RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 2, 2021
H. Birkholz
Fraunhofer SIT
E. Voit
Cisco
W. Pan
Huawei
March 31, 2021

Attestation Event Stream Subscription
draft-birkholz-rats-network-device-subscription-02

Abstract

This document defines how to subscribe to an Event Stream of attestation related Evidence on TPM-based network devices.

Table of Contents

1. Introduction

[I-D.ietf-rats-tpm-based-network-device-attest] and [I-D.ietf-rats-yang-tpm-charra] define the operational prerequisites and a YANG Model for the acquisition of Evidence from a TPM-based network device. However there is a limitation inherant in the challenge-response interaction models upon which these documents are based. This limitation is that it is up to the Verifier to request Evidence. The result is that the interval between the occurrence of a security event, and the event's visibility within the Relying Party can be unacceptably long.

This limitation results in two adverse effects:

1. Evidence is not streamed to an interested Verifier as soon as it is generated.

2. If it were to be streamed, the Evidence is not appraisable for freshness.

This specification addresses the first adverse effect by enabling a Verifier to subscribe via [RFC8639] to an <attestation> Event Stream which exists upon the Attester. When subscribed, the Attester will continuously stream a requested set of Evidence to the Verifier.

The second adverse effect results from the nonce based challenge-response of [I-D.ietf-rats-yang-tpm-charra]. In that document an Attester must wait for a new nonce from a Verifier before it generates a new TPM Quote. To address delays resulting from such a wait, this specification enables freshness to be asserted asynchronously.

By removing these two adverse effects, it becomes possible for a Verifier to continously maintain an appraisal of the Attested device without relying on continous polling.

2. Terminology

The following terms are imported from [I-D.ietf-rats-architecture]: Attester, Evidence, Relying Party, and Verifier. Also imported are the time definitions time(VG), time(NS), time(EG), time(RG), and time(RA) from that document's Appendix A. The following terms at imported from [RFC8639]: Event Stream, Subscription, Event Stream Filter, Dynamic Subscription.

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. Operational Model

3.1. Sequence Diagram

Figure 1 below is a sequence diagram which updates Figure 5 of [I-D.ietf-rats-tpm-based-network-device-attest]. This sequence diagram replaces the [I-D.ietf-rats-tpm-based-network-device-attest] challenge-response interaction model with an [RFC8639] Dynamic Subscription to an <attestation> Event Stream. The contents of the <attestation> Event Stream are defined below within Section 4.

```
.----------.                       .---------------------------.
| Attester |                       | Relying Party / Verifier |
'----------'                       '---------------------------'
   time(VG)                                                  |
     |<---------establish-subscription(<attestation>)--time(NS)
     |                                                       |
   time(EG)                                                  |
     |--filter(<pcr-extend>)---------------------------->|
     |--<tpm12-attestation> or <tpm20-attestation>------>|
     |                                                       |
     |                       verify time(EG) Evidence @ time(RG,RA)
     |                                                       |
     ~                                                       ~
   time(VG',EG')                                             |
     |--filter(<pcr-extend>)---------------------------->|
     |--<tpm12-attestation> or <tpm20-attestation>------>|
     |                                                       |
     |                      verify time(EG') Evidence @ time(RG',RA')
```

Figure 1: YANG Subscription Model for Remote Attestation

- o time(VG,RG,RA) are identical to the corresponding time definitions from [I-D.ietf-rats-tpm-based-network-device-attest]. Time(RG',RA') are subsequent instances of the corresponding times.

- o time(RG',RA') are subsequent instances of the corresponding times from Figure 5 of [I-D.ietf-rats-tpm-based-network-device-attest].

- o time(NS): The Verifier generates a nonce and makes an [RFC8639] <establish-subscription> request. This request also includes the augmentations defined in this document's YANG model. Key subscription RPC parameters include:

  * the nonce

  * a set of PCRs of interest which the Verifier wants to appraise

  * an optional filter which can reduce the logged events on the <attestation> stream pushed to the Verifier.

- o time(EG) - An initial response of Evidence is returned to the Verifier. This includes:

  * A replay of filtered log entries which have extended into a PCR of interest since boot are sent in the <pcr-extend> notification.

  * A signed TPM quote that contains at least the PCRs from the <establish-subscription> RPC are included in a <tpm12-attestation> or <tpm20-attestation>). This quote must have included the nonce provided at time(NS).

- o time(VG',EG') - This occurs when a PCR is extended subsequent to time(EG). Immediately after the extension, the following information needs to be pushed to the Verifier:

  * Any values extended into a PCR of interest, and

  * a signed TPM Quote showing the result the PCR extension.

3.2. Continuously Verifying Freshness

As there is no new Verifier nonce provided at time(EG'), it is important to validate the freshness of TPM Quotes which are delivered at that time. The method of doing this verification will vary based on the capabilities of the TPM cryptoprocessor used.

3.2.1. TPM 1.2 Quote

The [RFC8639] notification format includes the <eventTime> object. This can be used to determine the amount of time subsequent to the initial subscription each notification was sent. However this time is not part of the signed results which are returned from the Quote, and therefore is not trustworthy as objects returned in the Quote. Therefore a Verifier MUST periodically issue a new nonce, and receive this nonce within a TPM quote response in order to ensure the freshness of the results. This can be done using the <tpm12-challenge-response-attestation> RPC from [I-D.ietf-rats-yang-tpm-charra].

3.2.2. TPM 2 Quote

When the Attester includes a TPM2 compliant cryptoprocessor, internal time-related counters are included within the signed TPM Quote. By including a initial nonce in the [RFC8639] subscription request, fresh values for these counters are pushed as part of the first TPM Quote returned to the Verifier. And then as shown by [I-D.birkholz-rats-tuda], subsequent TPM Quotes delivered to the Verifier can the be appraised for freshness based on the predictable incrementing of these time-related countersr.

The relevant internal time-related counters defined within [TPM2.0] can be seen within <tpms-clock-info>. These counters include the <clock>, <reset-counter>, and <restart-counter> objects. The rules for appraising these objects are as follows:

o If the <clock> has incremented for no more than the same duration as both the <eventTime> and the Verifier's internal time since the initial time(EG) and any previous time(EG'), then the TPM Quote may be considered fresh. Note that [TPM2.0] allows for +/- 15% clock drift. However many chips significantly improve on this maximum drift. If available, chip specific maximum drifts SHOULD be considered during the appraisal process.

o If the <reset-counter>, <restart-counter> has incremented. The existing subscription MUST be terminated, and a new <establish-subscription> SHOULD be generated.

o If a TPM Quote on any subscribed PCR has not been pushed to the Verifier for a duration of an Attester defined heartbeat interval, then a new TPM Quote notification should be sent to the Verifier. This may often be the case, as certain PCRs might be infrequently updated.

```
.----------.                        .--------------------------.
| Attester |                        | Relying Party / Verifier |
'----------'                        '--------------------------'
   time(VG',EG')                                        |
     |-<tpm20-attestation>----------------------------->|
     |                                    :             |
     ~                             Heartbeat interval   ~
     |                                    :             |
   time(EG')                              :             |
     |-<tpm20-attestation>----------------------------->|
     |                                                  |
```

4. Remote Attestation Event Stream

The <attestation> Event Stream is an [RFC8639] compliant Event Stream which is defined within this section and within the YANG Module of [I-D.ietf-rats-yang-tpm-charra]. This Event Stream contains YANG notifications which carry Evidence which assists a Verifier in appraising the Trustworthiness Level of an Attester. Data Nodes within Section 4.6 allow the configuration of this Event Stream's contents on an Attester.

This <attestation> Event Stream may only be exposed on Attesters supporting [I-D.ietf-rats-tpm-based-network-device-attest]. As with [I-D.ietf-rats-tpm-based-network-device-attest], it is up to the Verifier to understand which types of cryptoprocessors and keys are acceptable.

4.1. Subscription to the <attestation> Event Stream

To establish a subscription to an Attester in a way which provides provably fresh Evidence, initial randomness must be provided to the Attester. This is done via the augmentation of a <nonce-value> into [RFC8639] the <establish-subscription> RPC. Additionally, a Verifier must ask for PCRs of interest from a platform.

```
  augment /sn:establish-subscription/sn:input:
    +---w nonce-value    binary
    +---w pcr-index*     tpm:pcr
```

The result of the subscription will be that passing of the following information:

1. <tpm12-attestation> and <tpm20-attestation> notifications which include the provided <nonce-value>. These attestation notifications MUST at least include all the <pcr-indicies> requested in the RPC.

2. a series of <pcr-extend> notifications which reference the requested PCRs on all TPM based cryptoprocessors on the Attester.

3. <tpm12-attestation> and <tpm20-attestation> notifications generated within a few seconds of the <pcr-extend> notifications. These attestation notifications MUST at least include any PCRs extended.

If the Verifier does not want to see the logged extend operations for all PCRs available from an Attester, an Event Stream Filter should be applied. This filter will remove Evidence from any PCRs which are not interesting to the Verifier.

4.2. Replaying a history of previous TPM extend operations

Unless it is relying on Known Good Values, a Verifier will need to acquire a history of PCR extensions since the Attester has been booted. This history may be requested from the Attester as part of the <establish-subscription> RPC. This request is accomplished by placing a very old <replay-start-time> within the original RPC request. As the very old <replay-start-time> will pre-date the time of Attester boot, a <replay-start-time-revision> will be returned in the <establish-subscription> RPC response, indicating when the Attester booted. Immediately following the response (and before the notifications above) one or more <pcr-extend> notifications which document all extend operations which have occurred for the requested PCRs since boot will be sent. Many extend operations to a single PCR index on a single TPM SHOULD be included within a single notification.

Note that if a Verifier has a partial history of extensions, the <replay-start-time> can be adjusted so that known extensions are not forwarded.

The end of this history replay will be indicated with the [RFC8639] <replay-completed> notification. For more on this sequence, see Section 2.4.2.1 of [RFC8639].

After the <replay-complete> notification is provided, a TPM Quote will be requested and the result passed to the Verifier via a <tpm12-attestation> and <tpm20-attestation> notification. If there have been any additional extend operations which have changed a subscribed PCR value in this quote, these MUST be pushed to the Verifier before the <tpm12-attestation> and <tpm20-attestation> notification.

At this point the Verifier has sufficient Evidence appraise the reported extend operations for each PCR, as well compare the expected value of the PCR value against that signed by the TPM.

4.2.1. TPM2 Heartbeat

For TPM2, make sure that every requested PCR is sent within an <tpm20-attestation> no less frequently than once per heartbeat interval. This MAY be done with a single <tpm20-attestation> notification that includes all requested PCRs every heartbeat interval. This MAY be done with several <tpm20-attestation> notifications at different times during that heartbeat interval.

4.3. YANG notifications placed on the <attestation> Event Stream

4.3.1. pcr-extend

This notification documents when a subscribed PCR is extended within a single TPM cryptoprocessor. It SHOULD be emmitted no less than the <marshalling-period> after an the PCR is first extended. (The reason for the marshalling is that it is quite possible that multiple extensions to the same PCR have been made in quick succession, and these should be reflected in the same notification.) This notification MUST be emmitted prior to a <tpm12-attestation> or <tpm20-attestation> notification which has included and signed the results of any specific PCR extension. If pcr extending events occur during the generation of the <tpm12-attestation> or <tpm20-attestation> notification, the marshalling period MUST be extended so that a new <pcr-extend> is not sent until the corresponding notifications have been sent.

```
    +---n pcr-extend
       +--ro certificate-name     certificate-name-ref
       +--ro pcr-index-changed*   tpm:pcr
       +--ro attested-event* []
          +--ro attested-event
             +--ro extended-with             binary
             +--ro (event-details)?
                +--:(bios-event-log)
                |  +--ro bios-event-entry* [event-number]
                |     +--ro event-number    uint32
                |     +--ro event-type?     uint32
                |     +--ro pcr-index?      pcr
                |     +--ro digest-list* []
                |     |  +--ro hash-algo?   identityref
                |     |  +--ro digest*      binary
                |     +--ro event-size?     uint32
                |     +--ro event-data*     uint8
                +--:(ima-event-log)
                |  +--ro ima-event-entry* [event-number]
                |     +--ro event-number                uint64
                |     +--ro ima-template?               string
                |     +--ro filename-hint?              string
                |     +--ro filedata-hash?              binary
                |     +--ro filedata-hash-algorithm?    string
                |     +--ro template-hash-algorithm?    string
                |     +--ro template-hash?              binary
                |     +--ro pcr-index?                  pcr
                |     +--ro signature?                  binary
                +--:(netequip-boot-event-log)
                   +--ro boot-event-entry* [event-number]
                      +--ro event-number                uint64
                      +--ro filename-hint?              string
                      +--ro filedata-hash?              binary
                      +--ro filedata-hash-algorithm?    string
                      +--ro file-version?               string
                      +--ro file-type?                  string
                      +--ro pcr-index?                  pcr
```

Each <pcr-extend> MUST include one or more values being extended into the PCR. These are passed within the <extended-with> object. For each extension, details of the event SHOULD be provided within the <event-details> object. The format of any included <event-details> is identified by the <event-type>. This document includes two YANG structures which may be inserted into the <event-details>. These two structures are: <ima-event-log< and <bios-event-log>.

Implementations wanting to provide additional documentation of a type of PCR extension may choose to define additional YANG structures which can be placed into <event-details>.

4.3.2. tpm12-attestation

This notification contains an instance of a TPM1.2 style signed cryptoprocessor measurement. It is supplemented by Attester information which is not signed. This notification is generated and emitted from an Attester when at least one PCR identified within the subscribed <pcr-indices> has changed from the previous <tpm12-attestation> notification. This notification MUST NOT include the results of any PCR extensions not previously reported by a <pcr-extend>. This notification SHOULD be emitted as soon as a TPM Quote can extract the latest PCR hashed values. This notification MUST be emitted prior to a subsequent <pcr-extend>.

```
+---n tpm12-attestation {taa:TPM12}?
   +--ro certificate-name       tpm:certificate-name-ref
   +--ro up-time?               uint32
   +--ro TPM_QUOTE2?            binary
   +--ro TPM12-hash-algo?       identityref
   +--ro unsigned-pcr-values* []
      +--ro pcr-index*   tpm:pcr
      +--ro pcr-value*   binary
```

All YANG objects above are defined within [I-D.ietf-rats-yang-tpm-charra]. The <tpm12-attestation> is not replayable.

4.3.3. tpm20-attestation

This notification contains an instance of TPM2 style signed cryptoprocessor measurements. It is supplemented by Attester information which is not signed. This notification is generated at two points in time:

- o every time at least one PCR has changed from a previous tpm20-attestation. In this case, the notification SHOULD be emitted within 10 seconds of the corresponding <pcr-extend> being sent:

- o after a locally configurable minimum heartbeat period since a previous tpm20-attestation was sent.

```
      +---n tpm20-attestation {taa:TPM20}?
         +--ro certificate-name       tpm:certificate-name-ref
         +--ro TPMS_QUOTE_INFO        binary
         +--ro quote-signature?       binary
         +--ro up-time?               uint32
         +--ro unsigned-pcr-values* []
            +--ro TPM20-hash-algo?   identityref
            +--ro pcr-values* [pcr-index]
               +--ro pcr-index    pcr
               +--ro pcr-value?   binary
```

All YANG objects above are defined within [I-D.ietf-rats-yang-tpm-charra]. The <tpm20-attestation> is not replayable.

4.4. Filtering Evidence at the Attester

It can be useful _not_ to receive all Evidence related to a PCR. An example of this is would be a when a Verifier maintains known good values of a PCR. In this case, it is not necessary to send each extend operation.

To accomplish this reduction, when an RFC8639 <establish-subscription> RPC is sent, a <stream-filter> as per RFC8639, Section 2.2 can be set to discard a <pcr-extend> notification when the <pcr-index-changed> is uninteresting to the verifier.

4.5. Replaying previous PCR Extend events

To verify the value of a PCR, a Verifier must either know that the value is a known good value [KGV] or be able to reconstruct the hash value by viewing all the PCR-Extends since the Attester rebooted. Wherever a hash reconstruction might be needed, the <attestation> Event Stream MUST support the RFC8639 <replay> feature. Through the <replay> feature, it is possible for a Verifier to retrieve and sequentially hash all of the PCR extending events since an Attester booted. And thus, the Verifier has access to all the evidence needed to verify a PCR's current value.

4.6. Configuring the <attestation> Event Stream

Figure 2 is tree diagram which exposes the operator configurable elements of the <attestation> Event Stream. This allows an Attester to select what information should be available on the stream. A fetch operation also allows an external device such as a Verifier to understand the current configuration of stream.

Almost all YANG objects below are defined via reference from [I-D.ietf-rats-yang-tpm-charra]. There is one object which is new with this model however. <tpm2-heartbeat> defines the maximum amount of time which should pass before a subscriber to the Event Stream should get a <tpm20-attestation> notification from devices which contain a TPM2.

```
  augment /tpm:rats-support-structures:
    +--rw marshalling-period?                  uint8
    +--rw tpm12-subscribed-signature-scheme?
    |    -> ../tpm:attester-supported-algos/tpm12-asymmetric-signing
    |       {taa:TPM12}?
    +--rw tpm20-subscribed-signature-scheme?
    |    -> ../tpm:attester-supported-algos/tpm20-asymmetric-signing
    |       {taa:TPM20}?
    +--rw tpm20-subscription-heartbeat?        uint16
  augment /tpm:rats-support-structures/tpm:tpms:
    +--rw subscription-aik?        tpm:certificate-name-ref
    +--rw (subscribable)?
       +--:(tpm12-stream) {taa:TPM12}?
       |  +--rw TPM12-hash-algo?   identityref
       |  +--rw tpm12-pcr-index*   tpm:pcr
       +--:(tpm20-stream) {taa:TPM20}?
          +--rw TPM20-hash-algo?   identityref
          +--rw tpm20-pcr-index*   tpm:pcr
```

