tudaV1SyncTokenFragmentIndex OBJECT-TYPE
SYNTAX      Integer32 (1..2147483647)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION

"Low-order index of this Sync Token fragment.

DEFVAL intentionally omitted - index object."

::= { tudaV1SyncTokenEntry 3 }

tudaV1SyncTokenData OBJECT-TYPE
SYNTAX      OCTET STRING (SIZE(0..1024))
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION

"A fragment of CBOR encoded Sync Token data."

DEFVAL      { "" }

::= { tudaV1SyncTokenEntry 4 }

--
-- Restriction Info
--
tudaV1Restrict OBJECT IDENTIFIER ::= { tudaV1MIBObjects 5 }

tudaV1RestrictCycles OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION

"Count of Restriction Info update cycles that have occurred.

DEFVAL intentionally omitted - counter object."

::= { tudaV1Restrict 1 }

tudaV1RestrictTable OBJECT-TYPE
SYNTAX      SEQUENCE OF TudaV1RestrictEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"A table of instances of Restriction Info data."
::= { tudaV1Restrict 2 }

tudaV1RestrictEntry OBJECT-TYPE
SYNTAX TudaV1RestrictEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "An entry for one instance of Restriction Info data."
INDEX { tudaV1RestrictCycleIndex }
::= { tudaV1RestrictTable 1 }

TudaV1RestrictEntry ::= SEQUENCE {
    tudaV1RestrictCycleIndex        Integer32,
    tudaV1RestrictData              OCTET STRING
}

tudaV1RestrictCycleIndex OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "Index of this Restriction Info entry.
Index of a Restriction Info update cycle that has occurred (bounded by the value of tudaV1RestrictCycles)."
DEFVAL intentionally omitted - index object."
::= { tudaV1RestrictEntry 1 }

tudaV1RestrictData OBJECT-TYPE
SYNTAX OCTET STRING (SIZE(0..1024))
MAX-ACCESS read-only
STATUS current
DESCRIPTION "An instance of CBOR encoded Restriction Info data."
DEFVAL { "" }
::= { tudaV1RestrictEntry 2 }

--
-- Measurement Log
--
tudaV1Measure OBJECT IDENTIFIER ::= { tudaV1MIBObjects 6 }

tudaV1MeasureCycles OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of Measurement Log update cycles that have occurred.
DEFVAL intentionally omitted - counter object."
::= { tudaV1Measure 1 }

tudaV1MeasureInstances OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of Measurement Log instance entries that have been recorded (some entries MAY have been pruned).
DEFVAL intentionally omitted - counter object."
::= { tudaV1Measure 2 }

tudaV1MeasureTable OBJECT-TYPE
SYNTAX SEQUENCE OF TudaV1MeasureEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"A table of instances of Measurement Log data."
::= { tudaV1MeasureTable 1 }

tudaV1MeasureEntry OBJECT-TYPE
SYNTAX TudaV1MeasureEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"An entry for one instance of Measurement Log data."
INDEX { tudaV1MeasureCycleIndex,
          tudaV1MeasureInstanceIndex }
::= { tudaV1MeasureTable 1 }

TudaV1MeasureEntry ::= SEQUENCE {
  tudaV1MeasureCycleIndex         Integer32,
  tudaV1MeasureInstanceIndex      Integer32,
  tudaV1MeasureData               OCTET STRING
}

tudaV1MeasureCycleIndex OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"High-order index of this Measurement Log entry.
Index of a Measurement Log update cycle that has
occurred (bounded by the value of tudaV1MeasureCycles).

DEFVAL intentionally omitted - index object."
::= { tudaV1MeasureEntry 1 }

tudaV1MeasureInstanceIndex OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Low-order index of this Measurement Log entry.
Ordinal of this instance of Measurement Log data
(NOT bounded by the value of tudaV1MeasureInstances).

DEFVAL intentionally omitted - index object."
::= { tudaV1MeasureEntry 2 }

tudaV1MeasureData OBJECT-TYPE
SYNTAX OCTET STRING (SIZE(0..1024))
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"A instance of CBOR encoded Measurement Log data."
DEFVAL { "" }
::= { tudaV1MeasureEntry 3 }

--
-- Verify Token
--
tudaV1VerifyToken OBJECT IDENTIFIER ::= { tudaV1MIBObjects 7 }

tudaV1VerifyTokenCycles OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of Verify Token update cycles that have
occurred.

DEFVAL intentionally omitted - counter object."
::= { tudaV1VerifyToken 1 }

tudaV1VerifyTokenTable OBJECT-TYPE
SYNTAX SEQUENCE OF TudaV1VerifyTokenEntry
MAX-ACCESS not-accessible
 STATUS   current
DESCRIPTION
  "A table of instances of Verify Token data."
 ::= { tudaV1VerifyToken 2 }

tudaV1VerifyTokenEntry OBJECT-TYPE
  SYNTAX   TudaV1VerifyTokenEntry
  MAX-ACCESS not-accessible
  STATUS   current
  DESCRIPTION
  "An entry for one instance of Verify Token data."
  INDEX    { tudaV1VerifyTokenCycleIndex }
 ::= { tudaV1VerifyTokenTable 1 }

TudaV1VerifyTokenEntry ::= SEQUENCE {
  tudaV1VerifyTokenCycleIndex     Integer32,
  tudaV1VerifyTokenData           OCTET STRING
}

tudaV1VerifyTokenCycleIndex OBJECT-TYPE
  SYNTAX   Integer32 (1..2147483647)
  MAX-ACCESS not-accessible
  STATUS   current
  DESCRIPTION
  "Index of this instance of Verify Token data."
  "Index of a Verify Token update cycle that has occurred (bounded by the value of tudaV1VerifyTokenCycles)."
  DEFVAL intentionally omitted - index object."
 ::= { tudaV1VerifyTokenEntry 1 }

tudaV1VerifyTokenData OBJECT-TYPE
  SYNTAX   OCTET STRING (SIZE(0..1024))
  MAX-ACCESS read-only
  STATUS   current
  DESCRIPTION
  "A instance of CBOR encoded Verify Token data."
  DEFVAL { "" }
 ::= { tudaV1VerifyTokenEntry 2 }

--
-- SWID Tag
--
tudaV1SWIDTag OBJECT IDENTIFIER ::= { tudaV1MIBObjects 8 }

tudaV1SWIDTagCycles OBJECT-TYPE
  SYNTAX   Counter32

MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of SWID Tag update cycles that have occurred.
DEFVAL intentionally omitted - counter object."
::= { tudaV1SWIDTag 1 }

tudaV1SWIDTagTable OBJECT-TYPE
SYNTAX SEQUENCE OF TudaV1SWIDTagEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"A table of fragments of SWID Tag data."
::= { tudaV1SWIDTagTable 1 }

tudaV1SWIDTagEntry OBJECT-TYPE
SYNTAX TudaV1SWIDTagEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"An entry for one fragment of SWID Tag data."
INDEX { tudaV1SWIDTagCycleIndex, tudaV1SWIDTagInstanceIndex, tudaV1SWIDTagFragmentIndex }
::= { tudaV1SWIDTagTable 1 }

TudaV1SWIDTagEntry ::= SEQUENCE {
  tudaV1SWIDTagCycleIndex         Integer32,
  tudaV1SWIDTagInstanceIndex      Integer32,
  tudaV1SWIDTagFragmentIndex      Integer32,
  tudaV1SWIDTagData               OCTET STRING
}

tudaV1SWIDTagCycleIndex OBJECT-TYPE
SYNTAX Integer32 (1..2147483647)
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"High-order index of this SWID Tag fragment.
Index of an SWID Tag update cycle that has occurred (bounded by the value of tudaV1SWIDTagCycles).
DEFVAL intentionally omitted - index object."
::= { tudaV1SWIDTagEntry 1 }

tudaV1SWIDTagInstanceIndex OBJECT-TYPE
SYNTAX      Integer32 (1..2147483647)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
  "Middle index of this SWID Tag fragment. Ordinal of this SWID Tag instance in this update cycle.
DEFVAL intentionally omitted - index object."
 ::= { tudaV1SWIDTagEntry 2 }

tudaV1SWIDTagFragmentIndex OBJECT-TYPE
SYNTAX      Integer32 (1..2147483647)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
  "Low-order index of this SWID Tag fragment.
DEFVAL intentionally omitted - index object."
 ::= { tudaV1SWIDTagEntry 3 }

tudaV1SWIDTagData OBJECT-TYPE
SYNTAX      OCTET STRING (SIZE(0..1024))
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION  
  "A fragment of CBOR encoded SWID Tag data."
DEFVAL      { "" }
 ::= { tudaV1SWIDTagEntry 4 }

--
-- Trap Cycles
--
tudaV1TrapV2Cycles NOTIFICATION-TYPE
OBJECTS  {
    tudaV1GeneralCycles,
    tudaV1AIKCertCycles,
    tudaV1TSACertCycles,
    tudaV1SyncTokenCycles,
    tudaV1SyncTokenInstances,
    tudaV1RestrictCycles,
    tudaV1MeasureCycles,
    tudaV1MeasureInstances,
    tudaV1VerifyTokenCycles,
    tudaV1SWIDTagCycles
  }
STATUS      current
DESCRIPTION
  "This trap is sent when the value of any cycle or instance
counter changes (i.e., one or more tables are updated).

Note: The value of sysUpTime in IETF MIB-II (RFC 1213) is always included in SNMPv2 traps, per RFC 3416.

::= { tudaV1MIBNotifications 1 }

--
-- Conformance Information
--
tudaV1Compliances OBJECT IDENTIFIER ::= { tudaV1MIBConformance 1 }
tudaV1ObjectGroups OBJECT IDENTIFIER ::= { tudaV1MIBConformance 2 }
tudaV1NotificationGroups OBJECT IDENTIFIER ::= { tudaV1MIBConformance 3 }

--
-- Compliance Statements
--
tudaV1BasicCompliance MODULE-COMPLIANCE
  STATUS current
  DESCRIPTION
    "An implementation that complies with this module MUST implement all of the objects defined in the mandatory group tudaV1BasicGroup."
  MODULE -- this module
  MANDATORY-GROUPS { tudaV1BasicGroup }
  GROUP tudaV1OptionalGroup
  DESCRIPTION
    "The optional TUDA MIB objects. An implementation MAY implement this group."
  GROUP tudaV1TrapGroup
  DESCRIPTION
    "The TUDA MIB traps. An implementation SHOULD implement this group."
  ::= { tudaV1Compliances 1 }

--
-- Compliance Groups
--
tudaV1BasicGroup OBJECT-GROUP
  OBJECTS {
    tudaV1GeneralCycles,
    tudaV1GeneralVersionInfo,
tudaV1SyncTokenCycles,
tudaV1SyncTokenInstances,
tudaV1SyncTokenData,
tudaV1RestrictTokenCycles,
tudaV1RestrictTokenData,
tudaV1VerifyTokenCycles,
tudaV1VerifyTokenData
}
STATUS current
DESCRIPTION
"The basic mandatory TUDA MIB objects."
::= { tudaV1ObjectGroups 1 }

tudaV1OptionalGroup OBJECT-GROUP
OBJECTS {
  tudaV1AIKCertCycles,
tudaV1AIKCertData,
  tudaV1TSACertCycles,
tudaV1TSACertData,
  tudaV1MeasureCycles,
tudaV1MeasureData,
  tudaV1MeasureInstances,
tudaV1MeasureData,
  tudaV1SWIDTagCycles,
tudaV1SWIDTagData
}
STATUS current
DESCRIPTION
"The optional TUDA MIB objects."
::= { tudaV1ObjectGroups 2 }

tudaV1TrapGroup NOTIFICATION-GROUP
NOTIFICATIONS { tudaV1TrapV2Cycles }
STATUS current
DESCRIPTION
"The recommended TUDA MIB traps - notifications."
::= { tudaV1NotificationGroups 1 }

END

<CODE ENDS>

Appendix C.  YANG Realization

<CODE BEGINS>
module TUDA-V1-ATTESTATION-MIB {

  prefix "tuda-v1";

import SNMP-FRAMEWORK-MIB { prefix "snmp-framework"; }
import yang-types { prefix "yang"; }

organization
"Fraunhofer SIT";

contact
"Andreas Fuchs
Fraunhofer Institute for Secure Information Technology
Email: andreas.fuchs@sit.fraunhofer.de

Henk Birkholz
Fraunhofer Institute for Secure Information Technology
Email: henk.birkholz@sit.fraunhofer.de

Ira E McDonald
High North Inc
Email: blueroofmusic@gmail.com

Carsten Bormann
Universitaet Bremen T2I
Email: cabo@tzi.org";

description
"The MIB module for monitoring of time-based unidirectional
attestation information from a network endpoint system,
based on the Trusted Computing Group TPM 1.2 definition.

Copyright (C) High North Inc (2017).";

revision "2017-10-30" {
  description
  "Fifth version, published as draft-birkholz-tuda-04.";
  reference
  "draft-birkholz-tuda-04";
}
revision "2017-01-09" {
  description
  "Fourth version, published as draft-birkholz-tuda-03.";
  reference
  "draft-birkholz-tuda-03";
}
revision "2016-07-08" {
  description
  "Third version, published as draft-birkholz-tuda-02.";
  reference
  "draft-birkholz-tuda-02";
}
revision "2016-03-21" {  
description  
"Second version, published as draft-birkholz-tuda-01.";  
reference  
"draft-birkholz-tuda-01";  
}  
revision "2015-10-18" {  
description  
"Initial version, published as draft-birkholz-tuda-00.";  
reference  
"draft-birkholz-tuda-00";  
}

container tudaV1General {  
description  
"TBD";  
leaf tudaV1GeneralCycles {  
type yang:counter32;  
config false;  
description  
"Count of TUDA update cycles that have occurred, i.e.,  
sum of all the individual group cycle counters.  
DEFVAL intentionally omitted - counter object.";  
}  
leaf tudaV1GeneralVersionInfo {  
type snmp-framework:SnmpAdminString {  
length "0..255";  
}  
config false;  
description  
"Version information for TUDA MIB, e.g., specific release  
version of TPM 1.2 base specification and release version  
of TPM 1.2 errata specification and manufacturer and model  
TPM module itself.";  
}  
}

container tudaV1AIKCert {  
description  
"TBD";  
leaf tudaV1AIKCertCycles {  
type yang:counter32;  
config false;  
description
"Count of AIK Certificate chain update cycles that have occurred.
DEFVAL intentionally omitted - counter object."
}

/* XXX table comments here XXX */
list tudaV1AIKCertEntry {
  key "tudaV1AIKCertCycleIndex tudaV1AIKCertInstanceIndex
       tudaV1AIKCertFragmentIndex";
  config false;
  description
       "An entry for one fragment of AIK Certificate data.";

  leaf tudaV1AIKCertCycleIndex {
    type int32 {
      range "1..2147483647";
    }
    config false;
    description
       "High-order index of this AIK Certificate fragment.
       Index of an AIK Certificate chain update cycle that has
       occurred (bounded by the value of tudaV1AIKCertCycles).
       DEFVAL intentionally omitted - index object.";
  }

  leaf tudaV1AIKCertInstanceIndex {
    type int32 {
      range "1..2147483647";
    }
    config false;
    description
       "Middle index of this AIK Certificate fragment.
       Ordinal of this AIK Certificate in this chain, where the AIK
       Certificate itself has an ordinal of '1' and higher ordinals
       go "up" the certificate chain to the Root CA.
       DEFVAL intentionally omitted - index object.";
  }

  leaf tudaV1AIKCertFragmentIndex {
    type int32 {
      range "1..2147483647";
    }
  }
}
config false;
description
"Low-order index of this AIK Certificate fragment.
DEFVAL intentionally omitted - index object.";
}
leaf tudaV1AIKCertData {
  type binary {
    length "0..1024";
  }
  config false;
description
"A fragment of CBOR encoded AIK Certificate data.";
} 
}
container tudaV1TSACert {
  description
"TBD";
leaf tudaV1TSACertCycles {
  type yang:counter32;
  config false;
description
"Count of TSA Certificate chain update cycles that have
occurred.
DEFVAL intentionally omitted - counter object.";
} /* XXX table comments here XXX */
list tudaV1TSACertEntry {
  key "tudaV1TSACertCycleIndex tudaV1TSACertInstanceIndex
tudaV1TSACertFragmentIndex";
  config false;
description
"An entry for one fragment of TSA Certificate data.";
leaf tudaV1TSACertCycleIndex {
  type int32 {
    range "1..2147483647";
}
leaf tudaV1TSACertInstanceIndex {
    type int32 {
        range "1..2147483647";
    }
    config false;
    description
    "Middle index of this TSA Certificate fragment.
    Ordinal of this TSA Certificate in this chain, where the TSA
    Certificate itself has an ordinal of '1' and higher ordinals
    go "up" the certificate chain to the Root CA.
    DEFVAL intentionally omitted - index object."
}

leaf tudaV1TSACertFragmentIndex {
    type int32 {
        range "1..2147483647";
    }
    config false;
    description
    "Low-order index of this TSA Certificate fragment.
    DEFVAL intentionally omitted - index object."
}

leaf tudaV1TSACertData { 
    type binary { 
        length "0..1024";
    }
    config false;
    description
    "A fragment of CBOR encoded TSA Certificate data.";
}

container tudaV1SyncToken {
    description

"TBD"

leaf tudaV1SyncTokenCycles {
  type yang:counter32;
  config false;
  description
  "Count of Sync Token update cycles that have occurred.
   DEFVAL intentionally omitted - counter object.";
}

leaf tudaV1SyncTokenInstances {
  type yang:counter32;
  config false;
  description
  "Count of Sync Token instance entries that have been recorded (some entries MAY have been pruned).
   DEFVAL intentionally omitted - counter object.";
}

list tudaV1SyncTokenEntry {
  key "tudaV1SyncTokenCycleIndex
         tudaV1SyncTokenInstanceIndex
         tudaV1SyncTokenFragmentIndex";
  config false;
  description
  "An entry for one fragment of Sync Token data.";

  leaf tudaV1SyncTokenCycleIndex {
    type int32 {
      range "1..2147483647";
    }
    config false;
    description
    "High-order index of this Sync Token fragment. Index of a Sync Token update cycle that has occurred (bounded by the value of tudaV1SyncTokenCycles).
     DEFVAL intentionally omitted - index object.";
  }

  leaf tudaV1SyncTokenInstanceIndex {
    type int32 {
      range "1..2147483647";
    }

leaf tudaV1SyncTokenFragmentIndex {
  type int32 {
    range "1..2147483647";
  }
  config false;
  description
  "Low-order index of this Sync Token fragment.
  DEFVAL intentionally omitted - index object.";
}

leaf tudaV1SyncTokenData {
  type binary {
    length "0..1024";
  }
  config false;
  description
  "A fragment of CBOR encoded Sync Token data.";
}

container tudaV1Restrict {
  description
  "TBD";

  leaf tudaV1RestrictCycles {
    type yang:counter32;
    config false;
    description
    "Count of Restriction Info update cycles that have occurred.
    DEFVAL intentionally omitted - counter object.";
  }

  /* XXX table comments here XXX */
list tudaV1RestrictEntry {
  key "tudaV1RestrictCycleIndex";
  config false;
  description
    "An entry for one instance of Restriction Info data.";
}
leaf tudaV1RestrictCycleIndex {
  type int32 {
    range "1..2147483647";
  }
  config false;
  description
    "Index of this Restriction Info entry. Index of a Restriction Info update cycle that has occurred (bounded by the value of tudaV1RestrictCycles). DEFVAL intentionally omitted - index object.";
}
leaf tudaV1RestrictData {
  type binary {
    length "0..1024";
  }
  config false;
  description
    "An instance of CBOR encoded Restriction Info data.";
}
}
container tudaV1Measure {
  description
    "TBD";
  leaf tudaV1MeasureCycles {
    type yang:counter32;
    config false;
    description
      "Count of Measurement Log update cycles that have occurred.
      DEFVAL intentionally omitted - counter object.";
  }
  leaf tudaV1MeasureInstances {
    type yang:counter32;
  }
}
config false;
description
"Count of Measurement Log instance entries that have
been recorded (some entries MAY have been pruned).

DEFVAL intentionally omitted - counter object.";
}

list tudaV1MeasureEntry {

key "tudaV1MeasureCycleIndex tudaV1MeasureInstanceIndex";
config false;
description
"An entry for one instance of Measurement Log data.";

leaf tudaV1MeasureCycleIndex {

type int32 {
    range "1..2147483647";
}
config false;
description
"High-order index of this Measurement Log entry.
Index of a Measurement Log update cycle that has
occurred (bounded by the value of tudaV1MeasureCycles).

DEFVAL intentionally omitted - index object.";
}

leaf tudaV1MeasureInstanceIndex {

type int32 {
    range "1..2147483647";
}
config false;
description
"Low-order index of this Measurement Log entry.
Ordinal of this instance of Measurement Log data
(NOT bounded by the value of tudaV1MeasureInstances).

DEFVAL intentionally omitted - index object.";
}

leaf tudaV1MeasureData {

type binary {
    length "0..1024";
}
config false;
description
"A instance of CBOR encoded Measurement Log data."
}
}
}

container tudaV1VerifyToken {
  description
    "TBD";

  leaf tudaV1VerifyTokenCycles {
    type yang:counter32;
    config false;
    description
      "Count of Verify Token update cycles that have occurred.
        DEFVAL intentionally omitted - counter object.";
  }

  /* XXX table comments here XXX */

  list tudaV1VerifyTokenEntry {
    key "tudaV1VerifyTokenCycleIndex";
    config false;
    description
      "An entry for one instance of Verify Token data.";

    leaf tudaV1VerifyTokenCycleIndex {
      type int32 {
        range "1..2147483647";
      }
      config false;
      description
        "Index of this instance of Verify Token data.
        Index of a Verify Token update cycle that has occurred (bounded by the value
        of tudaV1VerifyTokenCycles).
        DEFVAL intentionally omitted - index object.";
    }

    leaf tudaV1VerifyTokenData {
      type binary {
        length "0..1024";
      }
      config false;
    }

description
  "A instanc-V1-ATTESTATION-MIB.yang
 }
 }

container tudaV1SWIDTag {
  description
  "see CoSWID and YANG SIWD module for now"

  leaf tudaV1SWIDTagCycles {
    type yang:counter32;
    config false;
    description
      "Count of SWID Tag update cycles that have occurred."
      "DEFVAL intentionally omitted - counter object."
  }

  list tudaV1SWIDTagEntry {
    key "tudaV1SWIDTagCycleIndex tudaV1SWIDTagInstanceIndex
      tudaV1SWIDTagFragmentIndex";
    config false;
    description
      "An entry for one fragment of SWID Tag data."

    leaf tudaV1SWIDTagCycleIndex {
      type int32 {
        range "1..2147483647";
      }
      config false;
      description
        "High-order index of this SWID Tag fragment."
        "Index of an SWID Tag update cycle that has occurred (bounded by the value of tudaV1SWIDTagCycles)."
        "DEFVAL intentionally omitted - index object."
    }

    leaf tudaV1SWIDTagInstanceIndex {
      type int32 {
        range "1..2147483647";
      }
      config false;
      description
        "Middle index of this SWID Tag fragment."
Ordinal of this SWID Tag instance in this update cycle.

DEFVAL intentionally omitted - index object.

leaf tudaV1SWIDTagFragmentIndex {
  type int32 {
    range "1..2147483647";
  }
  config false;
  description
    "Low-order index of this SWID Tag fragment.

    DEFVAL intentionally omitted - index object."
}

leaf tudaV1SWIDTagData {
  type binary {
    length "0..1024";
  }
  config false;
  description
    "A fragment of CBOR encoded SWID Tag data."
}

notification tudaV1TrapV2Cycles {
  description
    "This trap is sent when the value of any cycle or instance
    counter changes (i.e., one or more tables are updated).

    Note: The value of sysUpTime in IETF MIB-II (RFC 1213) is
    always included in SNMPv2 traps, per RFC 3416."

  container tudaV1TrapV2Cycles-tudaV1GeneralCycles {
    description
      "TPD"

    leaf tudaV1GeneralCycles {
      type yang:counter32;
      description
        "Count of TUDA update cycles that have occurred, i.e.,
        sum of all the individual group cycle counters.

        DEFVAL intentionally omitted - counter object."
    }
  }
}
container tudaV1TrapV2Cycles-tudaV1AIKCertCycles {
  description  
  "TPD"
  leaf tudaV1AIKCertCycles {
    type yang:counter32;
    description  
      "Count of AIK Certificate chain update cycles that have  
       occurred."
    DEFVAL intentionally omitted - counter object."
  }
}

container tudaV1TrapV2Cycles-tudaV1TSACertCycles {
  description  
  "TPD"
  leaf tudaV1TSACertCycles {
    type yang:counter32;
    description  
      "Count of TSA Certificate chain update cycles that have  
       occurred."
    DEFVAL intentionally omitted - counter object."
  }
}

container tudaV1TrapV2Cycles-tudaV1SyncTokenCycles {
  description  
  "TPD"
  leaf tudaV1SyncTokenCycles {
    type yang:counter32;
    description  
      "Count of Sync Token update cycles that have  
       occurred."
    DEFVAL intentionally omitted - counter object."
  }
}

container tudaV1TrapV2Cycles-tudaV1SyncTokenInstances {
  description  
  "TPD"
  leaf tudaV1SyncTokenInstances {
    type yang:counter32;
    description  
      "Count of Sync Token instance entries that have  
       been recorded (some entries MAY have been pruned)."
  }
}
DEFVAL intentionally omitted - counter object.