Figure 2: Configuring the \<attestation\> Event Stream

5. YANG Module

This YANG module imports modules from [I-D.ietf-rats-yang-tpm-charra] and [RFC8639]. It is also work-in-progress.

```
<CODE BEGINS> ietf-rats-attestation-stream@2020-12-15.yang
module ietf-tpm-remote-attestation-stream {
  yang-version 1.1;
  namespace
     "urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation-stream";
  prefix tras;

  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-tpm-remote-attestation {
    prefix tpm;
    reference
```

```
       "draft-ietf-rats-yang-tpm-charra";
   }
   import ietf-tcg-algs {
     prefix taa;
   }

   organization "IETF";
   contact
     "WG Web:   <http://tools.ietf.org/wg/rats/>
      WG List:  <mailto:rats@ietf.org>

      Editor:   Eric Voit
                <mailto:evoit@cisco.com>";

   description
     "This module contains conceptual YANG specifications for
      subscribing to attestation streams being generated from TPM chips.

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

      Redistribution and use in source and binary forms, with
      or without modification, is permitted pursuant to, and
      subject to the license terms contained in, the Simplified
      BSD License set forth in Section 4.c of the IETF Trust's
      Legal Provisions Relating to IETF Documents
      (https://trustee.ietf.org/license-info).

      Redistribution and use in source and binary forms, with or
      without modification, is permitted pursuant to, and subject to
      the license terms contained in, the Simplified BSD License set
      forth in Section 4.c of the IETF Trust's Legal Provisions
      Relating to IETF Documents
      (https://trustee.ietf.org/license-info).

      This version of this YANG module is part of RFC XXXX
      (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
      itself for full legal notices.

      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 (RFC 2119)
      (RFC 8174) when, and only when, they appear in all
      capitals, as shown here.";

   revision 2020-12-15 {
     description
```

```
       "Initial version.";
     reference
       "draft-birkholz-rats-network-device-subscription";
   }

   /*
    * IDENTITIES
    */

   identity pcr-unsubscribable {
     base sn:establish-subscription-error;
     description
       "Requested PCR is subscribable by the Attester.";
   }


   /*
    * Groupings
    */

   grouping heartbeat {
     description
       "Allows an Attester to push verifiable, current TPM PCR values
       even when there have been no recent changes to PCRs.";
     leaf tpm20-subscription-heartbeat {
       type uint16;
       description
         "Number of seconds before the Attestation stream should send a
         new notification with a fresh quote.  This allows confirmation
         that the PCR values haven't changed since the last
         tpm20-attestation.";
     }
   }


   /*
    * RPCs
    */

   augment "/sn:establish-subscription/sn:input" {
     when 'derived-from-or-self(sn:stream, "attestation")';
     description
       "This augmentation adds a nonce to as a subscription parameters
        that apply specifically to datastore updates to RPC input.";
     uses tpm:nonce;
     leaf-list pcr-index {
       type tpm:pcr;
       min-elements 1;
```

```
       description
         "The numbers/indexes of the PCRs. This will act as a filter for
         the subscription so that 'tpm-extend' notifications related to
         non-requested PCRs will not be sent to a subscriber.";
     }
   }

   /*
    * NOTIFICATIONS
    */

   notification pcr-extend {
     description
       "This notification indicates that one or more PCRs have been
       extended within a TPM based cryptoprocessor.  In less than the
       'marshalling-period', it MUST be followed with either a
       corresponding tpm12-attestation or tpm20-attestation notification
       which exposes the result of the PCRs updated.";
     uses tpm:certificate-name-ref;
     leaf-list pcr-index-changed {
       type tpm:pcr;
       min-elements 1;
       description
         "The number of each PCR extended.  This list MUST contain the
         set of PCRs descibed within the event log details.  This leaf
         can be derived from the list of attested events, but exposing
         it here allows for easy filtering of the notifications of
         interest to a verifier.";
     }
     list attested-event {
       description
         "A set of events which extended an Attester PCR.  The sequence
         of elements represented in list must match the sequence of
         events placed into the TPM's PCR.";
       container attested-event {
         description
           "An instance of an event which extended an Attester PCR";
         leaf extended-with {
           type binary;
           mandatory true;
           description
             "Information extending the PCR.";
         }
         choice event-details {
           description
             "Contains the event happened the Attester thought
             was worthy of recording in a PCR.
```

```
            choices are of types defined by the identityref
            base tpm:attested_event_log_type";

          case bios-event-log {
            description
              "BIOS/UEFI event log format";
            uses tpm:bios-event-log;
          }
          case ima-event-log {
            description
              "IMA event log format";
            uses tpm:ima-event-log;
          }
          case netequip-boot-event-log {
            description
              "IMA event log format";
            uses tpm:network-equipment-boot-event-log;
          }
        }
      }
    }
  }

  notification tpm12-attestation {
    if-feature "taa:TPM12";
    description
      "Contains an instance of TPM1.2 style signed cryptoprocessor
      measurements.  It is supplemented by unsigned Attester
      information.";
    leaf certificate-name {
      type tpm:certificate-name-ref;
      mandatory true;
      description
        "Allows a TPM quote to be associated with a certificate.";
    }
    uses tpm:tpm12-attestation;
    uses tpm:TPM12-hash-algo;
    list unsigned-pcr-values {
      description
        "Allows notifications to be filtered by PCR number or
        PCR value based on via YANG related mechanisms such as PATH.
        This is done without requiring the filtering structure to be
        applied against TCG structured data.";
      leaf-list pcr-index {
        type tpm:pcr;
        description
          "PCR index number.";
        min-elements 1;
```

```
        }
        leaf-list pcr-value {
          type binary;
          description
            "PCR value in a sequence which matches to the 'pcr-index'.";
        }
      }
    }

    notification tpm20-attestation {
      if-feature "taa:TPM20";
      description
        "Contains an instance of TPM2 style signed cryptoprocessor
        measurements.  It is supplemented by unsigned Attester
        information.";
      leaf certificate-name {
        type tpm:certificate-name-ref;
        mandatory true;
        description
          "Allows a TPM quote to be associated with a certificate.";
      }
      uses tpm:tpm20-attestation {
        description
          "Provides the attestation info.  Also ensures PCRs can be XPATH
          filtered by refining the unsigned data so that it appears.";
        refine unsigned-pcr-values {
          min-elements 1;
        }
        refine unsigned-pcr-values/pcr-values {
          min-elements 1;
        }
      }
    }


    /*
     * DATA NODES
     */

    augment "/tpm:rats-support-structures" {
      description
        "Defines platform wide 'attestation' stream subscription
        parameters.";
      leaf marshalling-period {
        config true;
        type uint8;
        default 5;
        description
```

```
          "The maximum number of seconds between the time an event
          extends a PCR, and the 'tpm-extend' notification which reports
          it to a subscribed Verifier.  This period allows multiple
          extend operations bundled together and handled as a group.";
      }
      leaf tpm12-subscribed-signature-scheme {
        if-feature "taa:TPM12";
        type leafref {
          path "../tpm:attester-supported-algos" +
                 "/tpm:tpm12-asymmetric-signing";
        }
        description
          "A single signature-scheme which will be used to sign the
          evidence from a TPM 1.2. which is then placed onto the
          'attestation' event stream.";
      }
      leaf tpm20-subscribed-signature-scheme {
        if-feature "taa:TPM20";
        type leafref {
          path "../tpm:attester-supported-algos" +
                 "/tpm:tpm20-asymmetric-signing";
        }
        description
          "A single signature-scheme which will be used to sign the
          evidence from a TPM 2.0. which is then placed onto the
          'attestation' event stream.";
      }
      uses heartbeat{
        if-feature "taa:TPM20";
      }
    }

    augment "/tpm:rats-support-structures/tpm:tpms" {
      description
        "Allows the configuration 'attestation' stream parameters for a
        TPM.";
      leaf subscription-aik {
        config true;
        type tpm:certificate-name-ref;
        description
          "Identifies the certificate-name associated with the
          notifications in the 'attestation' stream.";
      }
      choice subscribable {
        config true;
        description
          "Indicates that the set of notifications which comprise the
          'attestation' event stream can be modified or tuned by a
```

```
          network administrator.";
        case tpm12-stream {
          if-feature "taa:TPM12";
          description
            "Configuration elements for a TPM1.2 event stream.";
          uses tpm:TPM12-hash-algo;
          leaf-list tpm12-pcr-index {
            type tpm:pcr;
            description
              "The numbers/indexes of the PCRs which can be subscribed.";
          }
        }
        case tpm20-stream {
          if-feature "taa:TPM20";
          description
            "Configuration elements for a TPM2.0 event stream.";
          uses tpm:TPM20-hash-algo;
          leaf-list tpm20-pcr-index {
            type tpm:pcr;
            description
              "The numbers/indexes of the PCRs which can be subscribed.";
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Security Considerations

To be written.

7. IANA Considerations

To be written.

8. References

8.1. Normative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-08 (work in progress), December 2020.

[I-D.ietf-rats-tpm-based-network-device-attest]
Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", draft-ietf-rats-tpm-based-network-device-attest-06 (work in progress), December 2020.

[I-D.ietf-rats-yang-tpm-charra]
Birkholz, H., Eckel, M., Bhandari, S., Voit, E., Sulzen, B., Xia, L., Laffey, T., and G. Fedorkow, "A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs", draft-ietf-rats-yang-tpm-charra-05 (work in progress), January 2021.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[TPM2.0] TCG, "TPM 2.0 Library Specification", n.d., <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

8.2. Informative References

[I-D.birkholz-rats-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-rats-tuda-04 (work in progress), January 2021.

[KGV] TCG, "KGV", October 2003, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-NetEq-Attestation-Workflow-Outline_v1r9b_pubrev.pdf>.

Appendix A. Change Log

v01-v02

- o Match YANG changes/simplifications made to charra v00-v01

- o rename notification: pcr-extended, which supports multiple PCRs
- o netequip boot added
- o YANG structure extension removed
- o Matched to structural changes made within charra

Acknowledgements

Thanks to ...

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Eric Voit
Cisco Systems, Inc.

Email: evoit@cisco.com

Wei Pan
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu 210012
China

Email: william.panwei@huawei.com

RATS H. Birkholz
Internet-Draft Fraunhofer SIT
Intended status: Standards Track B. Moran
Expires: July 17, 2021 Arm Limited
January 13, 2021

Trustworthiness Vectors for the Software Updates of Internet of Things (SUIT) Workflow Model
draft-birkholz-rats-suit-claims-01

Abstract

The IETF Remote Attestation Procedures (RATS) architecture defines Conceptual Messages as input and output of the appraisal process that assesses the trustworthiness of remote peers: Evidence and Attestation Results. Based on the Trustworthiness Vectors defined in Trusted Path Routing, this document defines a core set of Claims to be used in Evidence and Attestation Results for the Software Update for the Internet of Things (SUIT) Workflow Model. Consecutively, this document is in support of the Trusted Execution Environment Provisioning (TEEP) architecture, which defines the assessment of remote peers via RATS and uses SUIT for evidence generation as well as a remediation measure to improve trustworthiness of given remote peers.

Table of Contents

1. Introduction

Attestation Results are an essential output of Verifiers as defined in the Remote ATtestation procedureS (RATS) architecture [I-D.ietf-rats-architecture]. They are consumed by Relying Parties: the entities that intend to build future decisions on trustworthiness assessments of remote peers. Attestation Results must be easily appraised by Relying Parties - in contrast to the rather complex or domain-specific Evidence appraised by Verifiers.

In order to create Attestation Results, a Verifier must consume Evidence generated by a given Attester (amongst other Conceptual Messages, such as Endorsements and Attestation Policies). Both Evidence and Attestation Results are composed of Claims. This document highlights and defines a set of Claims to be used in Evidence and Attestation Results that are based on the SUIT Workflow Model [I-D.ietf-suit-manifest]. In the scope of this document, an Attester takes on the role of a SUIT Recipient: the system that receives a SUIT Manifest.

1.1. SUIT Workflow Model and Procedures

This document focuses on Evidence and Attestation Results that can be generated based on the output of SUIT Procedures. The SUIT Workflow Model allows for two types of SUIT Procedures generating Reports on the Attester as defined in the SUIT Manifest specification [I-D.ietf-suit-manifest]:

Update Procedures: A procedure that updates a device by fetching dependencies, software images, and installing them.

An Update Procedure creates a Report about mutable software components that are installed or updated on hardware components.

Boot Procedures: A procedure that boots a device by checking dependencies and images, loading images, and invoking one or more image.

A Boot Procedure creates a Report on measured boot events (e.g. during Secure Boot).

The Records contained in each type of Report can be used as Claims in Evidence generation on the Attester for Remote Attestation Procedures as described in this document. Analogously, a corresponding Verifier appraising that Evidence can generate Attestation Results using the Claims defined in this document.

Both types of SUIT Procedures pass several stages (e.g. dependency-checking is one stage). The type and sequence of stages are defined by the Command Sequences included in a SUIT Manifest. For each stage in which a Command from the Command Sequence is executed a Record is created. All Records of a SUIT procedure contain binary results limited to "fail" or "pass". The aggregated sequence of all Records is composed into a Report.

This document specifies new Claims derived from Command Sequence Reports and highlights existing Claims as defined in Trusted Path Routing [I-D.voit-rats-trusted-path-routing] that are applicable to the operational state of installed and updated software.

The Claims defined in this document are in support of the Trusted Execution Environment Provisioning (TEEP) architecture. During TEEP, the current operational state of an Attester is assessed via RATS. If the corresponding Attestation Results - as covered in this document - indicate insufficient Trustworthiness Levels with respect to installed software, the SUIT Workflow Model is used for remediation.

1.2. Terminology

This document uses the terms and concepts defined in [I-D.ietf-rats-architecture], [I-D.ietf-suit-manifest], and [I-D.ietf-teep-architecture].

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. Trustworthiness Vectors

While there are usage scenarios where Attestation Results can be binary decisions, more often than not the assessment of trustworthiness is represented by a more fine-grained spectrum or based on multiple factors. These shades of Attestation Results are captured by the definition of Trustworthiness Vectors in Trusted Path Routing [I-D.voit-rats-trusted-path-routing]. Trustworthiness Vectors are sets of Claims representing appraisal outputs created by a Verifier. Each of these Claims is called a Trustworthiness Level. Multiple Trustworthiness Levels are composed into a vector.

An Attester processing SUIT Manifests can create three types of Claims about its Target Environments. This includes Claims about:

- o installed manifests including initial state (e.g. factory default),

- o hardware component identifiers that represent the targets of updates, and

- o SUIT Interpreter results (e.g. test-failed) created during updates.

Every SUIT Manifest maps to a certain intended state of a device. Every intended device composition of software components associated with hardware components can therefore be expressed based on a SUIT Manifest. The current operational state of a device can be represented in the same form, including the initial state.

As a result, the Claims defined in this document are bundled by the scope of the information represented in SUIT Manifests, i.e., dedicated blobs of software that are the payload of a SUIT Manifest. All Claims associated with an identifiable SUIT Manifest must always be bundled together in a Claims set that is limited to the Claims defined in this document.

3. SUIT Claims

The Claim description in this document uses CDDL as the formal modeling language for Claims. This approach is derived from [I-D.ietf-rats-eat]. All Claims are based on information elements as used in the SUIT Manifest specification [I-D.ietf-suit-manifest]. For instance, a SUIT Vendor ID is represented as an UUID. Analogously, the corresponding vendor-identifier Claim found below is based on a UUID. SUIT Claims are differentiated in:

- o software and hardware characteristics (System Properties), and
- o reports about updates their SUIT Commands (SUIT Records).

Both types of Claims are always bundled in dedicated Claim Sets. Implementations can encode this information in various different ways (data models), e.g., sets, sequences, or nested structures. The following subsections define the SUIT Report Claims for RATS and are structured according to the following CDDL expression.

```
suit-report = {
  suit-system-properties => [ + system-property-claims ],
  suit-records => [ + interpreter-record-claims ],
}

system-property-claims => { + $$system-property-claim }
interpreter-record-claims => { + $$interpreter-record-claim }
```

3.1. System Properties Claims

System Properties Claims are composed of:

- o Hardware Component Claims and
- o Software Component Claims.

Correspondingly, the Claim definitions below highlight if a Claim is generic or hw/sw-component specific.

3.1.1. vendor-identifier

A RFC 4122 UUID representing the vendor of the Attester or one of its hardware and/or software components.

```
$$system-property-claim //= ( vendor-identifier => RFC4122_UUID )
```

3.1.2. class-identifier

A RFC 4122 UUID representing the class of the Attester or one of its hardware and/or software components.

```
$$system-property-claim //= ( class-identifier => RFC4122_UUID )
```

3.1.3. device-identifier

A RFC 4122 UUID representing the Attester.

```
$$system-property-claim //= ( device-identifier => RFC4122_UUID )
```

3.1.4. component-identifier

A sequence of binary identifiers that is intended to identify a software-component of an Attester uniquely. A binary identifier can represent a CoSWID [I-D.ietf-sacm-coswid] tag-id.

```
$$system-property-claim //=  ( class-identifier => [ + identifier ] )
```

3.1.5. image-digest

A fingerprint computed over a software component image on the Attester. This Claim is always bundled with a component-identifier or component-index.

```
$$system-property-claim //= ( image-digest => digest )
```

3.1.6. image-size

The size of a firmware image on the Attester.

```
$$system-property-claim //= ( image-size => size )
```

3.1.7. minimum-battery

The configured minimum battery level of the Attester in mWh.

```
$$system-property-claim //= ( minimum-battery => charge )
```

3.1.8. version

The Version of a hardware or software component of the Attester.

```
$$system-property-claim //= ( version => version-value )
```

3.2. Interpreter Record Claims

This class of Claims represents the content of SUIT Records generated by Interpreters running on Recipients. They are always bundled into Claim Sets representing SUIT Reports and are intended to be included in Evidence generated by an Attester. The Interpreter Record Claims appraised by a Verifier can steer a corresponding a Firmware Appraisal procedures that consumes this Evidence. Analogously, these Claims can be re-used in generated Attestation Results as Trustworthiness Vectors [I-D.voit-rats-trusted-path-routing].

3.2.1. record-success

The result of a Command that was executed by the Interpreter on an Attester.

```
$$interpreter-record-claim //= ( record-success => bool )
```

3.2.2. component-index

A positive integer representing an entry in a flat list of indices mapped to software component identifiers to be updated.

```
$$system-property-claim //= ( component-index => uint )
```

3.2.3. dependency-index

A thumbprint of a software component that an update depends on.

```
$$interpreter-record-claim //= ( dependency-index => digest )
```

3.2.4. command-index

A positive integer representing an entry in a SUIT_Command_Sequence identifying a Command encoded as a SUIT Manifest Directive or SUIT Manifest Condition.

```
$$interpreter-record-claim //= ( command-index => uint )
```

3.2.5. nominal-parameters

A list of SUIT_Parameters associated with a specific Command encoded as a SUIT Manifest Directive.

```
$$interpreter-record-claim //= ( nominal-parameters => parameter-list )
```

3.2.6. nominal-parameters

A list of SUIT_Parameters associated with a specific Command that was executed by the Interpreter on an Attester.

```
$$interpreter-record-claim //= ( actual-parameters => parameter-list )
```

3.3. Generic Record Conditions (TBD)

- test-failed
- unsupported-command
- unsupported-parameter
- unsupported-component-id
- payload-unavailable
- dependency-unavailable
- critical-application-failure
- watchdog-timeout

4. List of Commands (TBD)

- Check Vendor Identifier
- Check Class Identifier
- Verify Image
- Set Component Index
- Override Parameters
- Set Dependency Index

- o Set Parameters
- o Process Dependency
- o Run
- o Fetch
- o Use Before
- o Check Component Offset
- o Check Device Identifier
- o Check Image Not Match
- o Check Minimum Battery
- o Check Update Authorized
- o Check Version
- o Abort
- o Try Each
- o Copy
- o Swap
- o Wait For Event
- o Run Sequence
- o Run with Arguments

5. References

5.1. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

5.2. Informative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-08 (work in progress), December 2020.

[I-D.ietf-rats-eat]
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-06 (work in progress), December 2020.

[I-D.ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-16 (work in progress), November 2020.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", draft-ietf-suit-manifest-11 (work in progress), December 2020.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-13 (work in progress), November 2020.

[I-D.voit-rats-trusted-path-routing]
Voit, E., "Trusted Path Routing", draft-voit-rats-trusted-path-routing-02 (work in progress), June 2020.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 17, 2021

A. Fuchs
H. Birkholz
Fraunhofer SIT
I. McDonald
High North Inc
C. Bormann
Universitaet Bremen TZI

January 13, 2021

Time-Based Uni-Directional Attestation
draft-birkholz-rats-tuda-04

Abstract

This document defines the method and bindings used to convey Evidence via Time-based Uni-Directional Attestation (TUDA) in Remote ATtestation procedureS (RATS). 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. TUDA enables the creation of Secure Audit Logs that can constitute believable Evidence about both current and past operational states of an Attester. In TUDA, RATS entities require access to a Handle Distributor to which a trustable and synchronized time-source is available. The Handle Distributor takes on the role of a Time Stamp Authority (TSA) to distribute Handles incorporating Time Stamp Tokens (TST) to the RATS entities. RATS require an Attesting Environment that generates believable Evidence. While a TPM is used as the corresponding root of trust in this specification, any other type of root of trust can be used with TUDA.

Status of This Memo

Table of Contents

1. Introduction

Remote ATtestation procedureS (RATS) describe the attempt to determine and appraise system properties, such as integrity and trustworthiness, of a remote peer - the Attester - by the use of a Verifier in support of Relying Parties that intend to interact with the Attester. The Verifier carries the burden of appraisal of detailed Evidence about an Attester's trustworthiness. Evidence is generated by the Attester and consumed by the Verifier. To support security decisions, the Verifier generates digestable Attestation Results that can be easily consumed by Relying Parties. The RATS architecture specifies the corresponding concepts and terms [I-D.ietf-rats-architecture].

TUDA uses the architectural constituents of the RATS Architecture, such as the roles Attester and Verifier, and defines a method to convey Conceptual Messages between them. TUDA uses the Uni-Directional Remote Attestation interaction model described in [I-D.ietf-rats-reference-interaction-models]. While the Conceptual Message focussed on in this document is RATS Evidence, any type of

Conceptual Message content that requires a believable indication about the message's content freshness can be conveyed with TUDA (e.g. Attestation Results).

The conveyance of Evidence in RATS must ensure that Evidence always remains integrity protected, tamper-evident, originates from a trustable entity (or group of entities), and is accompanied by a proof of its freshness.

In contrast to bi-directional interactions as described by Challenge/Response Remote Attestation in [I-D.ietf-rats-reference-interaction-models], TUDA enables uni-directional conveyance in the interactions between Attester and Verifier. TUDA allows a Verifier to receive Evidence from an Attester without solicitation. Conversely, it allows a Verifier to retrieve Evidence from an Attester without it being generated ad-hoc. Exemplary applications of TUDA are the creation of beacons in vehicular environments [IEEE1609] or authentication mechanisms based on EAP [RFC5247].

The generation of Evidence in RATS requires an Attesting Environment. In this specification, the root of trust acting as an Attesting Environment is a Trusted Platform Module (TPM, see [TPM12] and [TPM2]). The Protected Capabilities [TCGGLOSS] provided by a TPM support various activities in RATS, e.g., Claims collection and Evidence generation.

A trusted coupling of Evidence generation with a global timescale is enabled via a Handle Distributor. Handles generated by a Handle Distributor can include nonces, signed timestamps, or other structured or opaque content used as qualifying data in Evidence generation. In TUDA, all RATS entities, such as the entities taking on the roles of Attester and Verifier, can receive signed timestamps from the Handle Distributor. These trusted timestamps replace nonces in Evidence generation and Evidence appraisal [I-D.ietf-rats-reference-interaction-models].

1.1. Forward Authenticity

Nonces enable an implicit time-keeping in which the freshness of Evidence is inferred by recentness. Recentness is estimated via the time interval between sending a nonce as part of a challenge for Evidence and the reception of Evidence based on that nonce (as outlined in the interaction model depicted in section 8.1 in [I-D.ietf-rats-reference-interaction-models]). Conversely, the omission of nonces in TUDA allows for explicit time-keeping where freshness is not inferred from recentness. Instead, a cryptographic binding of a trusted synchronization to a global timescale in the

Evidence itself allows for Evidence that can prove past operational states of an Attester. To capture and support this concept, this document introduces the term Forward Authenticity.

Forward Authenticity: 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. Forward Authenticity is achieved by timely recording of authenticity Claims from Target Environments (via "audit logs" during "audit sessions") that are authorized for this purpose and trustworthy (via endorsed roots of trusts, for example), 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 requests.

1.2. TUDA Objectives

Time-Based Uni-directional Attestation 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.

1.3. Terminology

This document uses the terms defined in the RATS Architecture [I-D.ietf-rats-architecture] and by the RATS Reference Interaction Models [I-D.ietf-rats-reference-interaction-models].

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. Remote Attestation Principles

Based on the RATS Architecture, the processing of TPM generated Evidence can be separated in three activities.

Evidence Generation: The retrieval of signed digests from an RTR based on a sequence of collected Claims about software component integrity (measurements).

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

Evidence Appraisal: The validation of Evidence signatures as well as the assessment of Claim values in Evidence by comparing them with Reference Values.

TUDA is specified in support of these RATS activities that align with the definitions presented in [PRIRA] and [TCGGLOSS].

2.1. Authenticity of Evidence

Remote attestation Evidence is composed of a set of Claims (assertions about the trustworthiness of an Attester’s Target Environments) that is accompanied by a proof of its veracity - typically a signature based on shielded, private, and potentially use-restricted key material used as an Authentication Secret as specified in section 6 of [I-D.ietf-rats-reference-interaction-models] (or a secure channel as illustrated in [I-D.birkholz-rats-uccs]). As key material alone is typically not self-descriptive with respect to its intended use (its semantics), the Evidence created via TUDA MUST be accompanied by two kinds of certificates that are cryptographically associated with a trust anchor (TA) [RFC4949] via certification paths:

- an Attestation Key (AK) Certificate (AK-Cert) that represents the attestation provenance of the Attesting Environment (see section 4.2. in [I-D.ietf-rats-architecture]) that generates Evidence, and

- an Endorsement Key (EK) Certificate (EK-Cert) that represents the Protection Capabilities of an Attesting Environment the AK is stored in.

If a Verifier decides to trust the TA of both an AK-Cert and an EK-Cert presented by an Attester - and thereby the included Claims about the trustworthiness of an Attester's Target Environments - the Evidence generated by the Attester can be considered trustable and believable. Ultimately, all trustable and believable Evidence MUST be appraised by a Verifier in order to assess the trustworthiness of the corresponding Attester. Assertions represented via Claims MUST NOT be considered believable by themselves.

In this document, Evidence is generated via TPMs that come with an AK-Cert and a EK-Cert as a basis for believable Evidence generation.

2.2. Generating Evidence about Software Component Integrity

Evidence generated by a TPM for TUDA is based on measured hash values of all software components deployed in Target Environments (see section 4.2. in [I-D.ietf-rats-architecture]) before they are executed ("measure then execute"). The underlying concept of "Attestation Logs" is elaborated on in Section 2.4.2. of [I-D.fedorkow-rats-network-device-attestation]. This concept is implemented, for example, in the Linux kernel where it is called the Linux Integrity Measurement Architecture (IMA) [Safford] and used to generates such a sequence of hash values. A representation for conveyance of corresponding event logs is described in the Canonical Event Log [CEL] specification. Open source solutions, for example, based on [RFC5209] use an IMA log to enable remote attestation [Steffens].

An Attester MUST generate such an event/measurement log.

2.3. Measurements and Digests Generated by an Attester

A hash value of a software component is created before it is executed by Attesters. These hash values are typically represented as event log entries referred to as measurements, which often occur in large quantities. Capabilities such as Linux IMA can be used to generate these measurements on an Attester. Measurements are chained by Attesters using a rolling hash function. A TPM acts as a root of trust for storage (RTS) by providing an Extend ([TPM12], [TPM2]) operation to feed hash values in a rolling hash function. Each measurement added to the sequence of all measurements results in a new current digest hash value. A TPM acts as a root of trust for reporting (RTR) by providing Quote ([TPM12], [TPM2]) operations to generate a digest of all currently extended hash values as Evidence.

TUDA requirements on TPM primitive operations and the information elements processed by them are illustrated using pseudo code in Appendix C and D.

2.4. Attesting Environments and Roots of Trust

The primitive operations used to generate an initial set of measurements at the beginning of an Attester's boot sequence MUST be provided by a Root of Trust for Measurement (RTM) that is a system component of the Attester. An RTM MUST be trusted via trust-relationships to TAs enabled by appropriate Endorsements (e.g.,EK-Certs). If a Verifier cannot trust an RTM, measurements based on values generated by the RTM MUST be considered invalid. At least one RTM MUST be accessible to the first Attesting Environment in Attester conducting Layered Attestation (see section 4.3. in [I-D.ietf-rats-architecture]). An RTM MAY aggregate and retain measurements until the first RTS becomes available in a Layered Attestation procedure - instead of feeding measurements into an RTS, instantly. The Protection Capabilities of an RTM to also act as a temporary RTS MUST be trusted via trust-relationships to TAs enabled by appropriate Endorsements. System components supporting the use of a TPM typically include such an appropriate RTM. In general, feeding measurements from an initial RTM into a TPM is automated and separated from Protected Capabilities that provide Claims collection from Target Environments that are regular execution environments. A TPM providing the Protection Capabilities for an isolated and shielded location to feed measurements into (integrity and confidentiality) is an appropriate RTS for TUDA.

The primitive operations used to store and chain measurements via a rolling hash function MUST be provided by an appropriate root of trust for storage (RTS) that is a system component of the Attester. An RTS MUST be trusted via trust-relationships to TAs enabled by appropriate Endorsements (e.g.,EK-Certs). If a Verifier cannot trust an RTS, Evidence generated based on digest values acquired from the RTS MUST be considered invalid. An RTS MUST be accessible to all Attesting Environments that are chained in a Layered Attestation procedure. A TPM providing the primitive operation for Extend is an appropriate RTM for TUDA.

The primitive operations used to generate Evidence based on digests MUST be provided by roots of trust for reporting (RTR) that are system components of the Attester. An RTR MUST be be trusted via trust-relationships to TAs enabled by appropriate Endorsements (e.g.,EK-Certs). If a Verifier cannot trust an RTR, Evidence generated by the RTR MUST be considered invalid. A TPM providing the primitive operations for Quote is an appropriate RTR for TUDA. In a Composite Device (see Section 3.5. in [I-D.ietf-rats-architecture] conducting a Layered Attestation procedure, Attesting Environments MAY not be TPMs. At least one Attesting Environment MUST be a TPM. At least one TPM MUST act as an RTR. Attesting Environments that are not TPMs MUST NOT act as an RTR.

A concise definition of the terms RTM, RTS, and RTR can be found in the Trusted Computing Group (TCG) Glossary [TCGGLOSS]. An RTS and an RTR are often tightly coupled. In TUDA, a Trusted Platform Module (TPM, see [TPM12] and [TPM2]) takes on the roles of an RTS and an RTR. The specification in this document requires the use of a TPM as a component of the Attester. The protocol part of this specification can also be used with other RTS and RTR as long as essential functional requirements are satisfied (e.g., a trusted relative source of time, such as a tick-counter). A sequence of Layered Attestation using at least an RTM, RTS, and RTR enables an authenticated boot sequence typically referred to as Secure Boot.

2.5. Indeterministic Measurements

The sequence of measurements that is extended into the RTS provided by a TPM may not be deterministic due to race conditions that are side-effects of parallelization. Parallelization occurs, for example, between different isolated execution environments or separate software components started in a execution environment. In order to enable the appraisal of Evidence in cases where sequence of measurement varies, a corresponding event log that records all measurements in sequence, such as the IMA log, has to be conveyed next to the Evidence as depicted in section 8.2. in [I-D.ietf-rats-reference-interaction-models].

In contrast to Evidence, event logs do not necessarily have to be integrity protected or tamper-evident. Event logs are conveyed to a Verifier in order to compute the reference values required for comparison with digest values (output of TPM Quote operations). While digest values MUST constitute Evidence, measurements in event logs MAY be part of Evidence, but do not have to be MAY be conveyed separately. If the values in event logs or their sequence are tampered with before or during conveyance from an Attester to a Verifier, the corresponding Evidence Appraisal fails. While this dependency reflects the intended behavior of RATS, integrity protected or tamper-evident can be beneficial or convenient in some usage scenarios. Additionally, event logs my allow insights into the composition of an Attester and typically come with confidentiality requirements.

In order to compute reference values to compare digest Claims in Evidence with, a Verifier MUST be able to replay the rolling hash function of the Extend operation provided by a TPM (see Section 2.4.2. in [I-D.fedorkow-rats-network-device-attestation]).

A Verifier has to replay the event log using its own extend operation with an identical rolling hash function in order to generate reference values as outlined in section 2.4.1. of

[I-D.fedorkow-rats-network-device-attestation]. During reply, the validity of each event log record MUST be appraised individually by the Verifier in order to infer if each started software component satisfies integrity requirements. These appraisal procedures require Reference Integrity Measurements/Manifests (RIM) as are provided via [I-D.birkholz-rats-coswid-rim] or [TCGRIM]. Each RIM includes Reference Values that are nominal reference hash values for sets of software components. The Reference Values can be compared with hash values about executed software components included in an event log. A Verifier requires an appropriate set of RIMs to compare every record in an event log successfully. RIMs or other sets Reverence Value are supplied by Reference Value Providers as defined in the RATS Architecture [I-D.ietf-rats-architecture]. Corresponding procedures that enable a Verifier to acquire Reference Values are out-of-scope of this document.

3. TUDA Principles and Requirements

Traditional remote attestation protocols typically use bi-directional challenge/response interaction models. Examples include the Platform Trust Service protocol [PTS] or CAVES [PRIRA], where one entity sends a challenge that is included inside the response to prove the freshness of Evidence via recentness. The corresponding interaction model depicted in Section 8.1. of [I-D.ietf-rats-reference-interaction-models] tightly couples the three RATS activities of generating, conveying and appraising Evidence.

Time-Based Uni-directional Attestation can decouple these three activities. 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 appraisal of past states,

- secure audit logs by combining the Evidence generated with integrity measurement logs (e.g. IMA logs) that represent a detailed record of corresponding past states,

- the use of the uni-directional interaction model [I-D.ietf-rats-reference-interaction-models] that can traverse "diode-like" network security functions (NSF) or can be leveraged RESTful telemetry as enabled by the CoAP Observe option [RFC7252]).

3.1. Attesting Environment Requirements

An Attesting Environment that generates Evidence in TUDA MUST support three specific Protected 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).

A TPM is capable of providing these Protected Capabilities for TUDA.

3.2. Handle Distributor Requirements: Time Stamp Authority

Both Evidence generation and Evidence appraisal require a Handle Distributor that can take on the role of a trusted Time Stamp Authority (TSA) as an additional third party. Time Stamp Tokens (TST) included in Handles MUST be generated by Time Stamp Authority based on [RFC3161] that acts as the Handle Distributor. The combination of a local source of time provided by a TPM (on the Attester) and the TST provided by the Handle Distributor (to both the Attester and the Verifier) enable an appropriate proof of freshness.

4. Information Elements and Conveyance

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

- MUST be 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 SHOULD only be re-generated when out-dated (not fresh, but stale), which reduces the overall resources required from the Attester, including the usage of a TPM's resources (re-generation of IE is determined by their age or by specific state changes on the Attester, e.g., due to a reboot-cycle)

- SHOULD only be 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 SHOULD be reused for multiple remote attestation procedures in the limits of its corresponding freshness-window, further reducing the load imposed on the Attester and corresponding TPMs.

5. TUDA Core Concept

Traditional Challenge/Response Remote Attestation [I-D.ietf-rats-reference-interaction-models] includes sending a nonce in the challenge to be used in ad-hoc Evidence generation. 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 a response containing evidence, 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 Attester's state by creating Evidence that a certain key is bound to that state. The latter provides proof that the Attester 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 [SFKE2008].

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

As highlighted above, the freshness properties of a challenge-response based protocol enable implicit time-keeping via a time window 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

- o the typical round-trip time between the Verifier and the Attester.

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

- o the time-synchronization between the Attester and the Handle Distributor. The time between the two tickstamps acquired via the RoT define the scope of the maximum drift (time "left" and time "right" in respect to the timeline) to the handle including the signed timestamp, and

- o the drift of clocks included in the 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 Handle Distributor and the RoT used. In consequence, TUDA Evidence can even serve as proof of integrity in audit logs with precise point-in-time guarantees.

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 CoREDONF.

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.

5.1. TPM Specific Terms

PCR: A Platform Configuration Register that is part of the TPM 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

5.2. Certificates

HD-CA: The Certificate Authority that provides the certificate for the TSA role of a Handle Distributor (HD)

AIK-CA: The Certificate Authority that provides the certificate for the AK of the TPM. This is the client platform credential for this protocol. It is a placeholder for a specific CA and AK-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].

6. The TUDA Protocol Family

Time-Based Uni-Directional Attestation consists of the following seven information elements:

Handle Distributor Certificate: The certificate of the Handle Distributor that takes on the role of TSA. The Handle Distributor certificate is used in a subsequent synchronization protocol tokens. This certificate is signed by the HD-CA.

AK Certificate: A certificate about the Attestation Key (AIK) used. An AK-Cert may 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 frame for Evidence is provided by the relative timestanps generated by the TPM. In order to put Evidence 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, trustable timesamps are acquired from a Handle Distributor.

Restriction Info: Evidence Generation relies on the capability of the Rot to operate on restricted keys. Whenever the PCR values of an Attesting Environment 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 RoT are used. The TPM creates a signed certificate using the AK about the newly created restricted key.

Measurement Log: A Verifier requires the means to derive the PCRs' values in order to appraise the trustworthiness of an Attester. As such, a list of those elements that were extended into the PCRs is reported. 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 Evidence: The actual Evidence is then based on a signed timestamp provided by the RoT using the restricted temporary key that was certified in the steps above. The signed timestamp generated provides the trustable assertion that at this point in time (with respect to the relative time of the TPM s tick counter) a certain configuration existed (namely the PCR values associated with the restricted key). In combination 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, sometimes called golden measurements, known-good-values, or nominal values) of all started software components to compare them with the entries in a 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 collection (a RIM Manifest [I-D.birkholz-rats-coswid-rim]).

These information elements can 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, 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 certificate provisioning completely in order to save bandwidth and computation time for certificate verification.

6.1. TUDA Information Elements Update Cycles

An Attester can be in various states during its uptime cycles. For TUDA, a subset of these states (which imply associated information) are important to the Evidence Generation. The specific states defined are:

- persistent, even after a hard reboot: includes certificates that are associated with the endpoint itself or with services it relies on.

- volatile to a degree: may change at the beginning of each boot cycle. This includes the capability of a TPM to provide relative time which provides the basis for the synchronization token and implicit attestation - and which can reset after an Attester is powered off.

- very volatile: can change during any time of an uptime cycle (periods of time an Attester is powered on, starting with its boot sequence). 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 fresh Evidence to be generated:

- The Attester completes a boot-cycle

- A relevant PCR changes

- Too much time has passed since the Evidence Generation

The third event listed above is variable per application use case and also depends on the precision of the clock included in the RoT. For usage scenarios, in which the Attester would periodically push information to be used in an audit-log, a time-frame of approximately one update per minute should be sufficient. For those usage scenarios, where Verifiers request (pull) fresh Evidence, an implementation could potentially use a TPM continuously to always present the most freshly created Evidence. This kind of utilization can result in a bottle-neck with respect to other purposes: if unavoidable, a periodic interval of once per ten seconds is recommended, which typically leaves about 80% of available TPM resource for other applications.

The following diagram is based on the reference interaction model found in section 8.1. of [I-D.ietf-rats-reference-interaction-models] and is enriched with the IE update cycles defined in this section.