} } }

container tudaV1TrapV2Cycles-tudaV1RestrictCycles {
    description
    "TPD"
    leaf tudaV1RestrictCycles {
        type yang:counter32;
        description
        "Count of Restriction Info update cycles that have occurred.
        DEFVAL intentionally omitted - counter object.";
    }
}

container tudaV1TrapV2Cycles-tudaV1MeasureCycles {
    description
    "TPD"
    leaf tudaV1MeasureCycles {
        type yang:counter32;
        description
        "Count of Measurement Log update cycles that have occurred.
        DEFVAL intentionally omitted - counter object.";
    }
}

container tudaV1TrapV2Cycles-tudaV1MeasureInstances {
    description
    "TPD"
    leaf tudaV1MeasureInstances {
        type yang:counter32;
        description
        "Count of Measurement Log instance entries that have been recorded (some entries MAY have been pruned).
        DEFVAL intentionally omitted - counter object.";
    }
}

container tudaV1TrapV2Cycles-tudaV1VerifyTokenCycles {
    description
    "TPD"
    leaf tudaV1VerifyTokenCycles {
        type yang:counter32;
        description
        "Count of Verify Token update cycles that have occurred.
        DEFVAL intentionally omitted - counter object.";
    }
}
Appendix D. Realization with TPM functions

D.1. TPM Functions

The following TPM structures, resources and functions are used within this approach. They are based upon the TPM specifications [TPM12] and [TPM2].

D.1.1. Tick-Session and Tick-Stamp

On every boot, the TPM initializes a new Tick-Session. Such a tick-session consists of a nonce that is randomly created upon each boot to identify the current boot-cycle - the phase between boot-time of the device and shutdown or power-off - and prevent replaying of old tick-session values. The TPM uses its internal entropy source that guarantees virtually no collisions of the nonce values between two of such boot cycles.

It further includes an internal timer that is being initialize to zero on each reboot. From this point on, the TPM increments this timer continuously based upon its internal secure clocking information until the device is powered down or set to sleep. By its hardware design, the TPM will detect attacks on any of those properties.
The TPM offers the function TPM_TickStampBlob, which allows the TPM to create a signature over the current tick-session and two externally provided input values. These input values are designed to serve as a nonce and as payload data to be included in a TickStampBlob: 

\[
\text{TickstampBlob := sig(TPM-key, currentTicks || nonce || externalData).}
\]

As a result, one is able to proof that at a certain point in time (relative to the tick-session) after the provisioning of a certain nonce, some certain externalData was known and provided to the TPM. If an approach however requires no input values or only one input value (such as the use in this document) the input values can be set to well-known value. The convention used within TCG specifications and within this document is to use twenty bytes of zero \(h'00000000000000000000000000000000'\) as well-known value.

D.1.2. Platform Configuration Registers (PCRs)

The TPM is a secure cryptoprocessor that provides the ability to store measurements and metrics about an endpoint’s configuration and state in a secure, tamper-proof environment. Each of these security relevant metrics can be stored in a volatile Platform Configuration Register (PCR) inside the TPM. These measurements can be conducted at any point in time, ranging from an initial BIOS boot-up sequence to measurements taken after hundreds of hours of uptime.

The initial measurement is triggered by the Platforms so-called pre-BIOS or ROM-code. It will conduct a measurement of the first loadable pieces of code; i.e.\ the BIOS. The BIOS will in turn measure its Option ROMs and the BootLoader, which measures the OS-Kernel, which in turn measures its applications. This describes a so-called measurement chain. This typically gets recorded in a so-called measurement log, such that the values of the PCRs can be reconstructed from the individual measurements for validation.

Via its PCRs, a TPM provides a Root of Trust that can, for example, support secure boot or remote attestation. The attestation of an endpoint’s identity or security posture is based on the content of an TPM’s PCRs (platform integrity measurements).

D.1.3. PCR restricted Keys

Every key inside the TPM can be restricted in such a way that it can only be used if a certain set of PCRs are in a predetermined state. For key creation the desired state for PCRs are defined via the PCRInfo field inside the keyInfo parameter. Whenever an operation using this key is performed, the TPM first checks whether the PCRs...
are in the correct state. Otherwise the operation is denied by the TPM.

D.1.4. CertifyInfo

The TPM offers a command to certify the properties of a key by means of a signature using another key. This includes especially the keyInfo which in turn includes the PCRInfo information used during key creation. This way, a third party can be assured about the fact that a key is only usable if the PCRs are in a certain state.

D.2. IE Generation Procedures for TPM 1.2

D.2.1. AIK and AIK Certificate

Attestations are based upon a cryptographic signature performed by the TPM using a so-called Attestation Identity Key (AIK). An AIK has the properties that it cannot be exported from a TPM and is used for attestations. Trust in the AIK is established by an X.509 Certificate emitted by a Certificate Authority. The AIK certificate is either provided directly or via a so-called PrivacyCA [AIK-Enrollment].

This element consists of the AIK certificate that includes the AIK’s public key used during verification as well as the certificate chain up to the Root CA for validation of the AIK certificate itself.

TUDA-Cert = [AIK-Cert, TSA-Cert]; maybe split into two for SNMP
AIK-Cert = Cert
TSA-Cert = Cert

Figure 2: TUDA-Cert element in CDDL

The TSA-Cert is a standard certificate of the TSA.

The AIK-Cert may be provisioned in a secure environment using standard means or it may follow the PrivacyCA protocols. Figure 3 gives a rough sketch of this protocol. See [AIK-Enrollment] for more information.

The X.509 Certificate is built from the AIK public key and the corresponding PKCS #7 certificate chain, as shown in Figure 3.

Required TPM functions:
create_AIK_Cert(...) = {
    AIK = TPM_MakeIdentity()
    IdReq = CollateIdentityRequest(AIK,EK)
    IdRes = Call(AIK-CA, IdReq)
    AIK-Cert = TPM_ActivateIdentity(AIK, IdRes)
}
/* Alternative */
create_AIK_Cert(...) = {
    AIK = TPM_CreateWrapKey(Identity)
    AIK-Cert = Call(AIK-CA, AIK.pubkey)
}

Figure 3: Creating the TUDA-Cert element

D.2.2. Synchronization Token

The reference for Attestations are the Tick-Sessions of the TPM. In order to put Attestations into relation with a Real Time Clock (RTC), it is necessary to provide a cryptographic synchronization between the tick session and the RTC. To do so, a synchronization protocol is run with a Time Stamp Authority (TSA) that consists of three steps:

- The TPM creates a TickStampBlob using the AIK
- This TickstampBlob is used as nonce to the Timestamp of the TSA
- Another TickStampBlob with the AIK is created using the TSA’s Timestamp a nonce

The first TickStampBlob is called "left" and the second "right" in a reference to their position on a time-axis.

These three elements, with the TSA’s certificate factored out, form the synchronization token
TUDA-Synctoken = [
    left: TickStampBlob-Output,
    timestamp: TimeStampToken,
    right: TickStampBlob-Output,
]

TimeStampToken = bytes ; RFC 3161

TickStampBlob-Output = [
    currentTicks: TPM-CURRENT-TICKS,
    sig: bytes,
]

TPM-CURRENT-TICKS = [
    currentTicks: uint
    ? { tickRate: uint
        tickNonce: TPM-NONCE
    }
]
; Note that TickStampBlob-Output "right" can omit the values for
;   tickRate and tickNonce since they are the same as in "left"

TPM-NONCE = bytes .size 20

Figure 4: TUDA-Sync element in CDDL

Required TPM functions:
dummyDigest = h’0000000000000000000000000000000000000000’
dummyNonce = dummyDigest

create_sync_token(AIKHandle, TSA) = {
   ts_left = TPM_TickStampBlob(
      keyHandle = AIK_Handle, /*TPM_KEY_HANDLE*/
      antiReplay = dummyNonce, /*TPM_NONCE*/
      digestToStamp = dummyDigest /*TPM_DIGEST*/)
	s = TSA_Timestamp(TSA, nonce = hash(ts_left))

   ts_right = TPM_TickStampBlob(
      keyHandle = AIK_Handle, /*TPM_KEY_HANDLE*/
      antiReplay = dummyNonce, /*TPM_NONCE*/
      digestToStamp = hash(ts) /*TPM_DIGEST*/)

   TUDA-SyncToken = [[ts_left.ticks, ts_left.sig], ts,
                    [ts_right.ticks.currentTicks, ts_right.sig]]
   /* Note: skip the nonce and tickRate field for ts_right.ticks */
}

Figure 5: Creating the Sync-Token element

D.2.3. RestrictionInfo

The attestation relies on the capability of the TPM to operate on restricted keys. Whenever the PCR values for the machine to be attested change, a new restricted key is created that can only be operated as long as the PCRs remain in their current state.

In order to prove to the Verifier that this restricted temporary key actually has these properties and also to provide the PCR value that it is restricted, the TPM command TPM_CertifyInfo is used. It creates a signed certificate using the AIK about the newly created restricted key.

This token is formed from the list of:

- PCR list,
- the newly created restricted public key, and
- the certificate.

TUDA-RestrictionInfo = [Composite,
                        restrictedKey_Pub: Pubkey,
                        CertifyInfo]
PCRSelection = bytes .size (2..4) ; used as bit string

Composite = [  
    bitmask: PCRSelection,  
    values: [*PCR-Hash],  
]

Pubkey = bytes ; may be extended to COSE pubkeys

CertifyInfo = [  
    TPM-CERTIFY-INFO,  
    sig: bytes,  
]

TPM-CERTIFY-INFO = [  
    ; we don't encode TPM-STRUCT-VER:  
    ; these are 4 bytes always equal to h’01010000’  
    keyUsage: uint, ; 4byte? 2byte?  
    keyFlags: bytes .size 4, ; 4byte  
    authDataUsage: uint, ; 1byte (enum)  
    algorithmParms: TPM-KEY-PARMS,  
    pubkeyDigest: Hash,  
    ; we don't encode TPM-NONCE data, which is 20 bytes, all zero  
    parentPCRStatus: bool,  
    ; no need to encode pcrinfoSize  
    pcrinfo: TPM-PCR-INFO, ; we have exactly one  
]

TPM-PCR-INFO = [  
    pcrSelection: PCRSelection; /* TPM_PCR_SELECTION */  
    digestAtRelease: PCR-Hash; /* TPM_COMPOSITE_HASH */  
    digestAtCreation: PCR-Hash; /* TPM_COMPOSITE_HASH */  
]

TPM-KEY-PARMS = [  
    ; algorithmID: uint, ; <= 4 bytes -- not encoded, constant for TPM1.2  
    encScheme: uint, ; <= 2 bytes  
    sigScheme: uint, ; <= 2 bytes  
    parms: TPM-RSA-KEY-PARMS,  
]

TPM-RSA-KEY-PARMS = [  
    ; "size of the RSA key in bits":  
    keyLength: uint  
    ; "number of prime factors used by this RSA key":  
    numPrimes: uint  
    ; "This SHALL be the size of the exponent":  
    exponentSize: null / uint / biguint

"If the key is using the default exponent then the exponentSize MUST be 0" -> we represent this case as null

Figure 6: TUDA-Key element in CDDL

Required TPM functions:

```plaintext
dummyDigest = h'0000000000000000000000000000000000000000'
dummyNonce = dummyDigest

create_Composite

create_restrictedKey_Pub(pcrsel) = {
  PCRInfo = {pcrSelection = pcrsel,
             digestAtRelease = hash(currentValues(pcrSelection))
             digestAtCreation = dummyDigest}
  /* PCRInfo is a TPM_PCR_INFO and thus also a TPM_KEY */
  wk = TPM_CreateWrapKey(keyInfo = PCRInfo)
  wk.keyInfo.pubKey
}

create_TPM-Certify-Info = {
  CertifyInfo = TPM_CertifyKey(
    certHandle = AIK, /* TPM_KEY_HANDLE */
    keyHandle = wk,  /* TPM_KEY_HANDLE */
    antiReply = dummyNonce) /* TPM_NONCE */
  CertifyInfo.strip()
  /* Remove those values that are not needed */
}
```

Figure 7: Creating the pubkey

D.2.4. Measurement Log

Similarly to regular attestations, the Verifier needs a way to reconstruct the PCRs’ values in order to estimate the trustworthiness of the device. As such, a list of those elements that were extended into the PCRs is reported. Note though that for certain environments, this step may be optional if a list of valid PCR configurations exists and no measurement log is required.
TUDA-Measurement-Log = [*PCR-Event]
PCR-Event = [
    type: PCR-Event-Type,
    pcr: uint,
    template-hash: PCR-Hash,
    filedata-hash: tagged-hash,
    pathname: text; called filename-hint in ima (non-ng)
]

PCR-Event-Type = &(
    bios: 0
    ima: 1
    ima-ng: 2
)

; might want to make use of COSE registry here
; however, that might never define a value for sha1
tagged-hash /= [sha1: 0, bytes .size 20]
tagged-hash /= [sha256: 1, bytes .size 32]

D.2.5. Implicit Attestation

The actual attestation is then based upon a TickStampBlob using the restricted temporary key that was certified in the steps above. The TPM-Tickstamp is executed and thereby provides evidence that at this point in time (with respect to the TPM internal tick-session) a certain configuration existed (namely the PCR values associated with the restricted key). Together with the synchronization token this tick-related timing can then be related to the real-time clock.