```
.----------.                                                .----------.
| Attester |                                                | Verifier |
'----------'                                                '----------'
     |                                                            |
   boot                                                           |
     |                                                            |
valueGeneration(targetEnvironment)                                |
     | => claims                                                  |
     |                   .--------------------.                   |
     | <----------handle |                    |                   |
     |                   | Handle Distributor |                   |
     |                   |                    | handle----------> |
     |                   '--------------------'                   |
syncTokenGeneration                                               |
     | => syncToken                                               |
     |                                                            |
restrictedKeyGeneration                                           |
restrictedKeyCertify                                              |
     |                                                            |
evidenceGeneration(handle, authSecIDs, collectedClaims)           |
     | => evidence                                                |
     |                                                            |
     | pushAKCert ----------------------------------------------> |
     | pushSyncToken -------------------------------------------> |
     | pushCertifyInfo -----------------------------------------> |
     | pushEventLog --------------------------------------------> |
     | pushEvidenceon ------------------------------------------> |
     |                                                            |
     |                  evidenceAppraisal(evidence, eventLog, refClaims)
     |                                       attestationResult <= |
     ~                                                            ~
  pcr-change                                                      |
     |                                                            |
restrictedKeyGeneration                                           |
restrictedKeyCertify                                              |
     |                                                            |
evidenceGeneration(handle, authSecIDs, collectedClaims)           |
     | => evidence                                                |
     |                                                            |
     | pushCertifyInfo -----------------------------------------> |
     | pushEventLog --------------------------------------------> |
     | pushEvidenceon ------------------------------------------> |
     |                                                            |
     |                  evidenceAppraisal(evidence, eventLog, refClaims)
     |                                       attestationResult <= |
     |                                                            |
```

Figure 1: Example sequence of events

7. 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.

- 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.

8. IANA Considerations

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

TBD

9. Security Considerations

There are Security Considerations. TBD

10. Contributors

TBD

11. References

11.1. Normative References

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

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/info/rfc3161>.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, DOI 10.17487/RFC3411, December 2002, <https://www.rfc-editor.org/info/rfc3411>.

[RFC5209] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <https://www.rfc-editor.org/info/rfc5209>.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <https://www.rfc-editor.org/info/rfc5247>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7320] Nottingham, M., "URI Design and Ownership", RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[RFC8640] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Dynamic Subscription to YANG Events and Datastores over NETCONF", RFC 8640, DOI 10.17487/RFC8640, September 2019, <https://www.rfc-editor.org/info/rfc8640>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

11.2. Informative References

[AIK-Credential]
TCG Infrastructure Working Group, "TCG Credential Profile", 2007, <https://www.trustedcomputinggroup.org/wp-content/uploads/IWG-Credential_Profiles_V1_R1_14.pdf>.

[AIK-Enrollment]
TCG Infrastructure Working Group, "A CMC Profile for AIK Certificate Enrollment", 2011, <https://www.trustedcomputinggroup.org/wp-content/uploads/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf>.

[CEL] TCG TNC Working Group, "DRAFT Canonical Event Log Format Version: 1.0, Revision: .12", 2018.

[I-D.birkholz-rats-coswid-rim]
Birkholz, H., Uiterwijk, P., Smith, N., Fitzgerald-McKay, J., Waltermire, D., and S. Bhandari, "Reference Integrity Measurement Extension for Concise Software Identities", draft-birkholz-rats-coswid-rim-01 (work in progress), July 2020.

[I-D.birkholz-rats-uccs]
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", draft-birkholz-rats-uccs-02 (work in progress), December 2020.

[I-D.fedorkow-rats-network-device-attestation]
Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", draft-fedorkow-rats-network-device-attestation-05 (work in progress), April 2020.

[I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface (CORECONF)", draft-ietf-core-comi-10 (work in progress), July 2020.

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-08 (work in progress), December 2020.

[I-D.ietf-rats-reference-interaction-models]
Birkholz, H., Eckel, M., Newton, C., and L. Chen, "Reference Interaction Models for Remote Attestation Procedures", draft-ietf-rats-reference-interaction-models-01 (work in progress), October 2020.

[I-D.ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-16 (work in progress), November 2020.

[I-D.ietf-sacm-terminology]
Birkholz, H., Lu, J., Strassner, J., Cam-Winget, N., and A. Montville, "Security Automation and Continuous Monitoring (SACM) Terminology", draft-ietf-sacm-terminology-16 (work in progress), December 2018.

[IEEE1609]
IEEE Computer Society, "1609.4-2016 - IEEE Standard for Wireless Access in Vehicular Environments (WAVE) -- Multi-Channel Operation", IEEE Std 1609.4, 2016.

[IEEE802.11P]
IEEE Computer Society, "802.11P-2010 - IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments", IEEE Std 802.11P, 2010.

[IEEE802.1AR]
IEEE Computer Society, "802.1AR-2009 - IEEE Standard for Local and metropolitan area networks - Secure Device Identity", IEEE Std 802.1AR, 2009.

[PRIRA] Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., O'Hanlon, B., Ramsdell, J., Segall, A., Sheehy, J., and B. Sniffen, "Principles of Remote Attestation", Springer International Journal of Information Security, Vol. 10, pp. 63-81, DOI 10.1007/s10207-011-0124-7, April 2011.

[PTS] TCG TNC Working Group, "TCG Attestation PTS Protocol Binding to TNC IF-M", 2011, <https://www.trustedcomputinggroup.org/wp-content/uploads/IFM_PTS_v1_0_r28.pdf>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, DOI 10.17487/RFC1213, March 1991, <https://www.rfc-editor.org/info/rfc1213>.

[RFC2790] Waldbusser, S. and P. Grillo, "Host Resources MIB", RFC 2790, DOI 10.17487/RFC2790, March 2000, <https://www.rfc-editor.org/info/rfc2790>.

[RFC3418] Presuhn, R., Ed., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, DOI 10.17487/RFC3418, December 2002, <https://www.rfc-editor.org/info/rfc3418>.

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

[RFC6933] Bierman, A., Romascanu, D., Quittek, J., and M. Chandramouli, "Entity MIB (Version 4)", RFC 6933, DOI 10.17487/RFC6933, May 2013, <https://www.rfc-editor.org/info/rfc6933>.

[Safford] Safford, D., Zohar, M., and R. Sailer, "Using IMA for Integrity Measurement and Attestation", Linux Plumbers Conference 2009 , 2009.

[SCALE] Fuchs, A., "Improving Scalability for Remote Attestation", Master Thesis (Diplomarbeit), Technische Universitaet Darmstadt, Germany, 2008.

[SFKE2008]
Stumpf, F., Fuchs, A., Katzenbeisser, S., and C. Eckert, "Improving the scalability of platform attestation", ACM Proceedings of the 3rd ACM workshop on Scalable trusted computing - STC ’08 , page 1-10, DOI 10.1145/1456455.1456457, 2008.

[STD62] "Internet Standard 62", STD 62, RFCs 3411 to 3418, December 2002.

[Steffens]
Steffen, A., "The linux Integrity Measurement Architecture and TPM-based Network Endpoint Assessment", Linux Security Summit , 2012.

[TCGGLOSS]
TCG, "TCG Glossary", 2012, <https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_Glossary_Board-Approved_12.13.2012.pdf>.

[TCGRIM] TCG, "TCG Reference Integrity Manifest (RIM) Information Model", 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_RIM_Model_v1-r13_2feb20.pdf>.

[TEE] Global Platform, "TEE System Architecture v1.1, GPD_SPE_009", 2017.

[TPM12] "Information technology -- Trusted Platform Module -- Part 1: Overview", ISO/IEC 11889-1, 2009.

[TPM2] Trusted Computing Group, "Trusted Platform Module Library Specification, Family 2.0, Level 00, Revision 01.16 ed.", 2014.

Appendix A. REST Realization

Each of the seven data items is defined as a media type (Section 8). 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:

Group/Table	Cycle	Instance	Fragment	Required
General				x
AIKCert	x	x	x	
TSACert	x	x	x	
SyncToken	x		x	x
Restrict	x			x
Measure	x	x		
VerifyToken	x			x
SWIDTag	x	x	x	

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. accomodation 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
    LAST-UPDATED    "202101130000Z" --  13 January 2021
    ORGANIZATION
        "Fraunhofer SIT"
    CONTACT-INFO
        "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 TZI
        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 (2021)."
```

```
REVISION "202101130000Z" -- 13 January 2021
DESCRIPTION
    "Twelfth version, published as draft-birkholz-rats-tuda-04."

REVISION "202007130000Z" -- 13 July 2020
DESCRIPTION
    "Eleventh version, published as draft-birkholz-rats-tuda-03."

REVISION "202003090000Z" -- 09 March 2020
DESCRIPTION
    "Tenth version, published as draft-birkholz-rats-tuda-02."

REVISION "201909110000Z" -- 11 September 2019
DESCRIPTION
    "Ninth version, published as draft-birkholz-rats-tuda-01."

REVISION "201903120000Z" -- 12 March 2019
DESCRIPTION
    "Eighth version, published as draft-birkholz-rats-tuda-00."

REVISION "201805030000Z" -- 03 May 2018
DESCRIPTION
    "Seventh version, published as draft-birkholz-i2nsf-tuda-03."

REVISION "201805020000Z" -- 02 May 2018
DESCRIPTION
    "Sixth version, published as draft-birkholz-i2nsf-tuda-02."

REVISION "201710300000Z" -- 30 October 2017
DESCRIPTION
    "Fifth version, published as draft-birkholz-i2nsf-tuda-01."

REVISION "201701090000Z" -- 09 January 2017
DESCRIPTION
    "Fourth version, published as draft-birkholz-i2nsf-tuda-00."

REVISION "201607080000Z" -- 08 July 2016
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."
        ::= { tudaV1AIKCert 2 }

    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
           occurred (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.

           DEFVAL intentionally omitted - index object."
       ::= { 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."
        ::= { tudaV1Measure 3 }

    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."
        ::= { tudaV1SWIDTag 2 }

    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,
           tudaV1RestrictCycles,
           tudaV1RestrictData,
           tudaV1VerifyTokenCycles,
           tudaV1VerifyTokenData
       }
       STATUS  current
       DESCRIPTION
           "The basic mandatory TUDA MIB objects."
       ::= { tudaV1ObjectGroups 1 }
```

```
   tudaV1OptionalGroup OBJECT-GROUP
       OBJECTS {
           tudaV1AIKCertCycles,
           tudaV1AIKCertData,
           tudaV1TSACertCycles,
           tudaV1TSACertData,
           tudaV1MeasureCycles,
           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 {

  namespace "urn:ietf:params:xml:ns:yang:smiv2: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 TZI
    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 (2021).";

  revision "2021-01-13" {
    description
     "Twelfth version, published as draft-birkholz-rats-tuda-04.";
    reference
     "draft-birkholz-rats-tuda-04";
  }
  revision "2020-07-13" {
    description
     "Eleventh version, published as draft-birkholz-rats-tuda-03.";
    reference
     "draft-birkholz-rats-tuda-03";
  }
  revision "2020-03-09" {
    description
     "Tenth version, published as draft-birkholz-rats-tuda-02.";
    reference
     "draft-birkholz-rats-tuda-02";
  }
  revision "2019-09-11" {
    description
     "Ninth version, published as draft-birkholz-rats-tuda-01.";
    reference
     "draft-birkholz-rats-tuda-01";
  }
  revision "2019-03-12" {
    description
     "Eighth version, published as draft-birkholz-rats-tuda-00.";
    reference
     "draft-birkholz-rats-tuda-00";
  }
  revision "2018-05-03" {
```

```
    description
     "Seventh version, published as draft-birkholz-i2nsf-tuda-03.";
    reference
     "draft-birkholz-i2nsf-tuda-03";
  }
  revision "2018-05-02" {
    description
     "Sixth version, published as draft-birkholz-i2nsf-tuda-02.";
    reference
     "draft-birkholz-i2nsf-tuda-02";
  }
  revision "2017-10-30" {
    description
     "Fifth version, published as draft-birkholz-i2nsf-tuda-01.";
    reference
     "draft-birkholz-i2nsf-tuda-01";
  }
  revision "2017-01-09" {
    description
     "Fourth version, published as draft-birkholz-i2nsf-tuda-00.";
    reference
     "draft-birkholz-i2nsf-tuda-00";
  }
  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";
        }
        config false;
        description
         "High-order index of this TSA Certificate fragment.
          Index of a TSA Certificate chain update cycle that has
          occurred (bounded by the value of tudaV1TSACertCycles).

          DEFVAL intentionally omitted - index object.";
      }

      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";
        }
        config false;
        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.";
      }

      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.";
      }
    }

    container tudaV1TrapV2Cycles-tudaV1SWIDTagCycles {
      description
       "TPD"
      leaf tudaV1SWIDTagCycles {
        type yang:counter32;
        description
         "Count of SWID Tag update cycles that have occurred.

          DEFVAL intentionally omitted - counter object.";
      }
    }

  }
}
```

<CODE ENDS>

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: 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'0000000000000000000000000000000000000000' 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:

- o 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*/)

  ts = 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:

- o PCR list,
- o the newly created restricted public key, and
- o 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:

```
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:

```
| 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 6.1), 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

```
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_whitelist(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 == */
| ts.currentTicks.tickRate = ts_left.currentTicks.tickRate
| ts.currentTicks.tickNonce = ts_left.currentTicks.tickNonce
|
| verifySig(restrictedKey_pub, dummyNonce || dummyDigest || ts)
|
| 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:

- o hash() : description TBD
- o sig() : description TBD
- o 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 TZI
Bibliothekstr. 1
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

RATS Working Group H. Birkholz
Internet-Draft Fraunhofer SIT
Intended status: Standards Track J. O'Donoghue
Expires: 9 September 2021 Qualcomm Technologies Inc.
N. Cam-Winget
Cisco Systems
C. Bormann
Universitaet Bremen TZI
8 March 2021

A CBOR Tag for Unprotected CWT Claims Sets
draft-birkholz-rats-uccs-03

Abstract

CBOR Web Token (CWT, RFC 8392) Claims Sets sometimes do not need the protection afforded by wrapping them into COSE, as is required for a true CWT. This specification defines a CBOR tag for such unprotected CWT Claims Sets (UCCS) and discusses conditions for its proper use.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

A CBOR Web Token (CWT) as specified by [RFC8392] is always wrapped in a CBOR Object Signing and Encryption (COSE, [RFC8152]) envelope. COSE provides -- amongst other things -- the integrity protection mandated by RFC 8392 and optional encryption for CWTs. Under the right circumstances, though, a signature providing proof for authenticity and integrity can be provided through the transfer protocol and thus omitted from the information in a CWT without compromising the intended goal of authenticity and integrity. If a mutually Secured Channel is established between two remote peers, and if that Secure Channel provides the required properties (as discussed below), it is possible to omit the protection provided by COSE, creating a use case for unprotected CWT Claims Sets. Similarly, if there is one-way authentication, the party that did not authenticate may be in a position to send authentication information through this channel that allows the already authenticated party to authenticate the other party.

This specification allocates a CBOR tag to mark Unprotected CWT Claims Sets (UCCS) as such and discusses conditions for its proper use in the scope of Remote ATtestation procedureS (RATS) and the conveyance of Evidence from an Attester to a Verifier.

This specification does not change [RFC8392]: A true CWT does not make use of the tag allocated here; the UCCS tag is an alternative to using COSE protection and a CWT tag. Consequently, in a well-defined scope, it might be acceptable to use the contents of a CWT without its COSE container and tag it with a UCCS CBOR tag for further processing -- or to use the contents of a UCCS CBOR tag for building a CWT to be signed by some entity that can vouch for those contents.

1.1. Terminology

The term Claim is used as in [RFC8725].

The terms Claim Key, Claim Value, and CWT Claims Set are used as in [RFC8392].

The terms Attester, Attesting Environment and Verifier are used as in [I-D.ietf-rats-architecture].

UCCS: Unprotected CWT Claims Set(s); CBOR map(s) of Claims as defined by the CWT Claims Registry that are composed of pairs of Claim Keys and Claim Values.

Secure Channel: A protected communication channel between two peers that can ensure the same qualities associated for UCCS conveyance as CWT conveyance without any additional protection.

All terms referenced or defined in this section are capitalized in the remainder of this document.

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. Motivation and Requirements

Use cases involving the conveyance of Claims, in particular, remote attestation procedures (RATS, see [I-D.ietf-rats-architecture]) require a standardized data definition and encoding format that can be transferred and transported using different communication channels. As these are Claims, [RFC8392] is a suitable format. However, the way these Claims are secured depends on the deployment, the security capabilities of the device, as well as their software stack. For example, a Claim may be securely stored and conveyed using a device’s Trusted Execution Environment (TEE, see [I-D.ietf-teep-architecture]) or especially in some resource constrained environments, the same process that provides the secure communication transport is also the delegate to compose the Claim to be conveyed. Whether it is a transfer or transport, a Secure Channel is presumed to be used for conveying such UCCS. The following sections further describe the RATS usage scenario and corresponding requirements for UCCS deployment.

3. Characteristics of a Secure Channel

A Secure Channel for the conveyance of UCCS needs to provide the security properties that would otherwise be provided by COSE for a CWT. In this regard, UCCS is similar in security considerations to JWTs [RFC8725] using the algorithm "none". RFC 8725 states: "if a JWT is cryptographically protected end-to-end by a transport layer, such as TLS using cryptographically current algorithms, there may be no need to apply another layer of cryptographic protections to the JWT. In such cases, the use of the "none" algorithm can be perfectly acceptable.". Analogously, the considerations discussed in Sections 2.1, 3.1, and 3.2 of RFC 8725 apply to the use of UCCS as elaborated on in this document.

Secure Channels are often set up in a handshake protocol that mutually derives a session key, where the handshake protocol establishes the authenticity of one of both ends of the communication. The session key can then be used to provide confidentiality and integrity of the transfer of information inside the Secure Channel. A well-known example of a such a Secure Channel setup protocol is the TLS [RFC8446] handshake; the TLS record protocol can then be used for secure conveyance.

As UCCS were initially created for use in Remote ATtestation procedureS (RATS) Secure Channels, the following subsection provides a discussion of their use in these channels. Where other environments are intended to be used to convey UCCS, similar considerations need to be documented before UCCS can be used.

3.1. UCCS and Remote ATtestation procedureS (RATS)

For the purposes of this section, the Verifier is the receiver of the UCCS and the Attester is the provider of the UCCS.

Secure Channels can be transient in nature. For the purposes of this specification, the mechanisms used to establish a Secure Channel are out of scope.

As a minimum requirement in the scope of RATS Claims, the Verifier MUST authenticate the Attester as part of the establishment of the Secure Channel. Furthermore, the channel MUST provide integrity of the communication from the Attester to the Verifier. If confidentiality is also required, the receiving side needs to be be authenticated as well, i.e., the Verifier and the Attester SHOULD mutually authenticate when establishing the Secure Channel.

The extent to which a Secure Channel can provide assurances that UCCS originate from a trustworthy attesting environment depends on the characteristics of both the cryptographic mechanisms used to establish the channel and the characteristics of the attesting environment itself.

A Secure Channel established or maintained using weak cryptography may not provide the assurance required by a relying party of the authenticity and integrity of the UCCS.

Ultimately, it is up to the Verifier's policy to determine whether to accept a UCCS from the Attester and to the type of Secure Channel it must negotiate. While the security considerations of the cryptographic algorithms used are similar to COSE, the considerations of the secure channel should also adhere to the policy configured at each of the Attester and the Verifier. However, the policy controls and definitions are out of scope for this document.

Where the security assurance required of an attesting environment by a relying party requires it, the attesting environment may be implemented using techniques designed to provide enhanced protection from an attacker wishing to tamper with or forge UCCS. A possible approach might be to implement the attesting environment in a hardened environment such as a TEE [I-D.ietf-teep-architecture] or a TPM [TPM2].

When UCCS emerge from the Secure Channel and into the Verifier, the security properties of the Secure Channel no longer apply and UCCS have the same properties as any other unprotected data in the Verifier environment. If the Verifier subsequently forwards UCCS, they are treated as though they originated within the Verifier.

As with EATs nested in other EATs (Section 3.12.1.2 of [I-D.ietf-rats-eat]), the Secure Channel does not endorse fully formed CWTs transferred through it. Effectively, the COSE envelope of a CWT shields the CWT Claims Set from the endorsement of the Secure Channel. (Note that EAT might add a nested UCCS Claim, and this statement does not apply to UCCS nested into UCCS, only to fully formed CWTs)

3.2. Privacy Preserving Channels

A Secure Channel which preserves the privacy of the Attester may provide security properties equivalent to COSE, but only inside the life-span of the session established. In general, a Verifier cannot correlate UCCS received in different sessions from the same attesting environment based on the cryptographic mechanisms used when a privacy preserving Secure Channel is employed.

In the case of a Remote Attestation, the attester must consider whether any UCCS it returns over a privacy preserving Secure Channel compromises the privacy in unacceptable ways. As an example, the use of the EAT UEID [I-D.ietf-rats-eat] Claim in UCCS over a privacy preserving Secure Channel allows a verifier to correlate UCCS from a single attesting environment across many Secure Channel sessions. This may be acceptable in some use-cases (e.g. if the attesting environment is a physical sensor in a factory) and unacceptable in others (e.g. if the attesting environment is a device belonging to a child).

4. IANA Considerations

In the registry [IANA.cbor-tags], IANA is requested to allocate the tag in Table 1 from the FCFS space, with the present document as the specification reference.

Tag	Data Item	Semantics
TBD601	map	Unprotected CWT Claims Set [RFCthis]

Table 1: Values for Tags

5. Security Considerations

The security considerations of [RFC7049] and [RFC8392] apply.

Section 3 discusses security considerations for Secure Channels, in which UCCS might be used. This documents provides the CBOR tag definition for UCCS and a discussion on security consideration for the use of UCCS in Remote ATtestation procedureS (RATS). Uses of UCCS outside the scope of RATS are not covered by this document. The UCCS specification - and the use of the UCCS CBOR tag, correspondingly - is not intended for use in a scope where a scope-specific security consideration discussion has not been conducted, vetted and approved for that use.

5.1. General Considerations

Implementations of Secure Channels are often separate from the application logic that has security requirements on them. Similar security considerations to those described in [I-D.ietf-cose-rfc8152bis-struct] for obtaining the required levels of assurance include:

* Implementations need to provide sufficient protection for private or secret key material used to establish or protect the Secure Channel.

* Using a key for more than one algorithm can leak information about the key and is not recommended.

* An algorithm used to establish or protect the Secure Channel may have limits on the number of times that a key can be used without leaking information about the key.

The Verifier needs to ensure that the management of key material used establish or protect the Secure Channel is acceptable. This may include factors such as:

* Ensuring that any permissions associated with key ownership are respected in the establishment of the Secure Channel.

* Cryptographic algorithms are used appropriately.

* Key material is used in accordance with any usage restrictions such as freshness or algorithm restrictions.

* Ensuring that appropriate protections are in place to address potential traffic analysis attacks.

5.2. AES-CBC_MAC

* A given key should only be used for messages of fixed or known length.

* Different keys should be used for authentication and encryption operations.

* A mechanism to ensure that IV cannot be modified is required.

[I-D.ietf-cose-rfc8152bis-algs], Section 3.2.1 contains a detailed explanation of these considerations.

5.3. AES-GCM

* The key and nonce pair are unique for every encrypted message.

* The maximum number of messages to be encrypted for a given key is not exceeded.

[I-D.ietf-cose-rfc8152bis-algs], Section 4.1.1 contains a detailed explanation of these considerations.

5.4. AES-CCM

* The key and nonce pair are unique for every encrypted message.

* The maximum number of messages to be encrypted for a given block cipher is not exceeded.

* The number of messages both successfully and unsuccessfully decrypted is used to determine when rekeying is required.

[I-D.ietf-cose-rfc8152bis-algs], Section 4.2.1 constains a detailed explanation of these considerations.

5.5. ChaCha20 and Poly1305

* The nonce is unique for every encrypted message.

* The number of messages both successfully and unsuccessfully decrypted is used to determine when rekeying is required.

[I-D.ietf-cose-rfc8152bis-algs], Section 4.3.1 contains a detailed explanation of these considerations.

6. References

6.1. Normative References

[IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", <http://www.iana.org/assignments/cbor-tags>.

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

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

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

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

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>.

[TPM2] "Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59 ed., Trusted Computing Group", 2019.

6.2. Informative References

[I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>.

[I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>.

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work

in Progress, Internet-Draft, draft-ietf-rats-architecture-10, 9 February 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-10.txt>.

[I-D.ietf-rats-eat]
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-09, 7 March 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-eat-09.txt>.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-14, 22 February 2021, <https://www.ietf.org/archive/id/draft-ietf-teep-architecture-14.txt>.

Appendix A. Example

The example CWT Claims Set from Appendix A.1 of [RFC8392] can be turned into an UCCS by enclosing it with a tag number TBD601:

```
<TBD601>(
  {
    / iss / 1: "coap://as.example.com",
    / sub / 2: "erikw",
    / aud / 3: "coap://light.example.com",
    / exp / 4: 1444064944,
    / nbf / 5: 1443944944,
    / iat / 6: 1443944944,
    / cti / 7: h'0b71'
  }
)
```

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough
GU14 7LS
United Kingdom

Email: jodonogh@qti.qualcomm.com

Nancy Cam-Winget
Cisco Systems
3550 Cisco Way
San Jose, CA 95134
United States of America

Email: ncamwing@cisco.com

Carsten Bormann
Universitaet Bremen TZI
Bibliothekstrasse 1
28369 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.de

RATS
Internet-Draft
Intended status: Informational
Expires: July 31, 2021
M. Chen
Li. Su
China Mobile
January 27, 2021

Use Cases for RATS
draft-chen-rats-usecase-03

Abstract

This document presents three scenarios from the Internet Service Providers' perspective as an supplement use case of the RATS work group. And make some discussions of access authentication, application authentication and trusted link. The requirements of trusted link is put forward to establish a protecttive network connection, thus ensure the native network security.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

At present, it is necessary to complete the authentication before accessing the operator's network to obtain the service. RATS aimed at the solutions to provide interoperable way for domain-specific attestation mechanisms, within RATS relying party may not to maintain the authentication background, as an ISP what may be involved at the level of access authentication is preshared secret keys based authentication, the authentication based on PSK(Preshared secret keys) is different from identity-based authentication, such as IBC(Identity-Based Cryptograph).

After access to the network, operators can also provide application layer authentication services for a variety of applications. At present, there are many application layer authentication methods, it can be divided into certificate-based and non-certificate-based certification systems, so there are the following situations. One application authenticated by certificate-based PKI system may request resource access to a server or service, but the server or service's authentication function is based on identity which is belong to non-certificate-based certification systems. These are all possible future demand scenarios, also in the context of the RATS. Due to limitation of resource, many companies are unable to operate their own certification and willing to rely on the result from operator to reduce their cost, and operator can provide authentication services. Multiple certification centers would be made due to different kinds of request from service and perspective of deployment, before obtaining a certification center's service, certification center need proof for identification, including software and hardware health information. These certification centers are based on regions then there have manage barriers, how can clients from a certification center asstest themselves' identities to another certification center. Especially now there are more virtual resources, cloud resources, one need to prove whether it has access to the resources and can protect the data. From an internal business perspective, how to integrate resources, achieve cross-domain trust and break down management barriers in order to streamline and improve flexibility will also be something rats[I-D.ietf-rats-architecture] can do.

AS Communication Technology and Internet Technology are converging, especially mobile communication network have stepped into 5G era, besides 5G network slice safety needs attention, basic routing is also need to pay much more attention since damage points of routing nodes will affect the security of the whole link. In some scenarios we need to form a trusted routing link. in the internet draft draft-voit-rats-trustworthy-path-routing-00 also have mentioned this problem.

A trusted link means that every device on the link is proven to be trusted dymaticly. In the real world, a new device or a small network is need to add into the core network, newly added associated equipments are required Security and Trustworthy. After the formation of deterministic networks, three problems need to be solved: how to dynamically check the security of equipment, how to dynamically select the best route based on business requirements, how to ensure computing and security capabilities.

2. Terminology

The readers should be familiar with the terms defined in.

In addition, this document makes use of the following terms:

PSK: Preshared secret keys means keys are shared in advance between the authentication parties.

IBC: Identity-Based Cryptograph, it is an asymmetric public key cryptosystem.

PKI: Public Key Infrastructure, an infrastructure built with a public-key mechanism.

3. Use Cases

This section describes use cases which happens inside an ISP.

3.1. Access authentication based on different method

This section considers the level of access authentication. For operators, the access of users is usually based on preshared secret keys, preset with symmetric secret keys before the release. The first access only needs to be activated, and subsequent authentication uses PSK to complete data protection which is based on Symmetric secret key system. In addition, there are other identity-based authentication methods, the access authentication based on identity is asymmetric and the identity is the public key, this approach makes it easier for the peer to obtain the public key of the other peer.

In short, these are two different authentication methods. When a psk-based authentication device needs to request an identity-based service, it needs to prove its' trustworthiness to the other party and the whole process need to ensure the confidentiality of evidences and attestation results.

```
              &#65288;relying party1)              (relying party2)
              +----------------+              +----------------+
              |PSK Auth_Center|              |IBC Auth_Center|
              +----------------+              +----------------+
                     |\          +------------//        |
                     |Evidence  /Attestation           |
        Attestation  |         /    Result             |
            Result  \|        /                        |
           +--------------------+      +--------------------+
          |App/SIMCard/IoTCard|        |App/SIMCard/IoTCard|
         +--------------------+          +--------------------+
               (attester)
```

Figure 1: different access authentication methods within RATS

The format and content of the evidence: TBD

The format of the Attestation Result: TBD

The transmission protocol for evidence or attestation result: TBD

3.2. Application authentication based on different system

At the application level, due to limitation of resource, many applications need operators to provide business authentication services. At present, there are two business authentication methods: one is certification-based PKI system authentication, because the management of certificates is always a very big problem, so the other is non-certificated, such as identity-based authentication whose identity is readable.

When cross-business authentication is required, how to prove one's identity to the other will be a common problem.

```
         (relying party1)                           (relying party2)
+------------------------------------+     +------------------------------------+
|Application authentication platform|     |Application authentication platform|
|     based on certificate          |-----|      based on non-certificate      |
+------------------------------------+     +------------------------------------+
                  |        {Attestation}  /|                  |
                  |        {    Result } /                    |
                  |           ----------                      |
                  |         /                                 |
          +----------------+                          +----------------+
          |  application   |                          |  application   |
          +----------------+                          +----------------+
               (attester1)                                 (attester2)
```

Figure 2: different application authentication methods in RATS architecture

The format and content of the evidence: TBD

The format of the Attestation Result: TBD

The transmission protocol for evidence or attestation result: TBD

Certification-based authentication process: TBD

Identity-based authentication process: TBD

3.3. virtualization-based systems

Cloud computing and other virtualized environment also need remote attestation, one service offered through cloud computing is Infrastructure as a Service (IaaS), which provides virtualized computing resources to enterprises, typically over the Internet. The virtualization platform or virtualization system needs to provide evidence to the user or third party, the process may involve vTPM, which support for establishing trust in a virtualized environment, especially remote verification of software integrity.

3.4. Use case based on trusted routing

5G provides security slices based on routing security, routing devices can be hijacked because of vulnerabilities, damaged equipment could be monitored, so ISP need to be able to dynamically retrieve the status of routing devices, according to the state of the devices dynamicly form a safety link, RATS needs to be used in this case. There are two scenarios for this case: a trusted link is formed within a single operator and a trusted link is formed across operators. The default prerequisite is routing devices support TPM or other relevant standard.

```
                 +----------------------------------+
                 |                                  |
                 |  Trusted Identification System   |
                 |                                  |
                 +-------- -------^--+--------------+
                                  |  |
                                  |  |
                                  |  |
              Device Conditions   |  |  Latest trusted status
               (evidence)or       |  |
               (Appraisal results)|  |
                                  |  |
   +-----------------+            |  |          +-------------------+
   |                 |            |  |          |                   |
   | Routing Device  +------------+  +---------->   Orchestrator    |
   |                 |                          |                   |
   +-----------------+                          +--------+----------+
                                                         |
                                                         v
                                               Form a routing path
     Figure 3: a trusted link is formed within an ISP in RATS

            ISP A                                         ISP B
   +-------------------+                           +--------------------+
   |                   |      Appraisal results    |                    |
   |  Trusted Path     +--------------------------^+   Trusted Path     |
   |                   <---------------------------+                    |
   +-------------------+ cross-domain trusted link+---------------------+

     Figure 4: a trusted link is formed between ISPs in RATS
```

4. Requirements of trusted link

From the operator’s point of view, a more secure link capability will be more competitive for external service. Therefore, Operators attach great importance to the innate security of links, the links innate security highly relied on each networking device that is the routing equipment.

4.1. New equipment into the network

How to add a new device to the network without the consideration of trustworthiness? new routing devices go through four steps before they are actually added to the network, step 1: manually configure the IP address; step 2: discovery device by broadcast protocol; step 3: Verify the identity of the device; step 4: Device Manager issues routing configuration policies. After completing these four steps, the route neighbor is formed.

```
           +------------+
   +-------+Orchestrator+---------+
   |       +----+--+----+         |
   |            |  ^              |
   |       step2|  |step3         |
   |       step4|  |              |
   |            v  |              |
+--+-+         ++--++          +--+-+
| RT |         | RT |step1     | RT |
+----+         +----+          +----+
                NEW
```

Figure 5-1: a new router added to network

How to add a new device to the network with the consideration of devices’ trustworthiness? A trusted link has been formed by default, when a new equipment apply to join the network, new router should provide evidences to the verifier, the orchestrator play the role of verifier, and feed back the attestation result back to the new router, it depends on the implementation. Provision of evidence and trustworthiness assessment actually happens in the Step3 which described in the figure above. After complete the trustworthiness assessment, the orchestrator forms routing policies based on the trustworthiness and issues them, the trusted link establishment is complete.

```
          +-------------------------------------+
          |                                     |
          | +------------+                      |
    Step3 | |            |       Result +----+  |
          | |Ochestrator +<-------------> RT |  |
          | +-----+------+  evidence    +----+  |
          |       |                       NEW   |
          +-------------------------------------+
                  |
   +--------------+---------------------------+
   |  +----+         +----+          +----+   |
   |  | RT +---------+ RT +----------+ RT |   |
   |  +----+   1     +----+     2    +----+   |
   +------------------------------------------+
               Trusted link
```

Figure 5-2: a new router added to network

4.2. Device status updating

How to maintain the freshness of trusted links for the network is always under threat of attack when need to form a trusted link to provide to the user for transmission. The Ochestrator can collect routing information in real time or periodically, including device information, log information, fault information, and trusty measure vendors. Then Ochestrator forms a path link based on users' trustworthiness requirements while the whole network has fault convergence.