This element consists only of the TPM_TickStampBlock with no nonce.

TUDA-Verifytoken = TickStampBlob-Output

Figure 8: TUDA-Verify element in CDDL

Required TPM functions:

```c
| imp_att = TPM_TickStampBlob(
|     keyHandle = restrictedKey.Handle, /*TPM_KEY_HANDLE*/
|     antiReplay = dummyNonce, /*TPM_NONCE*/
|     digestToStamp = dummyDigest) /*TPM_DIGEST*/
| VerifyToken = imp_att
```

Figure 9: Creating the Verify Token
D.2.6. Attestation Verification Approach

The seven TUDA information elements transport the essential content that is required to enable verification of the attestation statement at the Verifier. The following listings illustrate the verification algorithm to be used at the Verifier in pseudocode. The pseudocode provided covers the entire verification task. If only a subset of TUDA elements changed (see Section 4.1), only the corresponding code listings need to be re-executed.

| TSA_pub = verifyCert(TSA-CA, Cert.TSA-Cert) |
| AIK_pub = verifyCert(AIK-CA, Cert.AIK-Cert) |

Figure 10: Verification of Certificates

```plaintext
ts_left = Synctoken.left
ts_right = Synctoken.right
/* Reconstruct ts_right’s omitted values; Alternatively assert == */
ts_right.currentTicks.tickRate = ts_left.currentTicks.tickRate
ts_right.currentTicks.tickNonce = ts_left.currentTicks.tickNonce
ticks_left = ts_left.currentTicks
ticks_right = ts_right.currentTicks
/* Verify Signatures */
verifySig(AIK_pub, dummyNonce || dummyDigest || ticks_left)
verifySig(TSA_pub, hash(ts_left) || timestamp.time)
verifySig(AIK_pub, dummyNonce || hash(timestamp) || ticks_right)
delta_left = timestamp.time -
    ticks_left.currentTicks * ticks_left.tickRate / 1000

delta_right = timestamp.time -
    ticks_right.currentTicks * ticks_right.tickRate / 1000
```

Figure 11: Verification of Synchronization Token
compositeHash = hash_init()
for value in Composite.values:
    hash_update(compositeHash, value)
compositeHash = hash_finish(compositeHash)
certInfo = reconstruct_static(TPM-CERTIFY-INFO)
assert(Composite.bitmask == ExpectedPCRBitmask)
assert(certInfo.pcrinfo.PCRSelection == Composite.bitmask)
assert(certInfo.pcrinfo.digestAtRelease == compositeHash)
assert(certInfo.pubkeyDigest == hash(restrictedKey_Pub))
verifySig(AIK_pub, dummyNonce || certInfo)

Figure 12: Verification of Restriction Info

for event in Measurement-Log:
    if event.pcr not in ExpectedPCRBitmask:
        continue
    if event.type == BIOS:
        assert_whitelist-bios(event.pcr, event.template-hash)
    if event.type == ima:
        assert(event.pcr == 10)
        assert(event.pathname, event.filedata-hash)
        assert(event.template-hash ==
               hash(event.pathname || event.filedata-hash))
    if event.type == ima-ng:
        assert(event.pcr == 10)
        assert_whitelist-ng(event.pathname, event.filedata-hash)
        assert(event.template-hash ==
               hash(event.pathname || event.filedata-hash))
    virtPCR[event.pcr] = hash_extend(virtPCR[event.pcr],
                                     event.template-hash)

for pcr in ExpectedPCRBitmask:
    assert(virtPCR[pcr] == Composite.values[i++])

Figure 13: Verification of Measurement Log
ts = Verifytoken

/* Reconstruct ts’s omitted values; Alternatively assert == */

\[\text{ts.currentTicks.tickRate} = \text{ts_left.currentTicks.tickRate}\]
\[\text{ts.currentTicks.tickNonce} = \text{ts_left.currentTicks.tickNonce}\]

verifySig(restrictedKey_pub, dummyNonce || dummyDigest || ts)

ticks = ts.currentTicks

ticks = ts.currentTicks

time_left = delta_right + ticks.currentTicks * ticks.tickRate / 1000

time_right = delta_left + ticks.currentTicks * ticks.tickRate / 1000

[time_left, time_right]

Figure 14: Verification of Attestation Token

D.3. IE Generation Procedures for TPM 2.0

The pseudo code below includes general operations that are conducted as specific TPM commands:

- hash() : description TBD
- sig() : description TBD
- X.509-Certificate() : description TBD

These represent the output structure of that command in the form of a byte string value.

D.3.1. AIK and AIK Certificate

Attestations are based upon a cryptographic signature performed by the TPM using a so-called Attestation Identity Key (AIK). An AIK has the properties that it cannot be exported from a TPM and is used for attestations. Trust in the AIK is established by an X.509 Certificate emitted by a Certificate Authority. The AIK certificate is either provided directly or via a so-called PrivacyCA [AIK-Enrollment].

This element consists of the AIK certificate that includes the AIK’s public key used during verification as well as the certificate chain up to the Root CA for validation of the AIK certificate itself.
TUDA-Cert = [AIK-Cert, TSA-Cert]; maybe split into two for SNMP
AIK-Certificate = X.509-Certificate(AIK-Key, Restricted-Flag)
TSA-Certificate = X.509-Certificate(TSA-Key, TSA-Flag)

Figure 15: TUDA-Cert element for TPM 2.0

D.3.2. Synchronization Token

The synchronization token uses a different TPM command, TPM2
GetTime() instead of TPM TickStampBlob(). The TPM2 GetTime() command
contains the clock and time information of the TPM. The clock
information is the equivalent of TUDA v1’s tickSession information.

TUDA-SyncToken = [
    left_GetTime = sig(AIK-Key,
        TimeInfo = [
            time,
            resetCount,
            restartCount
        ],
    ),
    middle_TimeStamp = sig(TSA-Key,
        hash(left_TickStampBlob),
        UTC-localtime
    ),
    right_TickStampBlob = sig(AIK-Key,
        hash(middle_TimeStamp),
        TimeInfo = [
            time,
            resetCount,
            restartCount
        ]
    )
]

Figure 16: TUDA-Sync element for TPM 2.0

D.3.3. Measurement Log

The creation procedure is identical to Appendix D.2.4.

Measurement-Log = [
    * [ EventName,
        PCR-Num,
        Event-Hash ]
]

Figure 17: TUDA-Log element for TPM 2.0
D.3.4. Explicit time-based Attestation

The TUDA attestation token consists of the result of TPM2_Quote() or a set of TPM2_PCR_READ followed by a TPM2_GetSessionAuditDigest. It proves that -- at a certain point-in-time with respect to the TPM’s internal clock -- a certain configuration of PCRs was present, as denoted in the keys restriction information.

TUDA-AttestationToken = TUDA-AttestationToken_quote / TUDA-AttestationToken_audit

TUDA-AttestationToken_quote = sig(AIK-Key,
   TimeInfo = [time, resetCount, restartCount],
   PCR-Selection = [* PCR],
   PCR-Digest := PCRDigest)

TUDA-AttestationToken_audit = sig(AIK-key,
   TimeInfo = [time, resetCount, restartCount],
   Session-Digest := PCRDigest)

Figure 18: TUDA-Attest element for TPM 2.0

D.3.5. Sync Proof

In order to proof to the Verifier that the TPM’s clock was not ‘fast-forwarded’ the result of a TPM2_GetTime() is sent after the TUDA-AttestationToken.

TUDA-SyncProof = sig(AIK-Key,
   TimeInfo = [time, resetCount, restartCount])

Figure 19: TUDA-Proof element for TPM 2.0
Acknowledgements

Authors’ Addresses

Andreas Fuchs
Fraunhofer Institute for Secure Information Technology
Rheinstrasse 75
Darmstadt  64295
Germany

Email: andreas.fuchs@sit.fraunhofer.de

Henk Birkholz
Fraunhofer Institute for Secure Information Technology
Rheinstrasse 75
Darmstadt  64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Ira E McDonald
High North Inc
PO Box 221
Grand Marais  49839
US

Email: blueroofmusic@gmail.com

Carsten Bormann
Universitaet Bremen T2I
Bibliothekstr. 1
Bremen  D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org
The Entity Attestation Token (EAT)
draft-mandyam-rats-eat-00

Abstract

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

Contributing

TBD

Status of This Memo

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

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

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

This Internet-Draft will expire on September 25, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 3
   1.1. Entity Overview .................................. 4
   1.2. Use of CBOR and COSE ............................. 5
   1.3. EAT Operating Models ............................. 5
   1.4. What is Not Standardized ........................... 6
       1.4.1. Transmission Protocol ......................... 6
       1.4.2. Signing Scheme ................................ 7
2. Terminology .............................................. 7
3. The Claims ............................................... 8
   3.1. Universal Entity ID (UEID) Claim ................. 8
   3.2. Origination (origination) Claims ................. 11
   3.3. OEM identification by IEEE OUI ...................... 11
   3.4. Security Level (seclevel) Claim ................. 12
   3.5. Nonce (nonce) Claim ................................ 13
   3.6. Secure Boot and Debug Enable State Claims ...... 13
       3.6.1. Secure Boot Enabled (secbootenabled) Claim .. 13
       3.6.2. Debug Disabled (debugdisabled) Claim .......... 13
       3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim ........................................... 13
       3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim 13
       3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim ............................. 14
   3.7. Location (loc) Claim ............................... 14
       3.7.1. lat (latitude) claim .......................... 14
       3.7.2. long (longitude) claim ........................ 14
       3.7.3. alt (altitude) claim ........................... 14
       3.7.4. acc (accuracy) claim ........................... 14
       3.7.5. altacc (altitude accuracy) claim .............. 15
       3.7.6. heading claim .................................. 15
       3.7.7. speed claim .................................... 15
   3.8. ts (timestamp) claim ................................ 15
   3.9. age claim ........................................... 15
   3.10. uptime claim ........................................ 15
   3.11. The submods Claim .................................. 16
       3.11.1. The submod_name Claim ......................... 16
       3.11.2. Nested EATs, the eat Claim ................. 16
1. Introduction

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

Remote attestation is a fundamental service that can 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)
Internet-Draft                     EAT                        March 2019

- Proof of the make and model of the device processor, particularly for security oriented chips
- Measurement of the software (SW) running on the device
- Configuration and state of the device
- Environmental characteristics of the device such as its GPS location

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

1.1. Entity Overview

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

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

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

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

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

1.3. EAT Operating Models

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

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

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

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

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

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

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

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

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

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

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

1.4. What is Not Standardized

1.4.1. Transmission Protocol

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

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

1.4.2. Signing Scheme

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

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

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

2. Terminology

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

This document reuses terminology from JWT [RFC7519], COSE [RFC8152], and CWT [RFC8392].

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

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

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

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

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

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

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

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

3. The Claims

3.1. Universal Entity ID (UEID) Claim

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

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

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

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

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

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

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

When the entity constructs the UEID, the first byte is a type and the following bytes the ID for that type. Several types are allowed to accommodate different industries and different manufacturing processes and to give options to avoid paying fees for certain types of manufacturer registrations.
<table>
<thead>
<tr>
<th>Type Byte</th>
<th>Type Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>GUID</td>
<td>This is a 128 to 256 bit random number generated once and stored in the device. The GUID may be constructed from various identifiers on the device using a hash function or it may be just the raw random number.</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 six 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

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

- UEIDs types may vary freely from one manufacturer to the next.
New types of UEIDs may be created. For example a type 0x04 UEID may be created based on some other manufacturer registration scheme.

Device manufacturers are allowed to change from one type of UEID to another anytime they want. For example they may find they can optimize their manufacturing by switching from type 0x01 to type 0x02 or vice versa. The main requirement on the manufacturer is that UEIDs be universally unique.

3.2. Origination (origination) Claims

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

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

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

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

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

3.3. OEM identification by IEEE OUI

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

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

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

3.4. Security Level (seclevel) Claim

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

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

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

1 - Unrestricted There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise the EAT provides no meaningful security assurances.