```
               +-----------+
               |Ochestrator|
               +-+--+---+--+
                 ^  ^   ^
                 |  |   |
      +----------+  |   +------------+
      |          evidence            |
      |             |                |
   +--+-+          +-+--+         +-+--+
   | RT +---------+ RT +----------+ RT |
   +----+   1      +----+     2   +----+

                Trusted link
```

Figure 6: a new router added to network

5. Security Considerations

TBD

6. IANA Considerations

This document does not require any action from IANA.

7. Acknowledgement

TBD

8. Informative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-08 (work in progress), December 2020.

Authors' Addresses

Meiling Chen
China Mobile
32, Xuanwumen West
BeiJing, BeiJing 100053
China

Email:
chenmeiling@chinamobile.com

Li Su
China Mobile

32, Xuanwumen West

BeiJing

100053

China

Email:
suli@chinamobile.com

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 25 October 2021

H. Birkholz
Fraunhofer SIT
D. Thaler
Microsoft
M. Richardson
Sandelman Software Works
N. Smith
Intel
W. Pan
Huawei Technologies

23 April 2021

Remote Attestation Procedures Architecture
draft-ietf-rats-architecture-12

Abstract

In network protocol exchanges it is often useful for one end of a communication to know whether the other end is in an intended operating state. This document provides an architectural overview of the entities involved that make such tests possible through the process of generating, conveying, and evaluating evidentiary claims. An attempt is made to provide for a model that is neutral toward processor architectures, the content of claims, and protocols.

Note to Readers

Discussion of this document takes place on the RATS Working Group mailing list (rats@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/rats/ (https://mailarchive.ietf.org/arch/browse/rats/).

Source for this draft and an issue tracker can be found at https://github.com/ietf-rats-wg/architecture (https://github.com/ietf-rats-wg/architecture).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The question of how one system can know that another system can be trusted has found new interest and relevance in a world where trusted computing elements are maturing in processor architectures.

Systems that have been attested and verified to be in a good state (for some value of "good") can improve overall system posture. Conversely, systems that cannot be attested and verified to be in a good state can be taken out of service, or otherwise flagged for repair.

For example:

* A bank back-end system might refuse to transact with another system that is not known to be in a good state.

* A healthcare system might refuse to transmit electronic healthcare records to a system that is not known to be in a good state.

In Remote Attestation Procedures (RATS), one peer (the "Attester") produces believable information about itself - Evidence - to enable a remote peer (the "Relying Party") to decide whether to consider that Attester a trustworthy peer or not. RATS are facilitated by an additional vital party, the Verifier.

The Verifier appraises Evidence via appraisal policies and creates the Attestation Results to support Relying Parties in their decision process. This document defines a flexible architecture consisting of attestation roles and their interactions via conceptual messages. Additionally, this document defines a universal set of terms that can be mapped to various existing and emerging Remote Attestation Procedures. Common topological patterns and the sequence of data flows associated with them, such as the "Passport Model" and the "Background-Check Model", are illustrated. The purpose is to define useful terminology for remote attestation and enable readers to map their solution architecture to the canonical attestation architecture provided here. Having a common terminology that provides well-understood meanings for common themes such as roles, device composition, topological patterns, and appraisal procedures is vital for semantic interoperability across solutions and platforms involving multiple vendors and providers.

Amongst other things, this document is about trust and trustworthiness. Trust is a choice one makes about another system. Trustworthiness is a quality about the other system that can be used in making one’s decision to trust it or not. This is subtle difference and being familiar with the difference is crucial for using this document. Additionally, the concepts of freshness and trust relationships with respect to RATS are elaborated on to enable implementers to choose appropriate solutions to compose their Remote Attestation Procedures.

2. Reference Use Cases

This section covers a number of representative and generic use cases for remote attestation, independent of specific solutions. The purpose is to provide motivation for various aspects of the architecture presented in this document. Many other use cases exist, and this document does not intend to have a complete list, only to illustrate a set of use cases that collectively cover all the functionality required in the architecture.

Each use case includes a description followed by an additional summary of the Attester and Relying Party roles derived from the use case.

2.1. Network Endpoint Assessment

Network operators want a trustworthy report that includes identity and version information about the hardware and software on the machines attached to their network, for purposes such as inventory, audit, anomaly detection, record maintenance and/or trending reports (logging). The network operator may also want a policy by which full access is only granted to devices that meet some definition of hygiene, and so wants to get Claims about such information and verify its validity. Remote attestation is desired to prevent vulnerable or compromised devices from getting access to the network and potentially harming others.

Typically, solutions start with a specific component (called a root of trust) that is intended to provide trustworthy device identity and protected storage for measurements. The system components perform a series of measurements that may be signed via functions provided by a root of trust, considered as Evidence about present system components, such as hardware, firmware, BIOS, software, etc.

Attester: A device desiring access to a network.

Relying Party: Network equipment such as a router, switch, or access point, responsible for admission of the device into the network.

2.2. Confidential Machine Learning Model Protection

A device manufacturer wants to protect its intellectual property. The intellectual property's scope primarily encompasses the machine learning (ML) model that is deployed in the devices purchased by its customers. The protection goals include preventing attackers, potentially the customer themselves, from seeing the details of the model.

This typically works by having some protected environment in the device go through a remote attestation with some manufacturer service that can assess its trustworthiness. If remote attestation succeeds, then the manufacturer service releases either the model, or a key to decrypt a model already deployed on the Attester in encrypted form, to the requester.

Attester: A device desiring to run an ML model.

Relying Party: A server or service holding ML models it desires to protect.

2.3. Confidential Data Protection

This is a generalization of the ML model use case above, where the data can be any highly confidential data, such as health data about customers, payroll data about employees, future business plans, etc. As part of the attestation procedure, an assessment is made against a set of policies to evaluate the state of the system that is requesting the confidential data. Attestation is desired to prevent leaking data via compromised devices.

Attester: An entity desiring to retrieve confidential data.

Relying Party: An entity that holds confidential data for release to authorized entities.

2.4. Critical Infrastructure Control

Potentially harmful physical equipment (e.g., power grid, traffic control, hazardous chemical processing, etc.) is connected to a network in support of critical infrastructure. The organization managing such infrastructure needs to ensure that only authorized code and users can control corresponding critical processes, and that these processes are protected from unauthorized manipulation or other threats. When a protocol operation can affect a critical system component of the infrastructure, devices attached to that critical component require some assurances depending on the security context, including that: a requesting device or application has not been compromised, and the requesters and actors act on applicable policies. As such, remote attestation can be used to only accept commands from requesters that are within policy.

Attester: A device or application wishing to control physical equipment.

Relying Party: A device or application connected to potentially dangerous physical equipment (hazardous chemical processing, traffic control, power grid, etc.).

2.5. Trusted Execution Environment Provisioning

A Trusted Application Manager (TAM) server is responsible for managing the applications running in a Trusted Execution Environment (TEE) of a client device, as described in [I-D.ietf-teep-architecture]. To achieve its purpose, the TAM needs to assess the state of a TEE, or of applications in the TEE, of a client device. The TEE conducts Remote Attestation Procedures with the TAM, which can then decide whether the TEE is already in compliance with the TAM's latest policy. If not, the TAM has to uninstall, update, or install approved applications in the TEE to bring it back into compliance with the TAM's policy.

Attester: A device with a TEE capable of running trusted applications that can be updated.

Relying Party: A TAM.

2.6. Hardware Watchdog

There is a class of malware that holds a device hostage and does not allow it to reboot to prevent updates from being applied. This can be a significant problem, because it allows a fleet of devices to be held hostage for ransom.

A solution to this problem is a watchdog timer implemented in a protected environment such as a Trusted Platform Module (TPM), as described in [TCGarch] section 43.3. If the watchdog does not receive regular, and fresh, Attestation Results as to the system's health, then it forces a reboot.

Attester: The device that should be protected from being held hostage for a long period of time.

Relying Party: A watchdog capable of triggering a procedure that resets a device into a known, good operational state.

2.7. FIDO Biometric Authentication

In the Fast IDentity Online (FIDO) protocol [WebAuthN], [CTAP], the device in the user's hand authenticates the human user, whether by biometrics (such as fingerprints), or by PIN and password. FIDO authentication puts a large amount of trust in the device compared to typical password authentication because it is the device that verifies the biometric, PIN and password inputs from the user, not the server. For the Relying Party to know that the authentication is trustworthy, the Relying Party needs to know that the Authenticator part of the device is trustworthy. The FIDO protocol employs remote attestation for this.

The FIDO protocol supports several remote attestation protocols and a mechanism by which new ones can be registered and added. Remote attestation defined by RATS is thus a candidate for use in the FIDO protocol.

Other biometric authentication protocols such as the Chinese IFAA standard and WeChat Pay as well as Google Pay make use of remote attestation in one form or another.

Attester: Every FIDO Authenticator contains an Attester.

Relying Party: Any web site, mobile application back-end, or service that relies on authentication data based on biometric information.

3. Architectural Overview

Figure 1 depicts the data that flows between different roles, independent of protocol or use case.

```
    ************   *************    ************    *****************
    * Endorser *   * Reference *    * Verifier *    * Relying Party *
    ************   * Value     *    *  Owner   *    *  Owner        *
       |           * Provider  *    ************    *****************
       |           *************          |                 |
       |                  |               |                 |
       |Endorsements      |Reference      |Appraisal        |Appraisal
       |                  |Values         |Policy           |Policy for
       |                  |               |for              |Attestation
       .-----------.      |               |Evidence         |Results
                   |      |               |                 |
                   |      |               |                 |
                   v      v               v                 |
                 .---------------------------.              |
          .----->|          Verifier         |------.       |
          |      '---------------------------'      |       |
          |                                         |       |
          |                              Attestation|       |
          |                              Results    |       |
          | Evidence                                |       |
          |                                         |       |
          |                                         v       v
    .----------.                                .---------------.
    | Attester |                                | Relying Party |
    '----------'                                '---------------'
```

Figure 1: Conceptual Data Flow

The text below summarizes the activities conducted by the roles illustrated in Figure 1.

An Attester creates Evidence that is conveyed to a Verifier.

A Verifier uses the Evidence, any Reference Values from Reference Value Providers, and any Endorsements from Endorsers, by applying an Appraisal Policy for Evidence to assess the trustworthiness of the Attester. This procedure is called the appraisal of Evidence.

Subsequently, the Verifier generates Attestation Results for use by Relying Parties.

The Appraisal Policy for Evidence might be obtained from the Verifier Owner via some protocol mechanism, or might be configured into the Verifier by the Verifier Owner, or might be programmed into the Verifier, or might be obtained via some other mechanism.

A Relying Party uses Attestation Results by applying its own appraisal policy to make application-specific decisions, such as authorization decisions. This procedure is called the appraisal of Attestation Results.

The Appraisal Policy for Attestation Results might be obtained from the Relying Party Owner via some protocol mechanism, or might be configured into the Relying Party by the Relying Party Owner, or might be programmed into the Relying Party, or might be obtained via some other mechanism.

See Section 8 for further discussion of the conceptual messages shown in Figure 1. ## Two Types of Environments of an Attester

As shown in Figure 2, an Attester consists of at least one Attesting Environment and at least one Target Environment. In some implementations, the Attesting and Target Environments might be combined. Other implementations might have multiple Attesting and Target Environments, such as in the examples described in more detail in Section 3.1 and Section 3.2. Other examples may exist. All compositions of Attesting and Target Environments discussed in this architecture can be combined into more complex implementations.

```
     .--------------------------------.
     |                                |
     |            Verifier            |
     |                                |
     '--------------------------------'
                             ^
                             |
   .-------------------------|----------.
   |                         |          |
   |   .----------------.    |          |
   |   | Target         |    |          |
   |   | Environment    |    |          |
   |   |                |    | Evidence |
   |   '----------------'    |          |
   |                   |     |          |
   |                   |     |          |
   |          Collect  |     |          |
   |           Claims  |     |          |
   |                   |     |          |
   |                   v     |          |
   |                 .-------------.    |
   |                 | Attesting   |    |
   |                 | Environment |    |
   |                 |             |    |
   |                 '-------------'    |
   |               Attester             |
   '------------------------------------'
```

Figure 2: Two Types of Environments

Claims are collected from Target Environments. That is, Attesting Environments collect the values and the information to be represented in Claims, by reading system registers and variables, calling into subsystems, taking measurements on code, memory, or other security related assets of the Target Environment. Attesting Environments then format the Claims appropriately, and typically use key material and cryptographic functions, such as signing or cipher algorithms, to generate Evidence. There is no limit to or requirement on the types of hardware or software environments that can be used to implement an Attesting Environment, for example: Trusted Execution Environments (TEEs), embedded Secure Elements (eSEs), Trusted Platform Modules (TPMs) [TCGarch], or BIOS firmware.

An arbitrary execution environment may not, by default, be capable of Claims collection for a given Target Environment. Execution environments that are designed specifically to be capable of Claims collection are referred to in this document as Attesting Environments. For example, a TPM doesn't actively collect Claims itself, it instead requires another component to feed various values to the TPM. Thus, an Attesting Environment in such a case would be the combination of the TPM together with whatever component is feeding it the measurements.

3.1. Layered Attestation Environments

By definition, the Attester role generates Evidence. An Attester may consist of one or more nested environments (layers). The root layer of an Attester includes at least one root of trust. In order to appraise Evidence generated by an Attester, the Verifier needs to trust the Attester's root of trust. Trust in the Attester's root of trust can be established in various ways as discussed in Section 7.4.

In layered attestation, a root of trust is the initial Attesting Environment. Claims can be collected from or about each layer. The corresponding Claims can be structured in a nested fashion that reflects the nesting of the Attester's layers. Normally, Claims are not self-asserted, rather a previous layer acts as the Attesting Environment for the next layer. Claims about a root of trust typically are asserted by an Endorser.

The example device illustrated in Figure 3 includes (A) a BIOS stored in read-only memory, (B) a bootloader, and (C) an operating system kernel.

```
              .-------------.   Endorsement for ROM
              |  Endorser   |-----------------------.
              '-------------'                       |
                                                    v
              .-------------.   Reference      .----------.
              | Reference   |   Values for     |          |
              | Value       |----------------->| Verifier |
              | Provider(s) | ROM, bootloader, |          |
              '-------------'    and kernel    '----------'
                                                    ^
          .------------------------------------.    |
          |                                    |    |
          |   .---------------------------.    |    |
          |   | Kernel                    |    |    |
          |   |                           |    |    | Layered
          |   |   Target                  |    |    | Evidence
          |   | Environment               |    |    |   for
          |   '---------------------------'    |    | bootloader
          |           Collect |                |    |   and
          |           Claims  |                |    | kernel
          |   .---------------|-----------.    |    |
          |   | Bootloader    v           |    |    |
          |   |             .-----------. |    |    |
          |   |   Target    | Attesting | |    |    |
          |   | Environment |Environment|-----------'
          |   |             |           | |    |
          |   |             '-----------' |    |
          |   |                 ^         |    |
          |   '-----------------|---------'    |
          |          Collect |  | Evidence for |
          |          Claims  v  | bootloader   |
          |   .---------------------------.    |
          |   | ROM                       |    |
          |   |                           |    |
          |   |               Attesting   |    |
          |   |              Environment  |    |
          |   '---------------------------'    |
          |                                    |
          '------------------------------------'
```

Figure 3: Layered Attester

The first Attesting Environment, the read-only BIOS in this example, has to ensure the integrity of the bootloader (the first Target Environment). There are potentially multiple kernels to boot, and the decision is up to the bootloader. Only a bootloader with intact integrity will make an appropriate decision. Therefore, the Claims relating to the integrity of the bootloader have to be measured securely. At this stage of the boot-cycle of the device, the Claims collected typically cannot be composed into Evidence.

After the boot sequence is started, the BIOS conducts the most important and defining feature of layered attestation, which is that the successfully measured bootloader now becomes (or contains) an Attesting Environment for the next layer. This procedure in layered attestation is sometimes called "staging". It is important that the bootloader not be able to alter any Claims about itself that were collected by the BIOS. This can be ensured having those Claims be either signed by the BIOS or stored in a tamper-proof manner by the BIOS.

Continuing with this example, the bootloader's Attesting Environment is now in charge of collecting Claims about the next Target Environment, which in this example is the kernel to be booted. The final Evidence thus contains two sets of Claims: one set about the bootloader as measured and signed by the BIOS, plus a set of Claims about the kernel as measured and signed by the bootloader.

This example could be extended further by making the kernel become another Attesting Environment for an application as another Target Environment. This would result in a third set of Claims in the Evidence pertaining to that application.

The essence of this example is a cascade of staged environments. Each environment has the responsibility of measuring the next environment before the next environment is started. In general, the number of layers may vary by device or implementation, and an Attesting Environment might even have multiple Target Environments that it measures, rather than only one as shown by example in Figure 3.

3.2. Composite Device

A composite device is an entity composed of multiple sub-entities such that its trustworthiness has to be determined by the appraisal of all these sub-entities.

Each sub-entity has at least one Attesting Environment collecting the Claims from at least one Target Environment, then this sub-entity generates Evidence about its trustworthiness. Therefore, each sub-entity can be called an Attester. Among all the Attesters, there may be only some which have the ability to communicate with the Verifier while others do not.

For example, a carrier-grade router consists of a chassis and multiple slots. The trustworthiness of the router depends on all its slots' trustworthiness. Each slot has an Attesting Environment, such as a TEE, collecting the Claims of its boot process, after which it generates Evidence from the Claims.

Among these slots, only a "main" slot can communicate with the Verifier while other slots cannot. But other slots can communicate with the main slot by the links between them inside the router. So the main slot collects the Evidence of other slots, produces the final Evidence of the whole router and conveys the final Evidence to the Verifier. Therefore the router is a composite device, each slot is an Attester, and the main slot is the lead Attester.

Another example is a multi-chassis router composed of multiple single carrier-grade routers. Multi-chassis router setups create redundancy groups that provide higher throughput by interconnecting multiple routers in these groups, which can be treated as one logical router for simpler management. A multi-chassis router setup provides a management point that connects to the Verifier. Typically one router in the group is designated as the main router. Other routers in the multi-chassis setup are connected to the main router only via physical network links and are therefore managed and appraised via the main router's help. In consequence, a multi-chassis router setup is a composite device, each router is an Attester, and the main router is the lead Attester.

Figure 4 depicts the conceptual data flow for a composite device.

```
                      .-----------------------------.
                      |           Verifier          |
                      '-----------------------------'
                                      ^
                                      |
                                      | Evidence of
                                      | Composite Device
                                      |
   .----------------------------------|-------------------------------.
   | .--------------------------------|-----.      .------------.     |
   | |  Collect             .------------.  |      |            |     |
   | |  Claims   .--------->|  Attesting |<--------| Attester B |-.   |
   | |           |          |Environment |  |      '------------. |   |
   | |  .----------------.  |            |<----------| Attester C |-. |
   | |  |     Target     |  |            |  |        '------------' | |
   | |  | Environment(s) |  |            |<------------| ...        | |
   | |  |                |  '------------'  | Evidence '------------' |
   | |  '----------------'                  |    of                   |
   | |                                      | Attesters               |
   | | lead Attester A                      | (via Internal Links or  |
   | '--------------------------------------' Network Connections)    |
   |                                                                  |
   |                       Composite Device                           |
   '------------------------------------------------------------------'
```

Figure 4: Composite Device

In a composite device, each Attester generates its own Evidence by its Attesting Environment(s) collecting the Claims from its Target Environment(s). The lead Attester collects Evidence from other Attesters and conveys it to a Verifier. Collection of Evidence from sub-entities may itself be a form of Claims collection that results in Evidence asserted by the lead Attester. The lead Attester generates Evidence about the layout of the whole composite device, while sub-Attesters generate Evidence about their respective (sub-)modules.

In this scenario, the trust model described in Section 7 can also be applied to an inside Verifier.

3.3. Implementation Considerations

An entity can take on multiple RATS roles (e.g., Attester, Verifier, Relying Party, etc.) at the same time. Multiple entities can cooperate to implement a single RATS role as well. In essence, the combination of roles and entities can be arbitrary. For example, in the composite device scenario, the entity inside the lead Attester can also take on the role of a Verifier, and the outer entity of

Verifier can take on the role of a Relying Party. After collecting the Evidence of other Attesters, this inside Verifier uses Endorsements and appraisal policies (obtained the same way as by any other Verifier) as part of the appraisal procedures that generate Attestation Results. The inside Verifier then conveys the Attestation Results of other Attesters to the outside Verifier, whether in the same conveyance protocol as part of the Evidence or not.

4. Terminology

This document uses the following terms.

4.1. Roles

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

Produces: Evidence

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

Consumes: Attestation Results

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

Consumes: Evidence, Reference Values, Endorsements, Appraisal Policy for Evidence

Produces: Attestation Results

Relying Party Owner: A role performed by an entity (typically an administrator), that is authorized to configure Appraisal Policy for Attestation Results in a Relying Party.

Produces: Appraisal Policy for Attestation Results

Verifier Owner: A role performed by an entity (typically an administrator), that is authorized to configure Appraisal Policy for Evidence in a Verifier.

Produces: Appraisal Policy for Evidence

Endorser: A role performed by an entity (typically a manufacturer) whose Endorsements help Verifiers appraise the authenticity of Evidence.

Produces: Endorsements

Reference Value Provider: A role performed by an entity (typically a manufacturer) whose Reference Values help Verifiers appraise Evidence to determine if acceptable known Claims have been recorded by the Attester.

Produces: Reference Values

4.2. Artifacts

Claim: A piece of asserted information, often in the form of a name/value pair. Claims make up the usual structure of Evidence and other RATS artifacts. Compare /claim/ in [RFC7519].

Endorsement: A secure statement that an Endorser vouches for the integrity of an Attester's various capabilities such as Claims collection and Evidence signing.

Consumed By: Verifier

Produced By: Endorser

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

Consumed By: Verifier

Produced By: Attester

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

Consumed By: Relying Party

Produced By: Verifier

Appraisal Policy for Evidence: A set of rules that informs how a Verifier evaluates the validity of information about an Attester. Compare /security policy/ in [RFC4949].

Consumed By: Verifier

Produced By: Verifier Owner

Appraisal Policy for Attestation Results: A set of rules that direct how a Relying Party uses the Attestation Results regarding an Attester generated by the Verifiers. Compare /security policy/ in [RFC4949].

Consumed by: Relying Party

Produced by: Relying Party Owner

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

Consumed By: Verifier

Produced By: Reference Value Provider

5. Topological Patterns

Figure 1 shows a data-flow diagram for communication between an Attester, a Verifier, and a Relying Party. The Attester conveys its Evidence to the Verifier for appraisal, and the Relying Party receives the Attestation Result from the Verifier. This section refines the data-flow diagram by describing two reference models, as well as one example composition thereof. The discussion that follows is for illustrative purposes only and does not constrain the interactions between RATS roles to the presented patterns.

5.1. Passport Model

The passport model is so named because of its resemblance to how nations issue passports to their citizens. The nature of the Evidence that an individual needs to provide to its local authority is specific to the country involved. The citizen retains control of the resulting passport document and presents it to other entities when it needs to assert a citizenship or identity Claim, such as an airport immigration desk. The passport is considered sufficient because it vouches for the citizenship and identity Claims, and it is issued by a trusted authority. Thus, in this immigration desk analogy, the passport issuing agency is a Verifier, the passport is an Attestation Result, and the immigration desk is a Relying Party.

In this model, an Attester conveys Evidence to a Verifier, which compares the Evidence against its appraisal policy. The Verifier then gives back an Attestation Result. If the Attestation Result was a successful one, the Attester can then present the Attestation Result (and possibly additional Claims) to a Relying Party, which then compares this information against its own appraisal policy.

Three ways in which the process may fail include:

* First, the Verifier may not issue a positive Attestation Result due to the Evidence not passing the Appraisal Policy for Evidence.

* The second way in which the process may fail is when the Attestation Result is examined by the Relying Party, and based upon the Appraisal Policy for Attestation Results, the result does not pass the policy.

* The third way is when the Verifier is unreachable or unavailable.

Since the resource access protocol between the Attester and Relying Party includes an Attestation Result, in this model the details of that protocol constrain the serialization format of the Attestation Result. The format of the Evidence on the other hand is only constrained by the Attester-Verifier remote attestation protocol. This implies that interoperability and standardization is more relevant for Attestation Results than it is for Evidence.

```
        +------------+
        |            | Compare Evidence
        |  Verifier  | against appraisal policy
        |            |
        +------------+
             ^    |
    Evidence |    | Attestation
             |    | Result
             |    v
        +------------+              +-------------+
        |            |------------->|             | Compare Attestation
        |  Attester  | Attestation  |   Relying   | Result against
        |            | Result       |    Party    | appraisal policy
        +------------+              +-------------+
```

Figure 5: Passport Model

5.2. Background-Check Model

The background-check model is so named because of the resemblance of how employers and volunteer organizations perform background checks. When a prospective employee provides Claims about education or previous experience, the employer will contact the respective institutions or former employers to validate the Claim. Volunteer organizations often perform police background checks on volunteers in order to determine the volunteer’s trustworthiness. Thus, in this analogy, a prospective volunteer is an Attester, the organization is the Relying Party, and the organization that issues a report is a Verifier.

In this model, an Attester conveys Evidence to a Relying Party, which simply passes it on to a Verifier. The Verifier then compares the Evidence against its appraisal policy, and returns an Attestation Result to the Relying Party. The Relying Party then compares the Attestation Result against its own appraisal policy.

The resource access protocol between the Attester and Relying Party includes Evidence rather than an Attestation Result, but that Evidence is not processed by the Relying Party. Since the Evidence is merely forwarded on to a trusted Verifier, any serialization format can be used for Evidence because the Relying Party does not need a parser for it. The only requirement is that the Evidence can be _encapsulated in_ the format required by the resource access protocol between the Attester and Relying Party.

However, like in the Passport model, an Attestation Result is still consumed by the Relying Party. Code footprint and attack surface area can be minimized by using a serialization format for which the

Relying Party already needs a parser to support the protocol between the Attester and Relying Party, which may be an existing standard or widely deployed resource access protocol. Such minimization is especially important if the Relying Party is a constrained node.

```
                                   +-------------+
                                   |             | Compare Evidence
                                   |   Verifier  | against appraisal
                                   |             | policy
                                   +-------------+
                                        ^    |
                               Evidence |    | Attestation
                                        |    | Result
                                        |    v
      +------------+               +-------------+
      |            |-------------->|             | Compare Attestation
      |  Attester  |   Evidence    |   Relying   | Result against
      |            |               |    Party    | appraisal policy
      +------------+               +-------------+
```

Figure 6: Background-Check Model

5.3. Combinations

One variation of the background-check model is where the Relying Party and the Verifier are on the same machine, performing both functions together. In this case, there is no need for a protocol between the two.

It is also worth pointing out that the choice of model depends on the use case, and that different Relying Parties may use different topological patterns.

The same device may need to create Evidence for different Relying Parties and/or different use cases. For instance, it would use one model to provide Evidence to a network infrastructure device to gain access to the network, and the other model to provide Evidence to a server holding confidential data to gain access to that data. As such, both models may simultaneously be in use by the same device.

Figure 7 shows another example of a combination where Relying Party 1 uses the passport model, whereas Relying Party 2 uses an extension of the background-check model. Specifically, in addition to the basic functionality shown in Figure 6, Relying Party 2 actually provides the Attestation Result back to the Attester, allowing the Attester to use it with other Relying Parties. This is the model that the Trusted Application Manager plans to support in the TEEP architecture [I-D.ietf-teep-architecture].

```
       +-------------+
       |             | Compare Evidence
       |   Verifier  | against appraisal policy
       |             |
       +-------------+
            ^    |
    Evidence |    | Attestation
            |    | Result
            |    v
       +-------------+
       |             | Compare
       |   Relying   | Attestation Result
       |   Party 2   | against appraisal policy
       +-------------+
            ^    |
    Evidence |    | Attestation
            |    | Result
            |    v
       +-------------+               +-------------+
       |             |-------------->|             | Compare Attestation
       |   Attester  |  Attestation  |   Relying   | Result against
       |             |     Result    |   Party 1   | appraisal policy
       +-------------+               +-------------+
```

Figure 7: Example Combination

6. Roles and Entities

An entity in the RATS architecture includes at least one of the roles defined in this document.

An entity can aggregate more than one role into itself, such as being both a Verifier and a Relying Party, or being both a Reference Value Provider and an Endorser. As such, any conceptual messages (see Section 8 for more discussion) originating from such roles might also be combined. For example, Reference Values might be conveyed as part of an appraisal policy if the Verifier Owner and Reference Value Provider roles are combined. Similarly, Reference Values might be conveyed as part of an Endorsement if the Endorser and Reference Value Provider roles are combined.

Interactions between roles aggregated into the same entity do not necessarily use the Internet Protocol. Such interactions might use a loopback device or other IP-based communication between separate environments, but they do not have to. Alternative channels to convey conceptual messages include function calls, sockets, GPIO interfaces, local busses, or hypervisor calls. This type of conveyance is typically found in composite devices. Most importantly, these conveyance methods are out-of-scope of RATS, but they are presumed to exist in order to convey conceptual messages appropriately between roles.

In essence, an entity that combines more than one role creates and consumes the corresponding conceptual messages as defined in this document.

7. Trust Model

7.1. Relying Party

This document covers scenarios for which a Relying Party trusts a Verifier that can appraise the trustworthiness of information about an Attester. Such trust might come by the Relying Party trusting the Verifier (or its public key) directly, or might come by trusting an entity (e.g., a Certificate Authority) that is in the Verifier's certificate path. Such trust is expressed by storing one or more "trust anchors" in a secure location known as a trust anchor store.

As defined in [RFC6024], "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative." The trust anchor may be a certificate or it may be a raw public key along with additional data if necessary such as its public key algorithm and parameters.

The Relying Party might implicitly trust a Verifier, such as in a Verifier/Relying Party combination where the Verifier and Relying Party roles are combined. Or, for a stronger level of security, the Relying Party might require that the Verifier first provide information about itself that the Relying Party can use to assess the trustworthiness of the Verifier before accepting its Attestation Results.

For example, one explicit way for a Relying Party "A" to establish such trust in a Verifier "B", would be for B to first act as an Attester where A acts as a combined Verifier/Relying Party. If A then accepts B as trustworthy, it can choose to accept B as a Verifier for other Attesters.

As another example, the Relying Party can establish trust in the Verifier by out of band establishment of key material, combined with a protocol like TLS to communicate. There is an assumption that between the establishment of the trusted key material and the creation of the Evidence, that the Verifier has not been compromised.

Similarly, the Relying Party also needs to trust the Relying Party Owner for providing its Appraisal Policy for Attestation Results, and in some scenarios the Relying Party might even require that the Relying Party Owner go through a remote attestation procedure with it before the Relying Party will accept an updated policy. This can be done similarly to how a Relying Party could establish trust in a Verifier as discussed above.

7.2. Attester

In some scenarios, Evidence might contain sensitive information such as Personally Identifiable Information (PII) or system identifiable information. Thus, an Attester must trust entities to which it conveys Evidence, to not reveal sensitive data to unauthorized parties. The Verifier might share this information with other authorized parties, according to a governing policy that address the handling of sensitive information (potentially included in Appraisal Policies for Evidence). In the background-check model, this Evidence may also be revealed to Relying Party(s).

When Evidence contains sensitive information, an Attester typically requires that a Verifier authenticates itself (e.g., at TLS session establishment) and might even request a remote attestation before the Attester sends the sensitive Evidence. This can be done by having the Attester first act as a Verifier/Relying Party, and the Verifier act as its own Attester, as discussed above.

7.3. Relying Party Owner

The Relying Party Owner might also require that the Relying Party first act as an Attester, providing Evidence that the Owner can appraise, before the Owner would give the Relying Party an updated policy that might contain sensitive information. In such a case, authentication or attestation in both directions might be needed, in which case typically one side's Evidence must be considered safe to share with an untrusted entity, in order to bootstrap the sequence. See Section 11 for more discussion.

7.4. Verifier

The Verifier trusts (or more specifically, the Verifier's security policy is written in a way that configures the Verifier to trust) a manufacturer, or the manufacturer's hardware, so as to be able to appraise the trustworthiness of that manufacturer's devices. Such trust is expressed by storing one or more trust anchors in the Verifier's trust anchor store.

In a typical solution, a Verifier comes to trust an Attester indirectly by having an Endorser (such as a manufacturer) vouch for the Attester's ability to securely generate Evidence, in which case the Endorser's key material is stored in the Verifier's trust anchor store.

In some solutions, a Verifier might be configured to directly trust an Attester by having the Verifier have the Attester's key material (rather than the Endorser's) in its trust anchor store.

Such direct trust must first be established at the time of trust anchor store configuration either by checking with an Endorser at that time, or by conducting a security analysis of the specific device. Having the Attester directly in the trust anchor store narrows the Verifier's trust to only specific devices rather than all devices the Endorser might vouch for, such as all devices manufactured by the same manufacturer in the case that the Endorser is a manufacturer.

Such narrowing is often important since physical possession of a device can also be used to conduct a number of attacks, and so a device in a physically secure environment (such as one's own premises) may be considered trusted whereas devices owned by others would not be. This often results in a desire to either have the owner run their own Endorser that would only endorse devices one owns, or to use Attesters directly in the trust anchor store. When there are many Attesters owned, the use of an Endorser enables better scalability.

That is, a Verifier might appraise the trustworthiness of an application component, operating system component, or service under the assumption that information provided about it by the lower-layer firmware or software is true. A stronger level of assurance of security comes when information can be vouched for by hardware or by ROM code, especially if such hardware is physically resistant to hardware tampering. In most cases, components that have to be vouched for via Endorsements because no Evidence is generated about them are referred to as roots of trust.

The manufacturer having arranged for an Attesting Environment to be provisioned with key material with which to sign Evidence, the Verifier is then provided with some way of verifying the signature on the Evidence. This may be in the form of an appropriate trust anchor, or the Verifier may be provided with a database of public keys (rather than certificates) or even carefully curated and secured lists of symmetric keys.

The nature of how the Verifier manages to validate the signatures produced by the Attester is critical to the secure operation of a remote attestation system, but is not the subject of standardization within this architecture.

A conveyance protocol that provides authentication and integrity protection can be used to convey Evidence that is otherwise unprotected (e.g., not signed). Appropriate conveyance of unprotected Evidence (e.g., [I-D.birkholz-rats-uccs]) relies on the following conveyance protocol's protection capabilities:

1. The key material used to authenticate and integrity protect the conveyance channel is trusted by the Verifier to speak for the Attesting Environment(s) that collected Claims about the Target Environment(s).

2. All unprotected Evidence that is conveyed is supplied exclusively by the Attesting Environment that has the key material that protects the conveyance channel

3. The root of trust protects both the conveyance channel key material and the Attesting Environment with equivalent strength protections.

As illustrated in [I-D.birkholz-rats-uccs], an entity that receives unprotected Evidence via a trusted conveyance channel always takes on the responsibility of vouching for the Evidence's authenticity and freshness. If protected Evidence is generated, the Attester's Attesting Environments take on that responsibility. In cases where unprotected Evidence is processed by a Verifier, Relying Parties have to trust that the Verifier is capable of handling Evidence in a manner that preserves the Evidence's authenticity and freshness. Generating and conveying unprotected Evidence always creates significant risk and the benefits of that approach have to be carefully weighed against potential drawbacks.

See Section 12 for discussion on security strength.

7.5. Endorser, Reference Value Provider, and Verifier Owner

In some scenarios, the Endorser, Reference Value Provider, and Verifier Owner may need to trust the Verifier before giving the Endorsement, Reference Values, or appraisal policy to it. This can be done similarly to how a Relying Party might establish trust in a Verifier.

As discussed in Section 7.3, authentication or attestation in both directions might be needed, in which case typically one side's identity or Evidence must be considered safe to share with an untrusted entity, in order to bootstrap the sequence. See Section 11 for more discussion.

8. Conceptual Messages

Figure 1 illustrates the flow of a conceptual messages between various roles. This section provides additional elaboration and implementation considerations. It is the responsibility of protocol specifications to define the actual data format and semantics of any relevant conceptual messages.

8.1. Evidence

Evidence is a set of Claims about the target environment that reveal operational status, health, configuration or construction that have security relevance. Evidence is appraised by a Verifier to establish its relevance, compliance, and timeliness. Claims need to be collected in a manner that is reliable. Evidence needs to be securely associated with the target environment so that the Verifier cannot be tricked into accepting Claims originating from a different environment (that may be more trustworthy). Evidence also must be protected from man-in-the-middle attackers who may observe, change or misdirect Evidence as it travels from Attester to Verifier. The timeliness of Evidence can be captured using Claims that pinpoint the time or interval when changes in operational status, health, and so forth occur.

8.2. Endorsements

An Endorsement is a secure statement that some entity (e.g., a manufacturer) vouches for the integrity of the device's signing capability. For example, if the signing capability is in hardware, then an Endorsement might be a manufacturer certificate that signs a public key whose corresponding private key is only known inside the device's hardware. Thus, when Evidence and such an Endorsement are used together, an appraisal procedure can be conducted based on appraisal policies that may not be specific to the device instance, but merely specific to the manufacturer providing the Endorsement. For example, an appraisal policy might simply check that devices from a given manufacturer have information matching a set of Reference Values, or an appraisal policy might have a set of more complex logic on how to appraise the validity of information.

However, while an appraisal policy that treats all devices from a given manufacturer the same may be appropriate for some use cases, it would be inappropriate to use such an appraisal policy as the sole means of authorization for use cases that wish to constrain _which_ compliant devices are considered authorized for some purpose. For example, an enterprise using remote attestation for Network Endpoint Assessment [RFC5209] may not wish to let every healthy laptop from the same manufacturer onto the network, but instead only want to let devices that it legally owns onto the network. Thus, an Endorsement may be helpful information in authenticating information about a device, but is not necessarily sufficient to authorize access to resources which may need device-specific information such as a public key for the device or component or user on the device.

8.3. Reference Values

Reference Values used in appraisal procedures come from a Reference Value Provider and are then used by the Verifier to compare to Evidence. Reference Values with matching Evidence produces acceptable Claims. Additionally, appraisal policy may play a role in determining the acceptance of Claims.

8.4. Attestation Results

Attestation Results are the input used by the Relying Party to decide the extent to which it will trust a particular Attester, and allow it to access some data or perform some operation.

Attestation Results may carry a boolean value indicating compliance or non-compliance with a Verifier’s appraisal policy, or may carry a richer set of Claims about the Attester, against which the Relying Party applies its Appraisal Policy for Attestation Results.

The quality of the Attestation Results depends upon the ability of the Verifier to evaluate the Attester. Different Attesters have a different _Strength of Function_ [strengthoffunction], which results in the Attestation Results being qualitatively different in strength.

An Attestation Result that indicates non-compliance can be used by an Attester (in the passport model) or a Relying Party (in the background-check model) to indicate that the Attester should not be treated as authorized and may be in need of remediation. In some cases, it may even indicate that the Evidence itself cannot be authenticated as being correct.

By default, the Relying Party does not believe the Attester to be compliant. Upon receipt of an authentic Attestation Result and given the Appraisal Policy for Attestation Results is satisfied, the

Attester is allowed to perform the prescribed actions or access. The simplest such appraisal policy might authorize granting the Attester full access or control over the resources guarded by the Relying Party. A more complex appraisal policy might involve using the information provided in the Attestation Result to compare against expected values, or to apply complex analysis of other information contained in the Attestation Result.

Thus, Attestation Results often need to include detailed information about the Attester, for use by Relying Parties, much like physical passports and drivers licenses include personal information such as name and date of birth. Unlike Evidence, which is often very device- and vendor-specific, Attestation Results can be vendor-neutral, if the Verifier has a way to generate vendor-agnostic information based on the appraisal of vendor-specific information in Evidence. This allows a Relying Party's appraisal policy to be simpler, potentially based on standard ways of expressing the information, while still allowing interoperability with heterogeneous devices.

Finally, whereas Evidence is signed by the device (or indirectly by a manufacturer, if Endorsements are used), Attestation Results are signed by a Verifier, allowing a Relying Party to only need a trust relationship with one entity, rather than a larger set of entities, for purposes of its appraisal policy.

8.5. Appraisal Policies

The Verifier, when appraising Evidence, or the Relying Party, when appraising Attestation Results, checks the values of matched Claims against constraints specified in its appraisal policy. Examples of such constraints checking include:

* comparison for equality against a Reference Value, or

* a check for being in a range bounded by Reference Values, or

* membership in a set of Reference Values, or

* a check against values in other Claims.

Upon completing all appraisal policy constraints, the remaining Claims are accepted as input toward determining Attestation Results, when appraising Evidence, or as input to a Relying Party, when appraising Attestation Results.

9. Claims Encoding Formats

The following diagram illustrates a relationship to which remote attestation is desired to be added:

```
+-------------+               +------------+ Evaluate
|             |-------------->|            | request
|  Attester   |  Access some  |  Relying   | against
|             |    resource   |   Party    | security
+-------------+               +------------+ policy
```

Figure 8: Typical Resource Access

In this diagram, the protocol between Attester and a Relying Party can be any new or existing protocol (e.g., HTTP(S), COAP(S), ROLIE [RFC8322], 802.1x, OPC UA [OPCUA], etc.), depending on the use case.

Typically, such protocols already have mechanisms for passing security information for authentication and authorization purposes. Common formats include JWTs [RFC7519], CWTs [RFC8392], and X.509 certificates.

Retrofitting already deployed protocols with remote attestation requires adding RATS conceptual messages to the existing data flows. This must be done in a way that does not degrade the security properties of the systems involved and should use native extension mechanisms provided by the underlying protocol. For example, if a TLS handshake is to be extended with remote attestation capabilities, attestation Evidence may be embedded in an ad-hoc X.509 certificate extension (e.g., [TCG-DICE]), or into a new TLS Certificate Type (e.g., [I-D.tschofenig-tls-cwt]).

Especially for constrained nodes there is a desire to minimize the amount of parsing code needed in a Relying Party, in order to both minimize footprint and to minimize the attack surface. While it would be possible to embed a CWT inside a JWT, or a JWT inside an X.509 extension, etc., there is a desire to encode the information natively in a format that is already supported by the Relying Party.

This motivates having a common "information model" that describes the set of remote attestation related information in an encoding-agnostic way, and allowing multiple encoding formats (CWT, JWT, X.509, etc.) that encode the same information into the Claims format needed by the Relying Party.

The following diagram illustrates that Evidence and Attestation Results might be expressed via multiple potential encoding formats, so that they can be conveyed by various existing protocols. It also motivates why the Verifier might also be responsible for accepting Evidence that encodes Claims in one format, while issuing Attestation Results that encode Claims in a different format.