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

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

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

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

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

3.6. Secure Boot and Debug Enable State Claims

3.6.1. Secure Boot Enabled (secbootenabled) Claim

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

3.6.2. Debug Disabled (debugdisabled) Claim

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

3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim

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

3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim

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

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

3.7. Location (loc) Claim

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

3.7.1. lat (latitude) claim

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

3.7.2. long (longitude) claim

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

3.7.3. alt (altitude) claim

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

3.7.4. acc (accuracy) claim

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

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

3.7.6. heading claim

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

3.7.7. speed claim

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

3.8. ts (timestamp) claim

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

3.9. age claim

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

3.10. uptime claim

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

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

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

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

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

3.11.1. The submod_name Claim

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

3.11.2. Nested EATs, the eat Claim

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

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

4. CBOR Interoperability

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

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

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

4.2. String Encoding (major type 2 and 3)

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

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

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

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

4.4. Date and Time

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

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

4.5. URIs

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

4.6. Floating Point

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

4.7. Other types

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

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

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

5.1.1. Claims Registered by This Document

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

TODO: add the rest of the claims in here

5.2. EAT CBOR Tag Registration

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

5.2.1. Tag Registered by This Document

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

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

6.1. UEID Privacy Considerations

A UEID is usually not privacy preserving. Any set of relying parties that receives tokens that happen to be from a single device will be able to know the tokens are all from the same device and be able to track the device. Thus, in many usage situations UEID violates governmental privacy regulation. In other usage situations UEID will not be allowed for certain products like browsers that give privacy for the end user. It will often be the case that tokens will not have a UEID for these reasons.

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

- The device obtains explicit permission from the user of the device to use the UEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID.

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

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

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

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

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Examples

A.1. Very Simple EAT

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

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

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

```json
{
  / nonce /                 11:h'948f8860d13a463e8e' ,
  / UEID /                   8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f' ,
  / secbootenabled /        13:true ,
  / debugpermanentdisable / 15:true ,
  / ts /                    21:1526542894 ,
  / seclevel /              10:3 , / secure restricted OS /
  / submods / 30:
    [   1st submod, an Android Application / {   
      / submod_name / 30:'Android App "Foo"' ,
      / seclevel /    10:1 , / unrestricted /
      / app data /   -70000:'text string'
    },

    2nd submod, A nested EAT from a secure element / {   
      / submod_name / 30:'Secure Element EAT' ,
      / eat /         31:71( 18{
        / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]
      })
    }

    3rd submod, information about Linux Android / {   
      / submod_name / 30:'Linux Android' ,
      / seclevel /    10:1 , / unrestricted /
      / custom - release / -80000:'8.0.0' ,
      / custom - version / -80001:'4.9.51+'
    }
  }
}
```
Authors’ Addresses

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

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

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

Jeremy O’Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough GU14 7LS
United Kingdom
Phone: +44 1252 363189
EMail: jodonogh@qti.qualcomm.com
Abstract

Token binding allows HTTP servers to bind bearer tokens to TLS connections. In order to do this, clients or user agents must prove possession of a private key. However, proof-of-possession of a private key becomes truly meaningful to a server when accompanied by an attestation statement. This specification describes extensions to the existing token binding protocol to allow for attestation statements to be sent along with the related token binding messages.

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 July 28, 2019.

Copyright Notice

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

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

Mandyam, et al. Expires July 28, 2019
1. Introduction

[RFC8471] and [RFC8472] describe a framework whereby servers can leverage cryptographically-bound authentication tokens in part to create uniquely-identifiable TLS bindings that can span multiple connections between a client and a server. Once the use of token binding is negotiated as part of the TLS handshake, an application layer message (the Token Binding message) may be sent from the client to the relying party whose primary purpose is to encapsulate a signature over a value associated with the current TLS session. The payload used for the signature is the token binding public key (see [RFC8471]). Use of the token binding public key allows for generation of the attestation signature once over the lifetime of the public key.

Proof-of-possession of a private key is useful to a relying party, but the associated signature in the Token Binding message does not provide an indication as to how the private key is stored and in what kind of environment the associated cryptographic operation takes place. This information may be required by a relying party in order
to satisfy requirements regarding client platform integrity. Therefore, attestations are sometimes required by relying parties in order for them to accept signatures from clients. As per the definition in [I-D.birkholz-tuda], "remote attestation describes the attempt to determine the integrity and trustworthiness of an endpoint -- the attessee -- over a network to another endpoint -- the verifier -- without direct access." Attestation statements are therefore widely used in any server verification operation that leverages client cryptography.

TLS token binding can therefore be enhanced with remote attestation statements. The attestation statement can be used to augment Token Binding message. This could be used by a relying party for several different purpose, including (1) to determine whether to accept token binding messages from the associated client, or (2) require an additional mechanism for binding the TLS connection to an authentication operation by the client.

2. Attestation Enhancement to TLS Token Binding Message

The attestation statement can be processed ‘in-band’ as part of the Token Binding Message itself. This document leverages the TokenBinding.extensions field of the Token Binding Message as described in Section 3.4 of [RFC8471], where the extension data conforms to the guidelines of Section 6.3 of the same document. The value of the extension, as required by this same section, is assigned per attestation type. The extension data takes the form of a CBOR (compact binary object representation) Data Definition Language construct, i.e. CDDL.

```
extension_data = {attestation}
attestation = (  
    attestation_type:  tstr,
    attestation_data:  bstr,
)
```

The attestation data is determined according to the attestation type. In this document, the following types are defined: "KeyStore" (where the corresponding attestation data defined in [Keystore]) and "TPMv2" (where the corresponding attestation data defined in [TPMv2]). Additional attestation types may be accepted by the token binding implementation (for instance, see Section 8 of [webauthn]).

The attestation data will likely include a signature over a challenge (depending on the attestation type). The challenge can be used to prevent replay of the attestation. However since the attestation is
itself part of the token binding message (which has its own anti-replay protection mechanism), the attestation signature need only be generated over a known payload associated with the TLS token binding session - the token binding public key. As a result, the token binding client only needs to send the attestation once during the lifetime of the token binding public key. In other words, if an attestation is included in the token binding message, it should only be sent in the initial token binding message following the creation of the token binding key pair.

2.1. KeyStore Attestation

KeyStore attestation is relevant to the Android operating system. The Android Keystore mechanism allows for an application (such as a browser implementing the Token Binding stack) to create a key pair, export the public key, and protect the private key in a hardware-backed keystore. The Android Keystore can then be used to verify a keypair using the Keystore Attestation mechanism, which involves signing a payload according to a public key that chains to a root certificate signed by an attestation root key that is specific to the device manufacturer.

The octet value of the token binding extension that serves as identification for the Keystore attestation type is requested to be 0.

KeyStore attestation provides a signature over a payload generated by the application. The payload is a SHA-256 hash of the token binding public key corresponding to the current TLS connection (see Section 3.3 of [RFC8471]). Then the attestation takes the form of a signature, a signature-generation algorithmic identifier corresponding to the COSE algorithm registry ([cose_iana]), and a chain of DER-encoded x.509 certificates:

\[
\text{attestation\_data} = \{
\text{alg}: \text{int},
\text{sig}: \text{bytes},
\text{x5c}: [\text{credCert}: \text{bytes}, *\text{caCert}: \text{bytes}]
\}
\]

2.1.1. Verification Procedures

The steps at the server for verifying a Token Binding KeyStore Attestation are:
Retrieve token binding public key for the current TLS connection, and compute its SHA-256 hash.

- Verify that attestation_data is in the expected CBOR format.
- Parse the first certificate listed in x5c and extract the public key and challenge. If the challenge does not match the SHA-256 hash of the token binding public key then the attestation is invalid.
- If the challenge matches the expected hash of the token binding public key, verify the sig with respect to the extracted public key and algorithm from the previous step.
- Verify the rest of the certificate chain up to the root. The root certificate must match the expected root for the device.

2.2. TPMv2 Attestation

Version 2 of the Trusted Computing Group’s Trusted Platform Module (TPM) specification provides for an attestation generated within the context of a TPM. The attestation then is defined as

```plaintext
attestation_data = (  
  alg: int,  
  tpmt_sig: bytes,  
  tpms_attest: bytes,  
  x5c: [credCert: bytes, *(caCert: bytes)]  
)
```

The `tpmt_sig` is generated over a `tpms_attest` structure signed with respect to the certificate chain provided in the `x5c` array, and the algorithmic identifier corresponding to the COSE algorithm registry ([cose_iana]). It is derived from the TPMT_SIGNATURE data structure defined in Section 11.3.4 of [TPMv2]. `tpms_attest` is derived from the TPMS_ATTEST data structure in Section 10.2.8 of [TPMv2], specifically with the `extraData` field being set to a SHA-256 hash of the token binding public key.

The octet value of the token binding extension that serves as identification for the TPMv2 attestation type is requested to be 1.
2.2.1. Verification Procedures

The steps for verifying a Token Binding TPMv2 Attestation are:

- Extract the token binding public key for the current TLS connection.
- Verify that attestation_data is in the expected CBOR format.
- Parse the first certificate listed in x5c and extract the public key.
- Verify the tpms_attest structure, which includes
  - Verify that the type field is set to TPM_ST_ATTEST_CERTIFY.
  - Verify that extraData is equivalent to the SHA-256 hash of the token binding public key for the current TLS connection.
  - Verify that magic is set to the expected TPM_GENERATED_VALUE for the expected command sequence used to generate the attestation.
  - Verification of additional TPMS_ATTEST data fields is optional.
- Verify tpmt_sig with respect to the public key provided in the first certificate in x5c, using the algorithm as specified in the sigAlg field (see Sections 11.3.4, 11.2.1.5 and 9.29 of [TPMv2]).

3. Extension Support Negotiation

Even if the client supports a Token Binding extension, it may not be desirable to send the extension if the server does not support it. The benefits of client-suppression of an extension could include saving of bits "over the wire" or simplified processing of the Token Binding message at the server. Currently, extension support is not communicated as part of the Token Binding extensions to TLS (see [RFC8472]).

It is proposed that the Client and Server Hello extensions defined in Sections 3 and 4 of [RFC8472] be extended so that endpoints can communicate their support for specific TokenBinding.extensions. With reference to Section 3, it is recommended that the "token_binding" TLS extension be augmented by the client to include supported TokenBinding.extensions as follows:
enum {
    attestation(0), (255)
} TokenBindingExtensions;

struct {
    TB_ProtocolVersion token_binding_version;
    TokenBindingKeyParameters key_parameters_list<1..2^8-1>;
    TokenBindingExtensions supported_extensions_list<1..2^8-1>
} TokenBindingParameters;

The "supported_extensions_list" contains the list of identifiers of all token binding message extensions supported by the client. A server supporting token binding extensions will respond in the server hello with an appropriate "token_binding" extension that includes a "supported_extensions_list". This list must be a subset of the the extensions provided in the client hello.

Since a TLS extension cannot itself be extended, the "token_binding" TLS extension cannot be reused. Therefore it is proposed that a new TLS extension be defined - "token_binding_with_extensions". This TLS extension codepoint is identical to the existing "token_binding" extension except for the additional data structures defined above.

3.1. Negotiating Token Binding Protocol Extensions

The negotiation described in Section 4 of [RFC8472] still applies, except now the "token_binding_with_extensions" codepoint would be used if the client supports any token binding extension. In addition, a client can receive a "supported_extensions_list" from the server as part of the server hello. The client must terminate the handshake if the "supported_extensions_list" received from the server is not a subset of the "supported_extensions_list" sent by the client in the client hello. If the server hello list of supported extensions is a subset of the client supported extensions, then the client must only send those extensions specified in the server hello in the Token Binding protocol. If the server hello does not include a "supported_extensions_list", then the client must not send any extensions along with the Token Binding Message.

4. Example - Platform Attestation for Anomaly Detection

An example of where a platform-based attestation is useful can be for remote attestation based on client traffic anomaly detection. Many network infrastructure deployments employ network traffic monitors for anomalous pattern detection. Examples of anomalous patterns detectable in the TLS handshake could be unexpected cipher suite negotiation for a given source/destination pairing. In this case, it
may be desirable for a client-enhanced attestation reflecting for instance that an expected offered cipher suite in the client hello message is present or the originating browser integrity is intact (e.g. through a hash over the browser application package). If the network traffic monitor can interpret the attestation included in the token binding message, then it can verify the attestation and potentially emit alerts based on an unexpected attestation.

5. IANA Considerations

This memo includes the following requests to IANA.