```
                Evidence           Attestation Results
.--------------.   CWT                    CWT   .-------------------.
|  Attester-A  |------------.      .----------->|  Relying Party V  |
'______________'            v      |            `___________________'
.--------------.   JWT   .------------.   JWT   .-------------------.
|  Attester-B  |-------->|  Verifier  |-------->|  Relying Party W  |
'______________'         |            |         `___________________'
.--------------.  X.509  |            |  X.509  .-------------------.
|  Attester-C  |-------->|            |-------->|  Relying Party X  |
'______________'         |            |         `___________________'
.--------------.   TPM   |            |   TPM   .-------------------.
|  Attester-D  |-------->|            |-------->|  Relying Party Y  |
'______________'         '____________'         `___________________'
.--------------.  other     ^      |     other  .-------------------.
|  Attester-E  |------------'      '----------->|  Relying Party Z  |
'______________'                                `___________________'
```

Figure 9: Multiple Attesters and Relying Parties with Different Formats

10. Freshness

A Verifier or Relying Party might need to learn the point in time (i.e., the "epoch") an Evidence or Attestation Result has been produced. This is essential in deciding whether the included Claims and their values can be considered fresh, meaning they still reflect the latest state of the Attester, and that any Attestation Result was generated using the latest Appraisal Policy for Evidence.

Freshness is assessed based on the Appraisal Policy for Evidence or Attestation Results that compares the estimated epoch against an "expiry" threshold defined locally to that policy. There is, however, always a race condition possible in that the state of the Attester, and the appraisal policies might change immediately after the Evidence or Attestation Result was generated. The goal is merely to narrow their recentness to something the Verifier (for Evidence) or Relying Party (for Attestation Result) is willing to accept. Some flexibility on the freshness requirement is a key component for enabling caching and reuse of both Evidence and Attestation Results, which is especially valuable in cases where their computation uses a substantial part of the resource budget (e.g., energy in constrained devices).

There are three common approaches for determining the epoch of Evidence or an Attestation Result.

10.1. Explicit Timekeeping using Synchronized Clocks

The first approach is to rely on synchronized and trustworthy clocks, and include a signed timestamp (see [I-D.birkholz-rats-tuda]) along with the Claims in the Evidence or Attestation Result. Timestamps can also be added on a per-Claim basis to distinguish the time of generation of Evidence or Attestation Result from the time that a specific Claim was generated. The clock's trustworthiness can generally be established via Endorsements and typically requires additional Claims about the signer's time synchronization mechanism.

In some use cases, however, a trustworthy clock might not be available. For example, in many Trusted Execution Environments (TEEs) today, a clock is only available outside the TEE and so cannot be trusted by the TEE.

10.2. Implicit Timekeeping using Nonces

A second approach places the onus of timekeeping solely on the Verifier (for Evidence) or the Relying Party (for Attestation Results), and might be suitable, for example, in case the Attester does not have a trustworthy clock or time synchronization is otherwise impaired. In this approach, a non-predictable nonce is sent by the appraising entity, and the nonce is then signed and included along with the Claims in the Evidence or Attestation Result. After checking that the sent and received nonces are the same, the appraising entity knows that the Claims were signed after the nonce was generated. This allows associating a "rough" epoch to the Evidence or Attestation Result. In this case the epoch is said to be rough because:

* The epoch applies to the entire Claim set instead of a more granular association, and

* The time between the creation of Claims and the collection of Claims is indistinguishable.

10.3. Implicit Timekeeping using Epoch IDs

A third approach relies on having epoch identifiers (or "IDs") periodically sent to both the sender and receiver of Evidence or Attestation Results by some "Epoch ID Distributor".

Epoch IDs are different from nonces as they can be used more than once and can even be used by more than one entity at the same time. Epoch IDs are different from timestamps as they do not have to convey information about a point in time, i.e., they are not necessarily monotonically increasing integers.

Like the nonce approach, this allows associating a "rough" epoch without requiring a trustworthy clock or time synchronization in order to generate or appraise the freshness of Evidence or Attestation Results. Only the Epoch ID Distributor requires access to a clock so it can periodically send new epoch IDs.

The most recent epoch ID is included in the produced Evidence or Attestation Results, and the appraising entity can compare the epoch ID in received Evidence or Attestation Results against the latest epoch ID it received from the Epoch ID Distributor to determine if it is within the current epoch. An actual solution also needs to take into account race conditions when transitioning to a new epoch, such as by using a counter signed by the Epoch ID Distributor as the epoch ID, or by including both the current and previous epoch IDs in messages and/or checks, by requiring retries in case of mismatching epoch IDs, or by buffering incoming messages that might be associated with a epoch ID that the receiver has not yet obtained.

More generally, in order to prevent an appraising entity from generating false negatives (e.g., discarding Evidence that is deemed stale even if it is not), the appraising entity should keep an "epoch window" consisting of the most recently received epoch IDs. The depth of such epoch window is directly proportional to the maximum network propagation delay between the first to receive the epoch ID and the last to receive the epoch ID, and it is inversely proportional to the epoch duration. The appraising entity shall compare the epoch ID carried in the received Evidence or Attestation Result with the epoch IDs in its epoch window to find a suitable match.

Whereas the nonce approach typically requires the appraising entity to keep state for each nonce generated, the epoch ID approach minimizes the state kept to be independent of the number of Attesters or Verifiers from which it expects to receive Evidence or Attestation Results, as long as all use the same Epoch ID Distributor.

10.4. Discussion

Implicit and explicit timekeeping can be combined into hybrid mechanisms. For example, if clocks exist and are considered trustworthy but are not synchronized, a nonce-based exchange may be used to determine the (relative) time offset between the involved peers, followed by any number of timestamp based exchanges.

It is important to note that the actual values in Claims might have been generated long before the Claims are signed. If so, it is the signer's responsibility to ensure that the values are still correct when they are signed. For example, values generated at boot time might have been saved to secure storage until network connectivity is established to the remote Verifier and a nonce is obtained.

A more detailed discussion with examples appears in Section 16.

For a discussion on the security of epoch IDs see Section 12.3.

11. Privacy Considerations

The conveyance of Evidence and the resulting Attestation Results reveal a great deal of information about the internal state of a device as well as potentially any users of the device. In many cases, the whole point of attestation procedures is to provide reliable information about the type of the device and the firmware/software that the device is running. This information might be particularly interesting to many attackers. For example, knowing that a device is running a weak version of firmware provides a way to aim attacks better.

Many Claims in Evidence and Attestation Results are potentially Personally Identifying Information (PII) depending on the end-to-end use case of the remote attestation procedure. Remote attestation that goes up to include containers and applications, e.g., a blood pressure monitor, may further reveal details about specific systems or users.

In some cases, an attacker may be able to make inferences about the contents of Evidence from the resulting effects or timing of the processing. For example, an attacker might be able to infer the value of specific Claims if it knew that only certain values were accepted by the Relying Party.

Evidence and Attestation Results are expected to be integrity protected (i.e., either via signing or a secure channel) and optionally might be confidentiality protected via encryption. If confidentiality protection via signing the conceptual messages is omitted or unavailable, the protecting protocols that convey Evidence or Attestation Results are responsible for detailing what kinds of information are disclosed, and to whom they are exposed.

As Evidence might contain sensitive or confidential information, Attesters are responsible for only sending such Evidence to trusted Verifiers. Some Attesters might want a stronger level of assurance of the trustworthiness of a Verifier before sending Evidence to it. In such cases, an Attester can first act as a Relying Party and ask for the Verifier's own Attestation Result, and appraising it just as a Relying Party would appraise an Attestation Result for any other purpose.

Another approach to deal with Evidence is to remove PII from the Evidence while still being able to verify that the Attester is one of a large set. This approach is often called "Direct Anonymous Attestation". See [CCC-DeepDive] section 6.2 for more discussion.

12. Security Considerations

This document provides an architecture for doing remote attestation. No specific wire protocol is documented here. Without a specific proposal to compare against, it is impossible to know if the security threats listed below have been mitigated well. The security considerations below should be read as being essentially requirements against realizations of the RATS Architecture. Some threats apply to protocols, some are against implementations (code), and some threats are against physical infrastructure (such as factories).

12.1. Attester and Attestation Key Protection

Implementers need to pay close attention to the protection of the Attester and the manufacturing processes for provisioning attestation key material. If either of these are compromised, intended levels of assurance for RATS are compromised because attackers can forge Evidence or manipulate the Attesting Environment. For example, a Target Environment should not be able to tamper with the Attesting Environment that measures it, by isolating the two environments from each other in some way.

Remote attestation applies to use cases with a range of security requirements, so the protections discussed here range from low to high security where low security may be limited to application or process isolation by the device's operating system, and high security may involve specialized hardware to defend against physical attacks on a chip.

12.1.1. On-Device Attester and Key Protection

It is assumed that an Attesting Environment is sufficiently isolated from the Target Environment it collects Claims about and that it signs the resulting Claims set with an attestation key, so that the Target Environment cannot forge Evidence about itself. Such an isolated environment might be provided by a process, a dedicated chip, a TEE, a virtual machine, or another secure mode of operation. The Attesting Environment must be protected from unauthorized modification to ensure it behaves correctly. Confidentiality protection of the Attesting Environment's signing key is vital so it cannot be misused to forge Evidence.

In many cases the user or owner of a device that includes the role of Attester must not be able to modify or extract keys from the Attesting Environments, to prevent creating forged Evidence. Some common examples include the user of a mobile phone or FIDO authenticator. An essential value-add provided by RATS is for the Relying Party to be able to trust the Attester even if the user or owner is not trusted.

Measures for a minimally protected system might include process or application isolation provided by a high-level operating system, and restricted access to root or system privileges. In contrast, For really simple single-use devices that don't use a protected mode operating system, like a Bluetooth speaker, the only factual isolation might be the sturdy housing of the device.

Measures for a moderately protected system could include a special restricted operating environment, such as a TEE. In this case, only security-oriented software has access to the Attester and key material.

Measures for a highly protected system could include specialized hardware that is used to provide protection against chip decapping attacks, power supply and clock glitching, faulting injection and RF and power side channel attacks.

12.1.2. Attestation Key Provisioning Processes

Attestation key provisioning is the process that occurs in the factory or elsewhere to establish signing key material on the device and the validation key material off the device. Sometimes this is procedure is referred to as personalization or customization.

12.1.2.1. Off-Device Key Generation

One way to provision key material is to first generate it external to the device and then copy the key onto the device. In this case, confidentiality protection of the generator, as well as for the path over which the key is provisioned, is necessary. The manufacturer needs to take care to protect corresponding key material with measures appropriate for its value.

The degree of protection afforded to this key material can vary by device, based upon considerations as to a cost/benefit evaluation of the intended function of the device. The confidentiality protection is fundamentally based upon some amount of physical protection: while encryption is often used to provide confidentiality when a key is conveyed across a factory, where the attestation key is created or applied, it must be available in an unencrypted form. The physical protection can therefore vary from situations where the key is unencrypted only within carefully controlled secure enclaves within silicon, to situations where an entire facility is considered secure, by the simple means of locked doors and limited access.

The cryptography that is used to enable confidentiality protection of the attestation key comes with its own requirements to be secured. This results in recursive problems, as the key material used to provision attestation keys must again somehow have been provisioned securely beforehand (requiring an additional level of protection, and so on).

So, this is why, in general, a combination of some physical security measures and some cryptographic measures is used to establish confidentiality protection.

12.1.2.2. On-Device Key Generation

When key material is generated within a device and the secret part of it never leaves the device, then the problem may lessen. For public-key cryptography, it is, by definition, not necessary to maintain confidentiality of the public key: however integrity of the chain of custody of the public key is necessary in order to avoid attacks where an attacker is able get a key they control endorsed.

To summarize: attestation key provisioning must ensure that only valid attestation key material is established in Attesters.

12.2. Integrity Protection

Any solution that conveys information used for security purposes, whether such information is in the form of Evidence, Attestation Results, Endorsements, or appraisal policy must support end-to-end integrity protection and replay attack prevention, and often also needs to support additional security properties, including:

* end-to-end encryption,

* denial of service protection,

* authentication,

* auditing,

* fine grained access controls, and

* logging.

Section 10 discusses ways in which freshness can be used in this architecture to protect against replay attacks.

To assess the security provided by a particular appraisal policy, it is important to understand the strength of the root of trust, e.g., whether it is mutable software, or firmware that is read-only after boot, or immutable hardware/ROM.

It is also important that the appraisal policy was itself obtained securely. If an attacker can configure appraisal policies for a Relying Party or for a Verifier, then integrity of the process is compromised.

Security protections in RATS may be applied at different layers, whether by a conveyance protocol, or an information encoding format. This architecture expects conceptual messages (see Section 8) to be end-to-end protected based on the role interaction context. For example, if an Attester produces Evidence that is relayed through some other entity that doesn't implement the Attester or the intended Verifier roles, then the relaying entity should not expect to have access to the Evidence.

12.3. Epoch ID-based Attestation

Epoch IDs, described in Section 10.3, can be tampered with, replayed, dropped, delayed, and reordered by an attacker.

An attacker could be either external or belong to the distribution group, for example, if one of the Attester entities have been compromised.

An attacker who is able to tamper with epoch IDs can potentially lock all the participants in a certain epoch of choice for ever, effectively freezing time. This is problematic since it destroys the ability to ascertain freshness of Evidence and Attestation Results.

To mitigate this threat, the transport should be at least integrity protected and provide origin authentication.

Selective dropping of epoch IDs is equivalent to pinning the victim node to a past epoch. An attacker could drop epoch IDs to only some entities and not others, which will typically result in a denial of service due to the permanent staleness of the Attestation Result or Evidence.

Delaying or reordering epoch IDs is equivalent to manipulating the victim's timeline at will. This ability could be used by a malicious actor (e.g., a compromised router) to mount a confusion attack where, for example, a Verifier is tricked into accepting Evidence coming from a past epoch as fresh, while in the meantime the Attester has been compromised.

Reordering and dropping attacks are mitigated if the transport provides the ability to detect reordering and drop. However, the delay attack described above can't be thwarted in this manner.

12.4. Trust Anchor Protection

As noted in Section 7, Verifiers and Relying Parties have trust anchor stores that must be secured. Specifically, a trust anchor store must resist modification against unauthorized insertion, deletion, and modification.

If certificates are used as trust anchors, Verifiers and Relying Parties are also responsible for validating the entire certificate path up to the trust anchor, which includes checking for certificate revocation. See Section 6 of [RFC5280] for details.

13. IANA Considerations

This document does not require any actions by IANA.

14. Acknowledgments

Special thanks go to Joerg Borchert, Nancy Cam-Winget, Jessica Fitzgerald-McKay, Diego Lopez, Laurence Lundblade, Paul Rowe, Hannes Tschofenig, Frank Xia, and David Wooten.

15. Notable Contributions

Thomas Hardjono created initial versions of the terminology section in collaboration with Ned Smith. Eric Voit provided the conceptual separation between Attestation Provision Flows and Attestation Evidence Flows. Monty Wisemen created the content structure of the first three architecture drafts. Carsten Bormann provided many of the motivational building blocks with respect to the Internet Threat Model.

16. Appendix A: Time Considerations

Section 10 discussed various issues and requirements around freshness of evidence, and summarized three approaches that might be used by different solutions to address them. This appendix provides more details with examples to help illustrate potential approaches, to inform those creating specific solutions.

The table below defines a number of relevant events, with an ID that is used in subsequent diagrams. The times of said events might be defined in terms of an absolute clock time, such as the Coordinated Universal Time timescale, or might be defined relative to some other timestamp or timeticks counter, such as a clock resetting its epoch each time it is powered on.

ID	Event	Explanation of event
VG	Value generated	A value to appear in a Claim was created. In some cases, a value may have technically existed before an Attester became aware of it but the Attester might have no idea how long it has had that value. In such a case, the Value created time is the time at which the Claim containing the copy of the value was created.
NS	Nonce sent	A nonce not predictable to an Attester (recentness & uniqueness) is sent to an Attester.
NR	Nonce	A nonce is relayed to an Attester by

	relayed	another entity.
IR	Epoch ID received	An epoch ID is successfully received and processed by an entity.
EG	Evidence generation	An Attester creates Evidence from collected Claims.
ER	Evidence relayed	A Relying Party relays Evidence to a Verifier.
RG	Result generation	A Verifier appraises Evidence and generates an Attestation Result.
RR	Result relayed	A Relying Party relays an Attestation Result to a Relying Party.
RA	Result appraised	The Relying Party appraises Attestation Results.
OP	Operation performed	The Relying Party performs some operation requested by the Attester via a resource access protocol as depicted in Figure 8, e.g., across a session created earlier at time(RA).
RX	Result expiry	An Attestation Result should no longer be accepted, according to the Verifier that generated it.

Table 1

Using the table above, a number of hypothetical examples of how a solution might be built are illustrated below. This list is not intended to be complete, but is just representative enough to highlight various timing considerations.

All times are relative to the local clocks, indicated by an "_a" (Attester), "_v" (Verifier), or "_r" (Relying Party) suffix.

Times with an appended Prime (’) indicate a second instance of the same event.

How and if clocks are synchronized depends upon the model.

In the figures below, curly braces indicate containment. For example, the notation Evidence{foo} indicates that 'foo' is contained in the Evidence and is thus covered by its signature.

16.1. Example 1: Timestamp-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party, which depends on using a secure clock synchronization mechanism. As a result, the receiver of a conceptual message containing a timestamp can directly compare it to its own clock and timestamps.

```
   .----------.                     .----------.  .---------------.
   | Attester |                     | Verifier |  | Relying Party |
   '----------'                     '----------'  '---------------'
     time(VG_a)                           |              |
        |                                 |              |
        ~                                 ~              ~
        |                                 |              |
     time(EG_a)                           |              |
        |------Evidence{time(EG_a)}------>|              |
        |                              time(RG_v)        |
        |<-----Attestation Result---------|              |
        |      {time(RG_v),time(RX_v)}    |              |
        ~                                                ~
        |                                                |
        |----Attestation Result{time(RG_v),time(RX_v)}-->time(RA_r)
        |                                                |
        ~                                                ~
        |                                                |
        |                                             time(OP_r)
```

The Verifier can check whether the Evidence is fresh when appraising it at time(RG_v) by checking "time(RG_v) - time(EG_a) < Threshold", where the Verifier's threshold is large enough to account for the maximum permitted clock skew between the Verifier and the Attester.

If time(VG_a) is also included in the Evidence along with the Claim value generated at that time, and the Verifier decides that it can trust the time(VG_a) value, the Verifier can also determine whether the Claim value is recent by checking "time(RG_v) - time(VG_a) < Threshold". The threshold is decided by the Appraisal Policy for Evidence, and again needs to take into account the maximum permitted clock skew between the Verifier and the Attester.

The Relying Party can check whether the Attestation Result is fresh when appraising it at time(RA_r) by checking "time(RA_r) - time(RG_v) < Threshold", where the Relying Party's threshold is large enough to account for the maximum permitted clock skew between the Relying Party and the Verifier. The result might then be used for some time (e.g., throughout the lifetime of a connection established at time(RA_r)). The Relying Party must be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain fresh enough. Thus, it might allow use (at time(OP_r)) as long as "time(OP_r) - time(RG_v) < Threshold". However, if the Attestation Result contains an expiry time time(RX_v) then it could explicitly check "time(OP_r) < time(RX_v)".

16.2. Example 2: Nonce-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses nonces instead of timestamps. Compared to the timestamp-based example, it requires an extra round trip to retrieve a nonce, and requires that the Verifier and Relying Party track state to remember the nonce for some period of time.

The advantage is that it does not require that any clocks are synchronized. As a result, the receiver of a conceptual message containing a timestamp cannot directly compare it to its own clock or timestamps. Thus we use a suffix ("a" for Attester, "v" for Verifier, and "r" for Relying Party) on the IDs below indicating which clock generated them, since times from different clocks cannot be compared. Only the delta between two events from the sender can be used by the receiver.

```
      .----------.                      .----------.  .---------------.
      | Attester |                      | Verifier |  | Relying Party |
      '----------'                      '----------'  '---------------'
        time(VG_a)                            |               |
           |                                  |               |
           ~                                  ~               ~
           |                                  |               |
           |<--Nonce1---------------------time(NS_v)          |
        time(EG_a)                            |               |
           |---Evidence---------------------->|               |
           | {Nonce1, time(EG_a)-time(VG_a)}  |               |
           |                              time(RG_v)          |
           |<--Attestation Result-------------|               |
           |   {time(RX_v)-time(RG_v)}        |               |
           ~                                                  ~
           |                                                  |
           |<--Nonce2-------------------------------------time(NS_r)
        time(RR_a)                                            |
           |--[Attestation Result{time(RX_v)-time(RG_v)},  -->|time(RA_r)
           |        Nonce2, time(RR_a)-time(EG_a)]            |
           ~                                                  ~
           |                                                  |
           |                                              time(OP_r)
```

In this example solution, the Verifier can check whether the Evidence is fresh at "time(RG_v)" by verifying that "time(RG_v)-time(NS_v) < Threshold".

The Verifier cannot, however, simply rely on a Nonce to determine whether the value of a Claim is recent, since the Claim value might have been generated long before the nonce was sent by the Verifier. However, if the Verifier decides that the Attester can be trusted to correctly provide the delta "time(EG_a)-time(VG_a)", then it can determine recency by checking "time(RG_v)-time(NS_v) + time(EG_a)-time(VG_a) < Threshold".

Similarly if, based on an Attestation Result from a Verifier it trusts, the Relying Party decides that the Attester can be trusted to correctly provide time deltas, then it can determine whether the Attestation Result is fresh by checking "time(OP_r)-time(NS_r) + time(RR_a)-time(EG_a) < Threshold". Although the Nonce2 and "time(RR_a)-time(EG_a)" values cannot be inside the Attestation Result, they might be signed by the Attester such that the Attestation Result vouches for the Attester's signing capability.

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of "time(RX_v)-time(RG_v)", then the Relying Party can check "time(OP_r)-time(NS_r) < time(RX_v)-time(RG_v)".

16.3. Example 3: Epoch ID-based Passport Model Example

The example in Figure 10 illustrates a hypothetical Passport Model solution that uses epoch IDs instead of nonces or timestamps.

The Epoch ID Distributor broadcasts epoch ID "I" which starts a new epoch "E" for a protocol participant upon reception at "time(IR)".

The Attester generates Evidence incorporating epoch ID "I" and conveys it to the Verifier.

The Verifier appraises that the received epoch ID "I" is "fresh" according to the definition provided in Section 10.3 whereby retries are required in the case of mismatching epoch IDs, and generates an Attestation Result. The Attestation Result is conveyed to the Attester.

After the transmission of epoch ID "I'" a new epoch "E'" is established when "I'" is received by each protocol participant. The Attester relays the Attestation Result obtained during epoch "E" (associated with epoch ID "I") to the Relying Party using the epoch ID for the current epoch "I'". If the Relying Party had not yet received "I'", then the Attestation Result would be rejected, but in this example, it is received.

In the illustrated scenario, the epoch ID for relaying an Attestation Result to the Relying Party is current, while a previous epoch ID was used to generate Verifier evaluated evidence. This indicates that at least one epoch transition has occurred, and the Attestation Results may only be as fresh as the previous epoch. If the Relying Party remembers the previous epoch ID "I" during an epoch window as discussed in Section 10.3, and the message is received during that window, the Attestation Result is accepted as fresh, and otherwise it is rejected as stale.

```
                  .-------------.
   .----------.   | Epoch ID    |   .----------.  .---------------.
   | Attester |   | Distributor |   | Verifier |  | Relying Party |
   '----------'   '-------------'   '----------'  '---------------'
     time(VG_a)          |                |                |
        |                |                |                |
        ~                ~                ~                ~
        |                |                |                |
     time(IR_a)<------I--+--I--------time(IR_v)----->time(IR_r)
        |                |                |                |
     time(EG_a)          |                |                |
        |---Evidence--------------------->|                |
        |   {I,time(EG_a)-time(VG_a)}     |                |
        |                |                |                |
        |                |           time(RG_v)            |
        |<--Attestation Result------------|                |
        |   {I,time(RX_v)-time(RG_v)}     |                |
        |                |                |                |
     time(IR'_a)<-----I'-+--I'-------time(IR'_v)---->time(IR'_r)
        |                |                |                |
        |---[Attestation Result--------------------->time(RA_r)
        |   {I,time(RX_v)-time(RG_v)},I'] |                |
        |                |                |                |
        ~                ~                ~                ~
        |                |                |                |
        |                |                |           time(OP_r)
```

Figure 10: Epoch ID-based Passport Model

16.4. Example 4: Timestamp-based Background-Check Model Example

The following example illustrates a hypothetical Background-Check Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party.

```
   .----------.            .---------------.                .----------.
   | Attester |            | Relying Party |                | Verifier |
   '----------'            '---------------'                '----------'
     time(VG_a)                   |                              |
          |                       |                              |
          ~                       ~                              ~
          |                       |                              |
     time(EG_a)                   |                              |
          |----Evidence------->|                              |
          |   {time(EG_a)} time(ER_r)--Evidence{time(EG_a)}->|
          |                       |                        time(RG_v)
          |                  time(RA_r)<-Attestation Result---|
          |                       |          {time(RX_v)}      |
          ~                       ~                              ~
          |                       |                              |
          |                  time(OP_r)                          |
```

The time considerations in this example are equivalent to those discussed under Example 1 above.

16.5. Example 5: Nonce-based Background-Check Model Example

The following example illustrates a hypothetical Background-Check Model solution that uses nonces and thus does not require that any clocks are synchronized. In this example solution, a nonce is generated by a Verifier at the request of a Relying Party, when the Relying Party needs to send one to an Attester.

```
   .----------.            .---------------.              .----------.
   | Attester |            | Relying Party |              | Verifier |
   '----------'            '---------------'              '----------'
     time(VG_a)                   |                            |
       |                          |                            |
       ~                          ~                            ~
       |                          |                            |
       |                          |<-------Nonce-----------time(NS_v)
       |<---Nonce-----------time(NR_r)                         |
     time(EG_a)                   |                            |
       |----Evidence{Nonce}--->|                            |
       |                     time(ER_r)--Evidence{Nonce}--->|
       |                          |                      time(RG_v)
       |                     time(RA_r)<-Attestation Result-|
       |                          |   {time(RX_v)-time(RG_v)} |
       ~                          ~                            ~
       |                          |                            |
       |                     time(OP_r)                        |
```

The Verifier can check whether the Evidence is fresh, and whether a Claim value is recent, the same as in Example 2 above.

However, unlike in Example 2, the Relying Party can use the Nonce to determine whether the Attestation Result is fresh, by verifying that "time(OP_r)-time(NR_r) < Threshold".

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of "time(RX_v)-time(RG_v)", then the Relying Party can check "time(OP_r)-time(ER_r) < time(RX_v)-time(RG_v)".

17. References

17.1. Normative References

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

17.2. Informative References

[CCC-DeepDive] Confidential Computing Consortium, "Confidential Computing Deep Dive", n.d., <https://confidentialcomputing.io/whitepaper-02-latest>.

[CTAP] FIDO Alliance, "Client to Authenticator Protocol", n.d., <https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html>.

[I-D.birkholz-rats-tuda]
Fuchs, A., Birkholz, H., McDonald, I. E., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-04, 13 January 2021, <https://tools.ietf.org/html/draft-birkholz-rats-tuda-04>.

[I-D.birkholz-rats-uccs]
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-birkholz-rats-uccs-03, 8 March 2021, <https://tools.ietf.org/html/draft-birkholz-rats-uccs-03>.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-14, 22 February 2021, <https://tools.ietf.org/html/draft-ietf-teep-architecture-14>.

[I-D.tschofenig-tls-cwt]
Tschofenig, H. and M. Brossard, "Using CBOR Web Tokens (CWTs) in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-tschofenig-tls-cwt-02, 13 July 2020, <https://tools.ietf.org/html/draft-tschofenig-tls-cwt-02>.

[OPCUA] OPC Foundation, "OPC Unified Architecture Specification, Part 2: Security Model, Release 1.03", OPC 10000-2 , 25 November 2015, <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/>.

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

[RFC5209] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <https://www.rfc-editor.org/rfc/rfc5209>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC8322] Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, <https://www.rfc-editor.org/rfc/rfc8322>.

[strengthoffunction] NISC, "Strength of Function", n.d., <https://csrc.nist.gov/glossary/term/strength_of_function>.

[TCG-DICE] Trusted Computing Group, "DICE Certificate Profiles", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Certificate-Profiles-r01_3june2020-1.pdf>.

[TCGarch] Trusted Computing Group, "Trusted Platform Module Library - Part 1: Architecture", 8 November 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf>.

[WebAuthN] W3C, "Web Authentication: An API for accessing Public Key Credentials", n.d., <https://www.w3.org/TR/webauthn-1/>.

Contributors

Monty Wiseman

Email: montywiseman32@gmail.com

Liang Xia

Email: frank.xialiang@huawei.com

Laurence Lundblade

Email: lgl@island-resort.com

Eliot Lear

Email: elear@cisco.com

Jessica Fitzgerald-McKay

Sarah C. Helbe

Andrew Guinn

Peter Loscocco

Email: pete.loscocco@gmail.com

Eric Voit

Thomas Fossati

Email: thomas.fossati@arm.com

Paul Rowe

Carsten Bormann

Email: cabo@tzi.org

Giri Mandyam

Email: mandyam@qti.qualcomm.com

Kathleen Moriarty

Email: kathleen.moriarty.ietf@gmail.com

Guy Fedorkow

Email: gfedorkow@juniper.net

Simon Frost

Email: Simon.Frost@arm.com

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Dave Thaler
Microsoft
United States of America

Email: dthaler@microsoft.com

Michael Richardson
Sandelman Software Works
Canada

Email: mcr+ietf@sandelman.ca

Ned Smith
Intel Corporation
United States of America

Email: ned.smith@intel.com

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com

RATS Working Group G. Mandyam
Internet-Draft Qualcomm Technologies Inc.
Intended status: Standards Track L. Lundblade
Expires: September 6, 2021 Security Theory LLC
M. Ballesteros
J. O'Donoghue
Qualcomm Technologies Inc.
March 05, 2021

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-09

Abstract

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

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

Contributing

TBD

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

TODO: mention use for Attestation Evidence and Results.

1.1. CWT, JWT and UCCS

For flexibility and ease of imlpementation in a wide variety of environments, EATs can be either CBOR [RFC8949] or JSON [ECMAScript] format. This specification simultaneously describes both formats.

An EAT is either a CWT as defined in [RFC8392], a UCCS as defined in [UCCS.Draft], or a JWT as defined in [RFC7519]. This specification extends those specifications with additional claims for attestation.

The identification of a protocol element as an EAT, whether CBOR or JSON format, follows the general conventions used by CWT, JWT and UCCS. Largely this depends on the protocol carrying the EAT. In some cases it may be by content type (e.g., MIME type). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

1.2. CDDL

This specification uses CDDL, [RFC8610], as the primary formalism to define each claim. The implementor then interprets the CDDL to come to either the CBOR [RFC8949] or JSON [ECMAScript] representation. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 6.3.2 of this document where Appendix E is insufficient. (Note that this is not to define a general means to translate between CBOR and JSON, but only to define enough such that the claims defined in this document can be rendered unambiguously in JSON).

The CWT specification was authored before CDDL was available and did not use it. This specification includes a CDDL definition of most of what is described in [RFC8392].

1.3. 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.

1.4. EAT Operating Models

TODO: Rewrite (or eliminate) this section in light of the RATS architecture draft.

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

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

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

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

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

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

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

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

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

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

All these operating models are supported and there is no preference of one over the other. It is important to support this variety of operating models to generally facilitate deployment and to allow for some special scenarios. One special scenario has a validation service that is monetized, most likely by the manufacturer. In another, a privacy proxy service processes the EAT before it is transmitted to the relying party. In yet another, symmetric key material is used for signing. In this case the manufacturer should perform the verification, because any release of the key material would enable a participant other than the entity to create valid signed EATs.

1.5. What is Not Standardized

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

1.5.1. Transmission Protocol

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

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

1.5.2. Signing Scheme

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

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

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

2. Terminology

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

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

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

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

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

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

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

3. The Claims

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

Note also: * Any claim defined for CWT or JWT may be used in an EAT including those in the CWT [IANA.CWT.Claims] and JWT IANA [IANA.JWT.Claims] claims registries.

- All claims are optional

- No claims are mandatory

- All claims that are not understood by implementations MUST be ignored

There are no default values or meanings assigned to absent claims other than they are not reported. The reason for a claim's absence may be the implementation not supporting the claim, an inability to determine its value, or a preference to report in a different way such as a proprietary claim.

CDDL along with text descriptions is used to define each claim indepdent of encoding. Each claim is defined as a CDDL group (the group is a general aggregation and type definition feature of CDDL). In the encoding section Section 6, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

TODO: add paragraph here about use for Attestation Evidence and for Results.

3.1. Token ID Claim (cti and jti)

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

3.2. Timestamp claim (iat)

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

The data for some claims may be held or cached for some period of time before the token is created. This period may be long, even days. Examples are measurements taken at boot or a geographic position fix taken the last time a satellite signal was received. There are individual timestamps associated with these claims to indicate their age is older than the "iat" timestamp.

CWT allows the use floating-point for this claim. EAT disallows the use of floating-point. No token may contain an iat claim in float-point format. Any recipient of a token with a floating-point format iat claim may consider it an error. A 64-bit integer representation of epoch time can represent a range of +/- 500 billion years, so the only point of a floating-point timestamp is to have precession greater than one second. This is not needed for EAT.

3.3. Nonce Claim (nonce)

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

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

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

Multiple nonces are allowed to accommodate multistage verification and consumption.

3.3.1. nonce CDDL

```
nonce-type = bstr .size (8..64)

nonce-claim = (
    nonce => nonce-type / [ 2* nonce-type ]
)
```

3.4. Universal Entity ID Claim (ueid)

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

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

There are privacy considerations for UEID's. See Section 8.1.

The UEID should be permanent. It should never change for a given device / entity. In addition, it should not be reprogrammable. UEID's are variable length. All implementations MUST be able to receive UEID's that are 33 bytes long (1 type byte and 256 bits). The recommended maximum sent is also 33 bytes.

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

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

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

Table 1: UEID Composition Types

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

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

- o UEIDs types may vary freely from one manufacturer to the next.
- o New types of UEIDs may be created. For example, a type 0x07 UEID may be created based on some other manufacturer registration scheme.
- o 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.

3.4.1. ueid CDDL

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

ueid-claim = (
     ueid => ueid-type
)
```

3.5. Origination Claim (origination)

TODO: this claim is likely to be dropped in favor of Endorsement identifier and locators.