5.1. TLS Extensions Registry

This document proposes an update of the TLS "ExtensionType Values" registry. The following addition to the registry is requested:

Value: TBD
Extension name: token_binding_with_extensions
Reference: this document
Recommended: Yes

5.2. Token Binding Extensions for Attestation

This document proposes two extensions conformant with Section 6.3 of [RFC8471], with the following specifics:

Android Keystore Attestation:
- Value: 0
- Description: Android Keystore Attestation
- Specification: This document

TPM v2 Attestation:
- Value: 1
- Description: TPMv2 Attestation
- Specification: This document
6. Security and Privacy Considerations

The security and privacy considerations provided in Section 7 of [RFC8471] are applicable to the attestation extensions proposed in this document. Additional considerations are provided in this section.

6.1. Attestation Privacy Considerations

The root signing key for the certificate chain used in verifying an attestation can be unique to the device. As a result, this can be used to track a device and/or end user. This potential privacy issue can be mitigated by the use of batch keys as an alternative to unique keys, or by generation of origin-specific attestation keys.

The attestation data may also contain device-specific identifiers, or information that can be used to fingerprint a device. Sensitive information can be excluded from the attestation data when this is a concern.

7. Acknowledgments

Thanks to Andrei Popov for his detailed review and recommendations.

8. References

8.1. Normative References

[cose_iана]
Internet Assigned Numbers Authority, "COSE Algorithms", <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>.

[I-D.greevenbosch-appsawg-cbor-cddl]

[Keystore]
8.2. Informative References

[I-D.birkholz-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann,
"Time-Based Uni-Directional Attestation", draft-birkholz-tuda-02 (work in progress), July 2016.

Authors' Addresses

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California  92121
USA
Phone: +1 858 651 7200
Email: mandyam@qti.qualcomm.com

Laurence Lundblade
Security Theory LLC
Email: lgl@island-resort.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
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)"

Richardson, et al. Expires April 8, 2020 [Page 3]
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.
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
o 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:

o Device Manufacturer

o Attester ("device under attestation")

o Verifier (Network Management Station)

o "Explicit Attestation" is the TCG term for a static (platform) attestation

o "Implicit Attestation" is the TCG term for a session attestation

o Reference Integrity Measurements (RIM), which are signed my device manufacturer and integrated into firmware.

o Quotes: measured values (having been signed), and RIMs
Reference Integrity Values (RIV)

- devices have a Initial Attestation Key (IAK), which is provisioned at the same time as the IDevID [ieee802-1AR]

- PCR - Platform Configuration Registry (deals with hash chains)

The TCG document builds upon a number of IETF technologies: SNMP (Attestation MIB), YANG, XML, JSON, CBOR, NETCONF, RESTCONF, CoAP, TLS and SSH. The TCG document leverages the 802.1AR IDevID and LDevID processes.

6.2. Android Keystore system

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

The uses described in section Section 5.7 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 a 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 provide 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 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
    .setKeySize(2048)
    .setDigests(KeyProperties.DIGEST_NONE, KeyProperties.DIGEST_SHA256)
    .setBlockModes(KeyProperties.BLOCK_MODE_CBC, KeyProperties.BLOCK_MODE_GCM)
    .setEncryptionPaddings(KeyProperties.ENCRIPTION_PADDING_RSA_PKCS1, KeyProperties.ENCRIPTION_PADDING_RSA_OAEP)
    .setSignaturePaddings(KeyProperties.SIGNATURE_PADDING_RSA_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
    .setKeySize(2048)
    .setDigests(KeyProperties.DIGEST_NONE, KeyProperties.DIGEST_SHA256, KeyProperties.DIGEST_SHA384, KeyProperties.DIGEST_SHA512)
    .setBlockModes(KeyProperties.BLOCK_MODE_CBC, KeyProperties.BLOCK_MODE_CTR, KeyProperties.BLOCK_MODE_GCM)
    .setEncryptionPaddings(KeyProperties.ENCRIPTION_PADDING_RSA_PKCS1, KeyProperties.ENCRIPTION_PADDING_RSA_OAEP)
    .setSignaturePaddings(KeyProperties.SIGNATURE_PADDING_RSA_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, idAttestationFlags);

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 3E 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 {
429 341:   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
  :      '1100'B
  :    }
  :  }
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
475   1:      INTEGER 3 -- keymasterVersion
478   1:      ENUMERATED 1 -- keymasterSecurityLevel (TrustedEn
481   9:      OCTET STRING 'challenge' -- attestationChallenge
492  0:      OCTET STRING -- reserved
  :    Error: Object has zero length.
494 44:    SEQUENCE { -- softwareEnforced
496  8:      [701] {
500  6:        INTEGER 01 64 47 2A 4B 64
  :    }
508 28:      [709] {
512 26:        OCTET STRING, encapsulates {
514 24:          SEQUENCE { -- AttestationApplicationId
518 18:          SET {
520 13:            OCTET STRING 'AndroidSystem' -- package_na
524 12:            INTEGER 1 -- version
  :    }
  :  }
535  1:      INTEGER 0 -- KeyPurpose.ENCRYPT
538  0:      INTEGER 1 -- KeyPurpose.DECRYPT
541  1:      INTEGER 2 -- KeyPurpose.SIGN
544  1:      INTEGER 3 -- KeyPurpose.VERIFY
  :    }
547  1:      INTEGER 0 -- KeyPurpose.ENCRYPT
550  1:      INTEGER 1 -- KeyPurpose.DECRYPT
553  1:      INTEGER 2 -- KeyPurpose.SIGN
556  1:      INTEGER 3 -- KeyPurpose.VERIFY
  :    }
559  3:      [2] {
562  1:        INTEGER 1 -- Algorithm.RSA
  :    }
564  4:      [3] {
567  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] { -- rsaPublicKey
603  3:       INTEGER 65537
:
608  2:     [503] { -- noAuthRequired
612  0:       NULL -- documentation indicates this is a
614  3:     [702] { -- origin
618  1:       INTEGER 0 -- KeyOrigin.GENERATED
:
621  2:     [703] { -- rollbackResistant
625  0:       NULL -- documentation indicates this is a
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'
712  [712] {  -- attestationIdProduct
720                    OCTET STRING 'walleye'
729  [713] {  -- attestationIdSerial
733                    OCTET STRING 'HT83K1A03849'
747  [716] {  -- attestationIdManufacturer
751                    OCTET STRING 'Google'
759  [717] {  -- attestationIdModel
763                    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 ]
795    0: NULL
808 35: INTEGER 2
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 [Page 27]
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 {
105  23:       SEQUENCE {
107  3:         OBJECT IDENTIFIER serialNumber (2 5 4 5)
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 98 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 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
    :     }

0E 55 6F 46 F5 3B 77 67 E1 B9 73 DC 55 E6 AE EA B4 FD 27 DD

01'B (bit 0)

[2] ’invalid;email:invalid’

'https://android.googleapis.com/attestation/crl/1'

'0345332940868793872'
776 13:  SEQUENCE {
778  9:    OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1 11)
780  0:    NULL
781  513:  BIT STRING
784  706:    69 13 A7 56 B3 9F E1 2B CE A2 09 89 E5 DC 03 B4
785  818:    B6 FF F6 1E 96 C7 62 C2 31 D1 B3 D6 1A 9E 36 CF
786  930:    C2 FC 0E 06 FA 0E CF B5 2D F8 19 D6 13 96 0B 56
787 1042:    B0 EE 86 3B B1 B8 38 70 4E 57 EB D9 60 DC 58 74
788 1154:    FE C8 EB A5 78 9F B7 19 5C F0 80 CF 29 16 6B 04
789 1266:    3A 5D 7C 2E 5F 11 12 36 BE 46 29 45 04 41 8F B5
790 1378:    AB C6 31 5F 23 28 0C F2 7C 48 4A F6 43 AA 50 D0
791 1490:    53 96 1E AD 7C A3 89 96 BB 8B BF 2D 9A 0C 16 35
792 1602:      [ Another 384 bytes skipped ]
794 1816:  } 0 1393:  SEQUENCE {
796 1857:   SEQUENCE {
799 1979:     [0] {
801 2091:       INTEGER 2
803 2203:         
805 2313:   INTEGER 03 88 26 67 60 65 89 96 85 74
808 2425:   SEQUENCE {
810 2537:     OBJECT IDENTIFIER
812 2649:       sha256WithRSAEncryption (1 2 840 113549 1 1 11)
814 2761:     NULL
816 2873:   } 40 27:  SEQUENCE {
818 2995:   SEQUENCE {
820 3107:     SET {
822 3219:       SEQUENCE {
824 3331:         OBJECT IDENTIFIER serialNumber (2 5 4 5)
826 3443:       PrintableString 'f92009e853b6b045'
828 3555:         
830 3667:     } 69 30:  SEQUENCE {
832 3789:   UTCtime 26/05/2016 17:01:32 GMT
834 3901:   UTCtime 24/05/2026 17:01:32 GMT
836 4013:   } 101 27:  SEQUENCE {
103  25:   SET {
105  23:     SEQUENCE {
107   3:       OBJECT IDENTIFIER serialNumber (2 5 4 5)
112   16:       PrintableString ’87f4514475ba0a2b’
120   13:         };
133  546:   SEQUENCE {
134   13:     SEQUENCE {
136    9:       OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
144   0:         NULL
149  527:       BIT STRING, encapsulates {
154  522:         SEQUENCE {
158  513:           INTEGER 65537
169  37:             [ Another 385 bytes skipped ]
184  3:           INTEGER 65537
193:         };
204  3:         };
216  [3] {
223  179:   SEQUENCE {
230   29:     SEQUENCE {
236   3:       OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
243   22:       OCTET STRING, encapsulates {
251   20:         OCTET STRING
259   0:           OCTET STRING
267   0:             OCTET STRING
274:         };
284:         };
294:         };
304:         };
314:         };
324:         };
334:         };
344:         [Another 385 bytes skipped]
355:         };
365:         };
375:         };
385:         };
395:         };
405:         };
415:         };
425:         };
435:         };
445:         };
455:         };
465:         };
475:         };
485:         };
495:         };
505:         };
515:         };
525:         };
535:         };
545:         };
555:         [Another 385 bytes skipped]
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 B2 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 {
8 840:  SEQUENCE {
  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'
    }
  52 4:    }
68 30:  SEQUENCE {
  70 13:    UTCTime 26/05/2016 16:28:52 GMT
  85 13:    UTCTime 24/05/2026 16:28:52 GMT
  98 10:    }
100 27:  SEQUENCE {
  102 25:    SET {
    104 23:      SEQUENCE {
      106 3:        OBJECT IDENTIFIER serialNumber (2 5 4 5)
      111 16:        PrintableString 'f92009e853b6b045'
    }
  114 4:    }
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 98 2F 32 C7 7F 75 31 03 0C 97 52 4B
          1B 5F EB 09 FB C7 2A A9 45 1F 74 3C BD 9A 6F 13
          35 74 4A A5 5E 77 F6 B6 AC 35 35 EE 17 C2 5E 63
          95 17 DD 9C 92 E6 37 4A 53 CB FE 25 8F 8F FB B6
          FD 12 93 78 A2 2A 4C A9 9C 45 2D 47 A5 9F 32 01

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
: 36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
: 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]
: 36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
: 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
: '1100001'B

782 64: SEQUENCE {
784 3: OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
789 57: OCTET STRING, encapsulates {
791 55: SEQUENCE {

Richardson, et al. Expires April 8, 2020
793  53:    SEQUENCE {  
795  51:      [0] {  
797  49:        [0] {  
799  47:          [6]  
801  45:            'https://android.googleapis.com/attestation/crl/'  
803  43:            }  
805  41:            }  
807  39:          }  
809  37:        }  
811  35:      }  
813  33:    }  
815  31:  
817  29:  
821  23:  7.1.2.  Secure Element  
825  22: 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.  
829  21:  
833  15:  
837  14: 0 5143: SEQUENCE {  
840  13: 4 9: OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)  
844  12: 15 5128: [0] {  
848  11: 19 5124: SEQUENCE {  
852  10: 23 1: INTEGER 1  
856  9: 26 0: SET {  
860  8: 28 11: SEQUENCE {  
864  7: 30 9: OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)  
868  6:  }  
872  5: 41 5100: [0] {  
876  4: 45 1114: SEQUENCE {  
880  3: 49 834: SEQUENCE {
[0] {
  INTEGER 2
}

INTEGER 1

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

SEQUENCE {
  INTEGER 1
}

SEQUENCE {
  OBJECT IDENTIFIER serialNumber (2 5 4 5)
  PrintableString '90e8da3cadfc7820'
}

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

UTCTime 01/01/1970 00:00:00 GMT

UTCTime 23/05/2028 23:59:59 GMT

SEQUENCE {
  OBJECT IDENTIFIER commonName (2 5 4 3)
  UTF8String 'Android Keystore Key'
}

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

BIT STRING, encapsulates {
  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
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
508 375:   SEQUENCE {
512 10:     OBJECT IDENTIFIER ‘1 3 6 1 4 11129 2 1 17’
524 359:     OCTET STRING, encapsulates {
528 355:       SEQUENCE {
532 1:         INTEGER 3
535 1:         ENUMERATED 2 -- attestationSecurityLevel (StrongBox)
538 1:         INTEGER 4
541 1:         ENUMERATED 2 -- attestationSecurityLevel (StrongBox)
544 9:         OCTET STRING ‘challenge’
555 0:         OCTET STRING
557 53:       SEQUENCE {
559 2:         [509] {
563 0:           NULL
565 11:         [701] {
569 9:           INTEGER 00 FF FF FF FF FF E5 99 78
580 28:         [709] {
584 26:           OCTET STRING, encapsulates {
586 24:             SEQUENCE {
588 20:               SET {
590 18:                 SEQUENCE {
592 13:                   OCTET STRING ‘AndroidSystem’
607 1:                     INTEGER 1
610 9:                     OCTET STRING

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:             [2] { 
634  1:                INTEGER 1 
:                     } 
637  4:             [3] { 
639  2:                INTEGER 2048 
:                     } 
643  8:             [4] { 
645  6:                SET { 
647  1:                   INTEGER 2 
650  1:                   INTEGER 32 
:                     } 
:                     } 
653  8:             [5] { 
655  6:                SET { 
657  1:                   INTEGER 0 
660  1:                   INTEGER 4 
:                     } 
:                     } 
663 14:             [6] { 
665 12:                SET { 
667  1:                   INTEGER 2 
670  1:                   INTEGER 3 
673  1:                   INTEGER 4 
676  1:                   INTEGER 5 
:                     } 
:                     } 
679  2:             [503] { 
683  0:                NULL 
:                     } 
685  3:             [702] { 
689  1:                INTEGER 0 
:                     } 
692  76:             [704] { 
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

BOOLEAN TRUE

ENUMERATED 0

OCTET STRING

77 96 C5 3D 0E 09 46 2B BA BB FB 7B 8A 65 F6 8D

EF 5C 46 88 BF 99 C4 1E 88 42 01 4D 1F 01 2D C5

INTEGER 0

INTEGER 201903

OCTET STRING 'google'

OCTET STRING 'blueline'

OCTET STRING 'blueline'

OCTET STRING '8A2X0KLUU'

OCTET STRING 'Google'

OCTET STRING 'Pixel 3'

INTEGER 20180905

INTEGER 201903

OBJECT IDENTIFIER

sha256WithRSAEncryption (1 2 840 113549 1 1 11)
900  0:        NULL :
   
902  257:       BIT STRING :
   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 1B 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 61 65 97 2C 0D A4 78 43 A7 :
   9A 27 B2 4E 24 11 4F FF E2 D8 04 56 39 75 B2 34 :
   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 :
1192       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' :
1222    1:               } :
1232   18:           SET {
1234   16:             SEQUENCE {
1236    3:               OBJECT IDENTIFIER title (2 5 4 12)
1241    9:               UTF8String 'StrongBox' :
1250    1:               } :
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' :
1298    1:                 } :
1300   1:               } :
1303   1:               } :
1305   1:             } :
1307   1:           } :
1309   1:         } :
1311   1:       } :
1313   1:     } :
1315   1:   } :
1317   1: }:

Richardson, et al. Expires April 8, 2020 [Page 40]
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
1366  252:               }
1367  251:             [3] { 1370  183:               SEQUENCE {
1373  29:               SEQUENCE {
1375   3:                   OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14) 1380  22:               OCTET STRING, encapsulates {
1382  20:               OCTET STRING :                   77 A4 AD DF 1D 29 89 CA 92 E3 BA DE 27 3C 70 DF :                   36 03 7C 0C
1394  31:               SEQUENCE { 1397  31:               SEQUENCE {
1400  3:                   OCTET STRING, encapsulates {
1403  24:               OCTET STRING, encapsulates {
1406  22:               SEQUENCE {
1409  20:               OCTET STRING :                   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]
1751  65:                                     'https://android.googleapis.com/attestation/crl/1'
1755  63:                                     '7102468407102977850'
1759  61:                                 }
1763  59:                             }
1767  57:                         }
1771  55:                     }
1775  53:                 }
1779  51:             }
1816  13:             SEQUENCE {
1818  9:                 OBJECT IDENTIFIER
1822  7:                     sha256WithRSAEncryption (1 2 840 113549 1 1 11)
1826  0:                 NULL
1831  513:             BIT STRING
1835  511:                 13 22 DA F2 92 93 CE C0 9F 70 40 C9 DA 85 6B 61
1839  509:                 6F 8F BE E0 A4 04 55 C1 63 84 61 37 F5 4B 71 6D
1843  507:                 62 AA 6F BF 6C E8 48 03 AD 28 85 21 9E 3C 1C 91
: 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
2358 8: 
2361 9:
      INTEGER 00 E8 FA 19 63 14 D2 FA 18
2372 13: 
2374 9:
      OBJECT IDENTIFIER
2374 12:
        sha256WithRSAEncryption (1 2 840 113549 1 1 11)
2380 0:
      NULL
2384 8: 
2387 27: SEQUENCE {
2390 25:
      SET {
2394 23:
        SEQUENCE {
2398 3:
          OBJECT IDENTIFIER serialNumber (2 5 4 5)
2402 16:
            PrintableString 'f92009e853b6b045'
2406 
2407 : 
2409 : 
2412 : 
2415 : 
2418 13:
      UTCTime 26/05/2016 16:28:52 GMT
2430 13:
      UTCTime 24/05/2026 16:28:52 GMT
2438 13:
      SEQUENCE {
2442 25:
      SET {
2446 23:
        SEQUENCE {
2450 3:
          OBJECT IDENTIFIER serialNumber (2 5 4 5)
2454 16:
            PrintableString 'f92009e853b6b045'
2458 
2459 : 
2461 : 
2464 : 
2467 : 
2471 546: SEQUENCE {
2475 13: SEQUENCE {
2479 9:
          OBJECT IDENTIFIER
2483 12:
            rsaEncryption (1 2 840 113549 1 1 1)
2491 0:
          NULL
2495 8:
          BIT STRING, encapsulates {
2503 522: SEQUENCE {
2507 513:
          INTEGER
2511 16:
            00 AF B6 C7 82 2B B1 A7 01 EC 2B 4B 2E 8B CC 54

Richardson, et al. Expires April 8, 2020 [Page 43]
Internet-Draft                 useful RATS                  October 2019

:  16 63 AB EF 98 2F 32 C7 7F 75 31 03 0C 97 52 4B
:  1B 5F E8 09 FB C7 2A A9 45 1F 74 3C BD 9A 6F 13
:  35 74 4A A5 5E 77 F6 B6 AC 35 35 EE 17 C2 5E 63
:  95 17 DD 9C 92 E6 37 4A 53 CB FE 25 8F 8F FB B6
:  FD 12 93 78 A2 2A 4C A9 9C 45 2D 47 A5 9F 32 01
:  F4 41 97 CA 1C CD 7E 76 2F B2 F5 31 51 B6 FE B2
:  FF FD 2B 6F E4 FE 5B C6 BD 9E C3 4B FE 08 23 9D
:  [ Another 385 bytes skipped ]

3022 3:

:  INTEGER 65537

3027 166:

:  [3] {
3030 163:

:  SEQUENCE {
3033 29:

:    SEQUENCE {
3035 3:

:      OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
3040 22:

:        OCTET STRING, encapsulates {
3042 20:

:          OCTET STRING
3044 19:

:            36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
3063 31:

:        }
3066 31:

:    }
3071 24:

:  OCTET STRING, encapsulates {
3073 22:

:    SEQUENCE {
3075 20:

:      [0]
3084 20:

:        36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
3097 15:

:        C9 EA 4F 12
3099 3:

:    }
3104 1:

:  BOOLEAN TRUE
3107 5:

:  OCTET STRING, encapsulates {
3109 3:

:    SEQUENCE {
3111 1:

:      BOOLEAN TRUE
3114 14:

:    }
3116 3:

:  OBJECT IDENTIFIER basicConstraints (2 5 29 19)
3121 1:

:  BOOLEAN TRUE
3124 4:

:  OCTET STRING, encapsulates {
3126 2:

:    BIT STRING 1 unused bit

Richardson, et al.        Expires April 8, 2020                [Page 44]
: }

3130 64:  SEQUENCE {
3132  3:          OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
3137  57:          OCTET STRING, encapsulates {
3139  55:                SEQUENCE {
3141  53:                    SEQUENCE {
3143  51:                        [0] {
3145  49:                            [0] {
3147  47:                               [6]

: 'https://android.googleapis.com/attestation/crl/'
:
:
:
:
:
:
:
:
:
:
:
:
:
:

3196 13:  SEQUENCE {
3198  9:          OBJECT IDENTIFIER
3206  0:          sha256WithRSAEncryption (1 2 840 113549 1 1 11)
3209  0:          NULL
3211 513:  BIT STRING

3728 1413:  SEQUENCE {
3732  877:  SEQUENCE {
3736  3:    [0] {
3738  1:      INTEGER 2

3741 10:      INTEGER 03 88 26 67 60 65 89 96 85 99
3753 13:  SEQUENCE {
3755  9:          OBJECT IDENTIFIER
3763  0:          sha256WithRSAEncryption (1 2 840 113549 1 1 11)
3766  0:          NULL
3768 27:  SEQUENCE {
SET {
  SEQUENCE {
    OBJECT IDENTIFIER serialNumber (2 5 4 5)
    PrintableString 'f92009e853b6b045'
  }
}

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

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

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

BIT STRING, encapsulates {
  INTEGER 65537
}

... [Another 385 bytes skipped] ...

Richardson, et al. Expires April 8, 2020 [Page 46]
SEQUENCE {
  SEQUENCE {
    OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
    OCTET STRING, encapsulates {
      OCTET STRING :
        1B 17 70 C6 97 DC 84 54 75 7C 3C 98 5C E6 1D 1D
        08 59 5D 53
      }
    }
  }
}

SEQUENCE {
  OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29 35)
  OCTET STRING, encapsulates {
    SEQUENCE {
      [0] :
        36 61 E1 00 7C 88 05 09 51 8B 44 6C 47 FF 1A 4C
        C9 EA 4F 12
      }
    }
  }
}

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 2 unused bits :
      '100000'B (bit 5)
  }
}

SEQUENCE {
  OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
  OCTET STRING, encapsulates {
    SEQUENCE {
      [0] {
      }
    }
  }
    'https://android.googleapis.com/attestation/crl/8'
    'F6734C9FA504789'
  }
}
The next two sections provide two views of a CSR generated via invocation of the Certificate Enrollment Manager API similar to the below:

7.2. Windows 10 TPM

[Another 384 bytes skipped]
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" + "MBYGA1UECzAEMjB5MB8GA1UEChMNVS5LUHNfMVYGEIhCS5LUHNfMMDA8GA1UDBDQg" + "gTmEC证书公司 MIIDbmcgA1UEAxMEQTEcMBoGA1UExMTUHvyc2QgZhYjMjdmbYJpdmfeSBQQTCCASi" + "wDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMROV8sQJ707OsVjRxxoS55vMaGwEO" + "r7n97x0jSVpejo3/Ow9G9QdJtG9gARK1xqgKOPJkTfTixxvUvWwRtL9hjYs" + "IC2V/otsXJkgPeup2CTty31lADU7UD0/0MGqALbn+grDTaZOSi5p6CA/aefx0" + "07UNh5rZyWYOYAhZdhfS5F9B1O2EN/7pRyvKuiupf3STVQaMjMWOidRQCC+D0xya4" + "8bqUxVFPYc49mi7uNzkHDqdaoq1G5v52/01W37lBd06HRZ54118aIX7s7n9k" + "M67GbK4q4/1FTMvI5bBn/Pp4sy1i3f+oyQbS+zFQwfwBGLukTUzYPCDFVUCawIC" + "A411MVYwQYVRV4DB0BEÂ7AqFEB9fl9prSM65GYyC0EVDPU91WJ0BXMASGA1UDWdQEAwIC" + "pDAnoBGvNvSUEITAIFbgbrgEFBQcdAqYkYBWQBUbHsEGs6aGAQQBkgcVDJANBkgqk" + "hkiG9w0BAQsFAAOCAQEG777Bu/S/EXmuHiVcAt0n5s8u4Z2b619j9w1j1G3qIryGM" + "2ox5KPr36c7R2FmAgq4m9N9wh4FbckkYheZFPsp0hRFy79veE+wMCw+20B88" + "ri4aq2/oTDMw9f3r+BaZ3RPvoaW9eztm26DA3wvEdUE2QG4V5VXdiSU" + "pFVd4eyEPVnYo09ZDZDBP9vVCDx57V9Fg8rZoqaDcerwrsXJ9/WLDZ76A6d2/syHN" + "74CruXYGhpB7yLjIhWv16Rb4Dq3d9DIkmTqUecEknuX730odd/phgCh0MVrWUB" + "1XrHJbBuC+nuPbShhJ0vPRw13TX2deqjTs2j7XEc==";

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: https://msdn.microsoft.com/en-us/library/dn408990.aspx.

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 1C 00 00 00
: 00 00 00 00 B9 04 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 01 00 00
: 00 03 00 00 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 51 4F CB E5 AD 8C 60
: E6 C2 70 80 00 00 D4 65 4C 6B 30 BF 82 73 9C 00
: 02 0B E6 2C AD 8D E8 9A 85 04 D7 F3 7B 87 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 1A F1 4B 12 A1 C5 D1 A4 54 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 0A 4F 3F
: 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 EB 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

Richardson, et al. Expires April 8, 2020 [Page 50]
The format is structured as follows:

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

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

The remainder is the keyAttestation, which is structured as follows:
typedef struct {
    UINT32 Magic;
    UINT32 Platform;
    UINT32 HeaderSize;
    UINT32 cbKeyAttest;
    UINT32 cbSignature;
    UINT32 cbKeyBlob;
    BYTE keyAttest[cbKeyAttest];
    BYTE signature[cbSignature];
    BYTE keyBlob[cbKeyBlob];
} keyAttestation;

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

keyAttest (161 bytes) -------------- FF 54 43 47 80 17 00 22 00 OB 9A FD
AB 8A 0B E9 0B BB 3F 7F 83 B6 6E 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 9D BF 7A
54 26 36 2F 10 D6 00 00 00 00 00 00 51 4F CB E5 AD 8C 8C 60 E6 C2 70 80 00
D4 2C 65 4C 95 ED 95 00 22 00 OB 2B E6 2C AD 8D 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 B3 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 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

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.  

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

Signature (256 bytes) - generated using the AIK private key

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]
53  3:       [0] { 
55  1:         INTEGER 2 
58  9:       INTEGER 00 A4 85 22 AA 34 AF AE 4F 
69 13:      SEQUENCE { 
71  9:        OBJECT IDENTIFIER 
75  1:          sha256WithRSAEncryption (1 2 840 113549 1 1 11) 
82  0:        NULL 
84 43:      SEQUENCE { 
86 41:        SET { 
89  3:         OBJECT IDENTIFIER commonName (2 5 4 3) 
95 32:       UTF8String ’Yubico PIV Root CA Serial 263751’ 
129 32:     SEQUENCE { 
131 13:      UTCTime 14/03/2016 00:00:00 GMT 
146 15:     GeneralizedTime 17/04/2052 00:00:00 GMT 
163 33:     SEQUENCE { 
165 31:      SET { 
167 3:       OBJECT IDENTIFIER commonName (2 5 4 3) 
174 22:     UTF8String ’Yubico PIV Attestation’ 
198 290:   SEQUENCE { 
202 13:   SEQUENCE { 
204 9:     OBJECT IDENTIFIER 
210 6:       rsaEncryption (1 2 840 113549 1 1 1) 
215 0:      NULL 
217 271:   BIT STRING 
221:       30 82 01 0A 02 82 01 01 00 AB A9 0B 16 9B EF 31 
225:       CC 3E AC 18 5A 2D 45 80 75 70 C7 5B 8D 6C 3F 1B 
229:       59 0D 49 B9 89 E8 6F CE BB 27 6F 08 3C 60 3A 85 
233:       00 EF 5C BC 40 99 3D 41 EE EA C0 81 7F 76 48 E4 
237:       A9 4C BC D5 6B E1 1F 0A 60 93 C6 FE AA D2 8D 8E 
241:       E2 B7 CD 8B 2B F7 9B DD 5A AB 2F CF B9 0E 54 CE 
245:       EC 8D F5 5E D7 7B 91 C3 A7 56 9C DC C1 06 86 76 
249:       36 44 53 FB 08 25 D8 06 B9 06 8C 81 FD 63 67 CA 
253:       [ Another 142 bytes skipped ] 
257:     } 
492 21:   [3] { 
494 19:   SEQUENCE { 

496  17:  SEQUENCE {
498  10:       OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3'
510  3:       OCTET STRING 04 03 03
     :        }
     :        }
515  13:  SEQUENCE {
517  9:       OBJECT IDENTIFIER
528  0:       NULL
     :        }
530  257:  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 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 11 FD 8D 30 F1 7B FC 71
     :        DF 72 A5 B6 CE D7 4C 90 13 DF DE DB F2 8B 34 45
     :        [ Another 128 bytes skipped ]
     :        }
791  783:  SEQUENCE {
795  503:    SEQUENCE {
799  3:     [0] {
801  1:       INTEGER 2
     :        }
804  17:    INTEGER
     :        00 FE B9 AF 03 3B 0B A7 79 04 02 F5 67 AE DF 72
     :        ED
823  13:    SEQUENCE {
825  9:       OBJECT IDENTIFIER
528  0:       NULL
     :        }
836  32:  SEQUENCE {
840  31:    SET {
842  29:     SEQUENCE {
844  3:       OBJECT IDENTIFIER commonName (2 5 4 3)
849  22:       UTF8String 'Yubico PIV Attestation'
     :        }
     :        }
873  32:  SEQUENCE {
875  13:    UTCTime 14/03/2016 00:00:00 GMT
890  15:    GeneralizedTime 17/04/2052 00:00:00 GMT
     :        }

Richardson, et al. Expires April 8, 2020 [Page 56]
907 37: SEQUENCE {
909 35: SET {
911 33: SEQUENCE {
913 3: OBJECT IDENTIFIER commonName (2 5 4 3)
918 26: UTF8String 'YubiKey PIV Attestation 9e'
946 290: SEQUENCE {
950 13: SEQUENCE {
952 9: OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
965 271: BIT STRING : 30 82 01 0A 02 82 01 01 00 93 C4 C0 35 95 7E 26 :
963 0: NULL : }
1240 60: [3] {
1242 58: SEQUENCE {
1244 17: SEQUENCE {
1246 10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3'
1258 3: OCTET STRING 04 03 03 -- firmware version :
1263 19: SEQUENCE {
1265 10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 7'
1277 5: OCTET STRING 02 03 4F 9B B5 -- serial number :
1284 16: SEQUENCE {
1286 10: OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 8'
1298 2: OCTET STRING 01 01 -- PIN and touch policy :
1302 13: SEQUENCE {
1304 9: OBJECT IDENTIFIER sha256WithRSAEncryption (1 2 840 113549 1 1 11)
1315 0: NULL : }
1317 257: BIT STRING
7.3.2. Yubikey 5

0 1613: SEQUENCE {
  4 9:   OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)
15 1598:   [0] {
19 1594:     SEQUENCE {
23 1:       INTEGER 1
26 0:       SET {}
28 11:     SEQUENCE {
30 9:       OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
  :   
41 1570:     [0] {
45 762:       SEQUENCE {
49 482:         SEQUENCE {
53 3:           [0] {
55 1:             INTEGER 2
  :           }
58 9:             INTEGER 00 86 77 17 E0 1D 19 2B 26
69 13:           SEQUENCE {
71 9:             OBJECT IDENTIFIER
  :             sha256WithRSAEncryption (1 2 840 113549 1 1 11)
82 0:             NULL
  :           }
84 43:           SEQUENCE {
86 41:             SET {
88 39:               SEQUENCE {
90 3:                 OBJECT IDENTIFIER commonName (2 5 4 3)
95 32:                 UTF8String 'Yubico PIV Root CA Serial 263751'
  :                 }
  :               }
129 32:           SEQUENCE {

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 {
    SEQUENCE {
        OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
        NULL
    }
    BIT STRING
        30 82 01 0A 02 82 01 01 00 C5 5B 8D E9 B9 3C 53
        69 82 88 FE DA 70 FC 5C 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
    }
}

SEQUENCE {
    SEQUENCE {
        OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3'
        OCTET STRING 05 01 02
    }
    SEQUENCE {
        OBJECT IDENTIFIER basicConstraints (2 5 29 19)
        BOOLEAN TRUE
        OCTET STRING 30 06 01 01 FF 02 01 00
    }
    SEQUENCE {
    }
}

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

BIT STRING
: 05 57 B7 BF 5A 41 74 F9 5F EC 2E D2 B8 78 26 E5
: EF 4F EA BF 5A 64 C9 CF 06 7F CA 8C 0A FC 1A 47
: 1C D6 AC ED C8 5B 54 72 00 9F B8 59 AB 73 25 B2
: D6 02 A3 59 83 31 69 EE C1 5F 3D F2 2B 1B 22 CA
: B6 FC F9 FB 21 32 9E 08 F3 08 54 6D C9 26 10 42
: 08 1D 3C B5 F0 5A B1 98 D4 68 DC 91 F1 D3 91 54
: 7A A0 34 B8 F6 65 E8 13 9F 3A 1C BF 43 C5 D1 D0
: 33 23 C6 25 A0 4C E4 E9 AA 59 80 D8 02 1E B0 10
: [ Another 128 bytes skipped ]

811 800: SEQUENCE {
815 520:   SEQUENCE {
819   3: [0] {
821   1: INTEGER 2

824   16: INTEGER
: 17 7D 2D F7 D6 6D 97 CC D6 CF 69 33 87 5B F1 5E

842   13: SEQUENCE {
844   9: OCTET STRING 'Yubico PIV Attestation'
847   0: NULL

857   33: SEQUENCE {
859   31: SET {
861   29: SEQUENCE {
863   3: OCTET STRING 'Yubico PIV Attestation'
866   22: UTF8String 'Yubico PIV Attestation'

892   32: SEQUENCE {
894   13: UTCTime 14/03/2016 00:00:00 GMT
909   15: GeneralizedTime 17/04/2052 00:00:00 GMT

926   37: SEQUENCE {
928   35: SET {
930   33: SEQUENCE {
932   3: OCTET STRING 'YubiKey PIV Attestation 9e'
935   26: UTF8String 'YubiKey PIV Attestation 9e'

965   290: SEQUENCE {
969   13: SEQUENCE {
971   9: OCTET STRING '
974   0: NULL

Richardson, et al. Expires April 8, 2020 [Page 60]
984 271:     BIT STRING
:   30 82 01 0A 02 82 01 01 00 A9 02 2D 7A 4C 0B B1
:   0C B0 F9 E5 9C E5 6F 20 D1 9D F9 CE B3 B3 4D 1B
:   61 B0 B4 E0 3F 44 19 72 B8 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 18 1B A7 A2 76 8F D4 6C D0 75 96 99 0D
:   06 37 9D 90 D5 71 00 6E FB 82 D1 5B 2A 7C 3B 62
:   9E AB 15 81 B9 AD 7F 3D 30 1C C2 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 ]
:
1259 78:   [3] {
1261 76:     SEQUENCE {
1263 17:       SEQUENCE {
1265 10:         OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 3'
1277 3:         OCTET STRING 05 01 02
-- firmware version
:
1282 20:       SEQUENCE {
1284 10:         OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 7'
1296 6:         OCTET STRING 02 04 00 93 6A A0     -- serial number
:       }
1304 16:       SEQUENCE {
1306 10:         OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 8'
1318 2:         OCTET STRING 01 01
-- PIN and touch policy
:
1322 15:       SEQUENCE {
1324 10:         OBJECT IDENTIFIER '1 3 6 1 4 1 41482 3 9'
1336 1:         OCTET STRING 02
-- form factor
:
:     }
:     }
:     }
:   }
1339 13:     SEQUENCE {
1341 9:       OBJECT IDENTIFIER
:         sha256WithRSAEncryption (1 2 840 113549 1 1 11)
1352 0:       NULL
:   }
1354 257:     BIT STRING
:   9F EB 7A 4C F0 7C 67 11 ED C5 84 07 C8 19 41 B2
:   71 42 08 2B D6 CD A8 5F DC AE 79 75 6C F1 E5 4D
:   28 95 89 69 9D C0 2E A7 D4 48 51 B0 75 85 FD 63 FD
:   B5 79 93 03 EA BB 8A 67 D8 E7 EC C9 1C 8E 3F AF
:   74 30 D4 7E 74 A4 26 50 9F D4 57 AE 23 C0 8A 63
:   4E F3 C7 CF 5A AF 91 11 A2 6B 3B 49 24 32 26 88
:   D8 4F 6F BE BC F0 2D A9 88 B4 5F 54 AF 42 72
:   08 74 64 57 76 5A 02 9A 9D 21 4B FD 7F 44 8F AF
:     [ Another 128 bytes skipped ]
:   
Richardson, et al. Expires April 8, 2020 [Page 61]
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]


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