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:

Name	Description
Acme-TEE	The EATs are generated in the TEE authored and configured by "Acme"
Acme-TPM	The EATs are generated in a TPM manufactured by "Acme"
Acme-Linux-Kernel	The EATs are generated in a Linux kernel configured and shipped by "Acme"
Acme-TA	The EATs are generated in a Trusted Application (TA) authored by "Acme"

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.

3.5.1. origination CDDL

```
origination-claim = (
    origination => string-or-uri
)
```

3.6. OEM Identification by IEEE (oemid)

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

Companies that have more than one of these IDs or MAC address blocks should pick one and prefer that for all their devices.

Commonly, these are expressed in Hexadecimal Representation [IEEE.802-2001] also called the Canonical format. When this claim is encoded the order of bytes in the bstr are the same as the order in the Hexadecimal Representation. For example, an MA-L like "AC-DE-48" would be encoded in 3 bytes with values 0xAC, 0xDE, 0x48. For JSON encoded tokens, this is further base64url encoded.

3.6.1. oemid CDDL

```
oemid-claim = (
    oemid => bstr
)
```

3.7. Hardware Version Claims (hardware-version-claims)

The hardware version can be claimed at three different levels, the chip, the circuit board and the final device assembly. An EAT can include any combination these claims.

The hardware version is a simple text string the format of which is set by each manufacturer. The structure and sorting order of this text string can be specified using the version-scheme item from CoSWID [CoSWID].

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

Both the simple version string and EAN-13 versions may be included for the same hardware.

```
chip-version-claim = (
    chip-version => tstr
)

chip-version-scheme-claim = (
    chip-version-scheme => $version-scheme
)

board-version-claim = (
    board-version => tstr
)

board-version-scheme-claim = (
    board-version-scheme => $version-scheme
)

device-version-claim = (
    device-version => tstr
)
```

```
device-version-scheme-claim = (
    device-version-scheme => $version-scheme
)


ean-type = text .regexp "[0-9]{13}"

ean-chip-version-claim = (
    ean-chip-version => ean-type
)

ean-board-version-claim = (
    ean-board-version => ean-type
)

ean-device-version-claim = (
    ean-device-version => ean-type
)

hardware-version-claims = (
    ? chip-version-claim,
    ? board-version-claim,
    ? device-version-claim,
    ? chip-version-scheme-claim,
    ? board-version-scheme-claim,
    ? device-version-scheme-claim,
    ? ean-chip-version-claim,
    ? ean-board-version-claim,
    ? ean-device-version-claim,
)
```

3.8. Software Description and Version

TODO: Add claims that reference CoSWID.

3.9. The Security Level Claim (security-level)

This claim characterizes the device/entity ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is done by defining four security levels as described below. This is similar to the key protection types defined by the Fast Identity Online (FIDO) Alliance [FIDO.Registry].

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

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

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

3 - Secure Restricted Entities at this level must meet the criteria defined by FIDO Allowed Restricted Operating Environments [FIDO.AROE]. 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.

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

This claim is not intended as a replacement for a proper end-device security certification schemes such as those based on FIPS 140 [FIPS-140] or those based on Common Criteria [Common.Criteria]. The claim made here is solely a self-claim made by the Entity Originator.

3.9.1. security-level CDDL

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

security-level-claim = (
    security-level => security-level-type
)
```

3.10. Secure Boot Claim (secure-boot)

The value of true indicates secure boot is enabled. Secure boot is considered enabled when base software, the firmware and operating system, are under control of the entity manufacturer identified in the oemid claimd described in Section 3.6. This may because the software is in ROM or because it is cryptographically authenticated or some combination of the two or other.

3.10.1. secure-boot CDDL

```
secure-boot-claim = (
    secure-boot => bool
)
```

3.11. Debug Status Claim (debug-status)

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

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

As with all claims, the absence of the debug level claim means it is not reported. A conservative interpretation might assume the Not Disabled state. It could however be that it is reported in a proprietary claim.

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

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

There is no inheritance of claims from a submodule to a superior module or vice versa. There is no assumption, requirement or guarantee that the target of a superior module encompasses the targets of submodules. Thus, every submodule must explicitly describe its own debug state. The verifier or relying party receiving an EAT cannot assume that debug is turned off in a submodule because there is a claim indicating it is turned off in a superior module.

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

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

3.11.1. Enabled

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

3.11.2. Disabled

This level indicates all debug facilities are currently disabled. It may be possible to enable them in the future, and it may also be possible that they were enabled in the past after the target device/sub-system booted/started, but they are currently disabled.

3.11.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the target device/sub-system booted/started.

3.11.4. Disabled Permanently

This level indicates all non-manufacturer facilities are permanently disabled such that no end user or developer cannot enable them. Only the manufacturer indicated in the OEMID claim can enable them. This also indicates that all debug facilities are currently disabled and have been so since boot/start.

3.11.5. Disabled Fully and Permanently

This level indicates that all debug capabilities for the target device/sub-module are permanently disabled.

3.11.6. debug-status CDDL

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

debug-status-claim = (
    debug-status => debug-status-type
)
```

3.12. Including Keys

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

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

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

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

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

3.13. The Location Claim (location)

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

When encoding floating-point numbers half-precision should not be used. It usually does not provide enough precision for a geographic location. It is not a requirement that the receiver of an EAT implement half-precision, so the receiver may not be able to decode the location.

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

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

See location-related privacy considerations in Section 8.2 below.

3.13.1. location CDDL

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

latitude = 1
longitude = 2
altitude = 3
accuracy = 4
altitude-accuracy = 5
heading = 6
speed = 7
timestamp = 8
age = 9

location-claim = (
    location => location-type
)
```

3.14. The Uptime Claim (uptime)

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

3.14.1. uptime CDDL

```
uptime-claim = (
    uptime => uint
)
```

3.14.2. The Boot Seed Claim (boot-seed)

The Boot Seed claim is a random value created at system boot time that will allow differentiation of reports from different boot sessions. This value is usually public and not protected. It is not the same as a seed for a random number generator which must be kept secret.

```
boot-seed-claim = (
    boot-seed => bytes
)
```

3.15. The Intended Use Claim (intended-use)

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

1 - Generic Generic attestation describes an application where the EAT consumer requres the most up-to-date security assessment of the attesting entity. It is expected that this is the most commonly-used application of EAT.

2- Registration Entities that are registering for a new service may be expected to provide an attestation as part of the registration process. This intended-use setting indicates that the attestation is not intended for any use but registration.

3 - Provisioning Entities may be provisioned with different values or settings by an EAT consumer. Examples include key material or device management trees. The consumer may require an EAT to assess device security state of the entity prior to provisioning.

4 - Certificate Issuance (Certificate Signing Request) Certifying authorities (CA's) may require attestations prior to the issuance of certificates related to keypairs hosted at the entity. An EAT may be used as part of the certificate signing request (CSR).

5 - Proof-of-Possession An EAT consumer may require an attestation as part of an accompanying proof-of-possession (PoP) appication. More precisely, a PoP transaction is intended to provide to the recipient cryptographically-verifiable proof that the sender has posession of a key. This kind of attestation may be neceesary to verify the security state of the entity storing the private key used in a PoP application.

3.15.1. intended-use CDDL

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

intended-use-claim = (
    intended-use => intended-use-type
)
```

3.16. The Profile Claim (profile)

See Section 5 for the detailed description of a profile.

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

The OID is encoded in CBOR according to [CBOR-OID] and the URI according to [RFC8949]. Both are unwrapped and thus not tags. The OID is always absolute and never relative. If the claims CBOR type is a text string it is a URI and if a byte string it is an OID.

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

```
profile-claim = (
    profile => ~uri / ~oid
)
```

3.17. The Submodules Part of a Token (submods)

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

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

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

3.17.1. Two Types of Submodules

Each entry in the submod map is one of two types:

- o A non-token submodule that is a map or object directly containing claims for the submodule.
- o A nested EAT that is a fully formed, independently signed EAT token

3.17.1.1. Non-token Submodules

This is simply a map or object containing claims about the submodule.

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

It is signed/encrypted along with the rest of the token and thus the claims are secured by the same Attester with the same signing key as the rest of the token.

If a token is in CBOR format (a CWT or a UCCS), all non-token submodules must be CBOR format. If a token in in JSON format (a JWT), all non-token submodules must be in JSON format.

When decoding, this type of submodule is recognized from the other type by being a data item of type map for CBOR or type object for JSON.

3.17.1.2. Nested EATs

This type of submodule is a fully formed secured EAT as defined in this document except that it MUST NOT be a UCCS or an unsecured JWT. A nested token that is one that is always secured using COSE or JOSE, usually by an independent Attester. When the surrounding EAT is a CWT or secured JWT, the nested token becomes securely bound with the other claims in the surrounding token.

It is allowed to have a CWT as a submodule in a JWT and vice versa, but this SHOULD be avoided unless necessary.

3.17.1.2.1. Surrounding EAT is CBOR format

They type of an EAT nested in a CWT is determined by whether the CBOR type is a text string or a byte string. If a text string, then it is a JWT. If a byte string, then it is a CWT.

A CWT nested in a CBOR-format token is always wrapped by a byte string for easier handling with standard CBOR decoders and token processing APIs that will typically take a byte buffer as input.

Nested CWTs may be either a CWT CBOR tag or a CWT Protocol Message. COSE layers in nested CWT EATs MUST be a COSE_Tagged_Message, never a COSE_Untagged_Message. If a nested EAT has more than one level of COSE, for example one that is both encrypted and signed, a COSE_Tagged_message must be used at every level.

3.17.1.2.2. Surrounding EAT is JSON format

When a CWT is nested in a JWT, it must be as a 55799 tag in order to distinguish it from a nested JWT.

When a nested EAT in a JWT is decoded, first remove the base64url encoding. Next, check to see if it starts with the bytes 0xd9d9f7. If so, then it is a CWT as a JWT will never start with these four bytes. If not if it is a JWT.

Other than the 55799 tag requirement, tag usage for CWT's nested in a JSON format token follow the same rules as for CWTs nested in CBOR-format tokens. It may be a CWT CBOR tag or a CWT Protocol Message and COSE_Tagged_Message MUST be used at all COSE layers.

3.17.1.3. Unsecured JWTs and UCCS Tokens as Submodules

To incorporate a UCCS token as a submodule, it MUST be as a non-token submodule. This can be accomplished inserting the content of the UCCS Tag into the submodule map. The content of a UCCS tag is exactly a map of claims as required for a non-token submodule. If the UCCS is not a UCCS tag, then it can just be inserted into the submodule map directly.

The definition of a nested EAT type of submodule is that it is one that is secured (signed) by an Attester. Since UCCS tokens are unsecured, they do not fulfill this definition and must be non-token submodules.

To incorporate an Unsecured JWT as a submodule, the null-security JOSE wrapping should be removed. The resulting claims set should be inserted as a non-token submodule.

To incorporate a UCCS token in a surrounding JSON token, the UCCS token claims should be translated from CBOR to JSON. To incorporate an Unsecured JWT into a surrounding CBOR-format token, the null-security JOSE should be removed and the claims translated from JSON to CBOR.

3.17.2. No Inheritance

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

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

3.17.3. Security Levels

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

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

3.17.4. Submodule Names

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

3.17.5. submods CDDL

```
; The part of a token that contains all the submodules.  It is a peer
; with the claims in the token, but not a claim, only a map/object to
; hold all the submodules.

submods-part = (
    submods => submods-type
)

submods-type = { + submod-type }


; The type of a submodule which can either be a nested claim set or a
; nested separately signed token. Nested tokens are wrapped in a bstr
; or a tstr.

submod-type = (
    submod-name => eat-claim-set / nested-token
)


; When this is a bstr, the contents are an eat-token in CWT or UCCS
; format.  When this is a tstr, the contents are an eat-token in JWT
; format.

nested-token = bstr / tstr;


; Each submodule has a unique text string name.

submod-name = tstr
```

4. Endorsements and Verification Keys

TODO: fill this section in. It will discuss key IDs, endorsement ID and such that are needed as input needed to by the Verifier to verify the signature. This will NOT discuss the contents of an Endorsement, just and ID/locator.

5. Profiles

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

The profile can be named in the token using the profile claim described in Section 3.16.

5.1. List of Profile Issues

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

5.1.1. Use of JSON, CBOR or both

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

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

5.1.2. CBOR Map and Array Encoding

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

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

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

5.1.3. CBOR String Encoding

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

5.1.4. COSE/JOSE Protection

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

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

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

5.1.5. COSE/JOSE Algorithms

The profile document should list the COSE algorithms that a Verifier must implement. The Attester will select one of them. Since there is no negotiation, the Verifier should implement all algorithms listed in the profile.

5.1.6. Verification Key Identification

Section Section 4 describes a number of methods for identifying a verification key. The profile document should specify one of these or one that is not described. The ones described in this document are only roughly described. The profile document should go into the full detail.

5.1.7. Endorsement Identification

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

5.1.8. Required Claims

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

5.1.9. Prohibited Claims

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

5.1.10. Additional Claims

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

5.1.11. Refined Claim Definition

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

5.1.12. CBOR Tags

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

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

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

6. Encoding

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

Some of the CDDL included here is for claims that are defined in CWT [RFC8392] or JWT [RFC7519] or are in the IANA CWT or JWT registries. CDDL was not in use when these claims where defined.

6.1. Common CDDL Types

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

```
string-or-uri = tstr

time-int = #6.1(int)
```

6.2. CDDL for CWT-defined Claims

This section provides CDDL for the claims defined in CWT. It is non-normative as [RFC8392] is the authoritative definition of these claims.

```
   $$eat-extension //= (
       ? issuer => text,
       ? subject => text,
       ? audience => text,
       ? expiration => time,
       ? not-before => time,
       ? issued-at => time,
       ? cwt-id => bytes,
   )

   issuer = 1
   subject = 2
   audience = 3
   expiration = 4
   not-before = 5
   issued-at = 6
   cwt-id = 7
```

6.3. JSON

6.3.1. JSON Labels

```
; The following are Claim Keys (labels) assigned for JSON-encoded tokens.

ueid /= "ueid"
nonce /= "nonce"
origination /= "origination"
oemid /= "oemid"
security-level /= "seclevel"
secure-boot /= "secboot"
debug-status /= "dbgstat"
location /= "location"
uptime /= "uptime"
profile /= "eat-profile"
intended-use /= "intuse"
boot-seed /= "bootseed"
submods /= "submods"
timestamp /= "timestamp"

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

6.3.2. JSON Interoperability

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

- o bstr - must be base64url encoded
- o time - must be encoded as NumericDate as described section 2 of [RFC7519].
- o string-or-uri - must be encoded as StringOrURI as described section 2 of [RFC7519].
- o uri - must be a URI [RFC3986].
- o oid - encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1").

6.4. CBOR

6.4.1. CBOR Interoperability

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

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

One way to guarantee interoperability is to clearly specify CBOR serialization in a profile document. See Section 5 for a list of serialization issues that should be addressed.

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

6.4.1.1. EAT Constrained Device Serialization

- o Preferred serialization described in section 4.1 of [RFC8949] is not required. The EAT decoder must accept all forms of number serialization. The EAT encoder may use any form it wishes.

- o The EAT decoder must accept indefinite length arrays and maps as described in section 3.2.2 of [RFC8949]. The EAT encoder may use indefinite length arrays and maps if it wishes.

- o The EAT decoder must accept indefinite length strings as described in section 3.2.3 of [RFC8949]. The EAT encoder may use indefinite length strings if it wishes.

- o Sorting of maps by key is not required. The EAT decoder must not rely on sorting.

- o Deterministic encoding described in Section 4.2 of [RFC8949] is not required.

- o Basic validity described in section 5.3.1 of [RFC8949] must be followed. The EAT encoder must not send duplicate map keys/labels or invalid UTF-8 strings.

6.5. Collected CDDL

```
; This is the top-level definition of the claims in EAT tokens.  To
; form an actual EAT Token, this claim set is enclosed in a COSE, JOSE
; or UCCS message.

eat-claim-set = {
    ? ueid-claim,
    ? nonce-claim,
    ? origination-claim,
    ? oemid-claim,
    ? hardware-version-claims,
    ? security-level-claim,
    ? secure-boot-claim,
    ? debug-status-claim,
    ? location-claim,
    ? uptime-claim,
    ? submods-part,
    * $$eat-extension,
}


; This is the top-level definition of an EAT Token.  It is a CWT, JWT
; or UCSS where the payload is an eat-claim-set. A JWT_Message is what
; is defined by JWT in RFC 7519. (RFC 7519 doesn't use CDDL so a there
; is no actual CDDL definition of JWT_Message).

eat-token = EAT_Tagged_Message / EAT_Untagged_Message / JWT_Message
```

```
; This is CBOR-format EAT token in the CWT or UCCS format that is a
; tag.  COSE_Tagged_message is defined in RFC 8152.  Tag 601 is
; proposed by the UCCS draft, but not yet assigned.

EAT_Tagged_Message = #6.61(COSE_Tagged_Message) / #6.601(eat-claim-set)


; This is a CBOR-format EAT token that is a CWT or UCSS that is not a
; tag COSE_Tagged_message and COSE_Untagged_Message are defined in RFC
; 8152.

EAT_Untagged_Message = COSE_Tagged_Message / COSE_Untagged_Message / UCCS_Untagge
d_Message


; This is an "unwrapped" UCCS tag. Unwrapping a tag means to use the
; definition of its content without the preceding type 6 tag
; integer. Since a UCCS is nothing but a tag for an unsecured CWT
; claim set, unwrapping reduces to a bare eat-claim-set.

UCCS_Untagged_Message = eat-claim-set

string-or-uri = tstr

time-int = #6.1(int)
$$eat-extension //= (
    ? issuer => text,
    ? subject => text,
    ? audience => text,
    ? expiration => time,
    ? not-before => time,
    ? issued-at => time,
    ? cwt-id => bytes,
)

issuer = 1
subject = 2
audience = 3
expiration = 4
not-before = 5
issued-at = 6
cwt-id = 7

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

```
)

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

latitude = 1
longitude = 2
altitude = 3
accuracy = 4
altitude-accuracy = 5
heading = 6
speed = 7
timestamp = 8
age = 9

location-claim = (
    location => location-type
)
nonce-type = bstr .size (8..64)

nonce-claim = (
    nonce => nonce-type / [ 2* nonce-type ]
)
oemid-claim = (
    oemid => bstr
)
; copied from CoSWID
; TODO: how to properly make reference to CoSWID and have tool validate

  $version-scheme /= multipartnumeric
  $version-scheme /= multipartnumeric-suffix
  $version-scheme /= alphanumeric
  $version-scheme /= decimal
  $version-scheme /= semver
  $version-scheme /= uint / text
  multipartnumeric = 1
```

```
  multipartnumeric-suffix = 2
  alphanumeric = 3
  decimal = 4
  semver = 16384

chip-version-claim = (
    chip-version => tstr
)

chip-version-scheme-claim = (
    chip-version-scheme => $version-scheme
)

board-version-claim = (
    board-version => tstr
)

board-version-scheme-claim = (
    board-version-scheme => $version-scheme
)

device-version-claim = (
    device-version => tstr
)

device-version-scheme-claim = (
    device-version-scheme => $version-scheme
)


ean-type = text .regexp "[0-9]{13}"

ean-chip-version-claim = (
    ean-chip-version => ean-type
)

ean-board-version-claim = (
    ean-board-version => ean-type
)

ean-device-version-claim = (
    ean-device-version => ean-type
)

hardware-version-claims = (
    ? chip-version-claim,
    ? board-version-claim,
    ? device-version-claim,
```

```
    ? chip-version-scheme-claim,
    ? board-version-scheme-claim,
    ? device-version-scheme-claim,
    ? ean-chip-version-claim,
    ? ean-board-version-claim,
    ? ean-device-version-claim,
)

origination-claim = (
    origination => string-or-uri
)
secure-boot-claim = (
    secure-boot => bool
)
security-level-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)

security-level-claim = (
    security-level => security-level-type
)
; The part of a token that contains all the submodules.  It is a peer
; with the claims in the token, but not a claim, only a map/object to
; hold all the submodules.

submods-part = (
    submods => submods-type
)

submods-type = { + submod-type }


; The type of a submodule which can either be a nested claim set or a
; nested separately signed token. Nested tokens are wrapped in a bstr
; or a tstr.

submod-type = (
    submod-name => eat-claim-set / nested-token
)


; When this is a bstr, the contents are an eat-token in CWT or UCCS
; format.  When this is a tstr, the contents are an eat-token in JWT
; format.
```

```
nested-token = bstr / tstr;


; Each submodule has a unique text string name.

submod-name = tstr


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

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

intended-use-claim = (
    intended-use => intended-use-type
)
uptime-claim = (
    uptime => uint
)
; The following Claim Keys (labels) are pre-assigned by IANA.
; They are for CBOR-based tokens (CWT and UCCS).
; They are not expected to change in the final publication as an RFC.

nonce = 10
ueid = 11
oemid = 13
security-level = 14
secure-boot = 15
debug-status = 16
location = 17
profile = 18
submods = 20

; These are not yet assigned in any way and may change.
; These are intentionally above 24 so as to not use up
; single-byte labels.

origination = <TBD30>
uptime = <TBD31>
chip-version = <TBD32>
```

```
board-version = <TBD33>
device-version = <TBD34>
chip-version-scheme = <TBD35>
board-version-scheme = <TBD36>
device-version-scheme = <TBD37>
ean-chip-version = <TBD38>
ean-board-version = <TBD39>
ean-device-version = <TBD40>
intended-use = <TBD41>
; The following are Claim Keys (labels) assigned for JSON-encoded tokens.

ueid /= "ueid"
nonce /= "nonce"
origination /= "origination"
oemid /= "oemid"
security-level /= "seclevel"
secure-boot /= "secboot"
debug-status /= "dbgstat"
location /= "location"
uptime /= "uptime"
profile /= "eat-profile"
intended-use /= "intuse"
boot-seed /= "bootseed"
submods /= "submods"
timestamp /= "timestamp"

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

7. IANA Considerations

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

Claims defined for EAT are compatible with those of CWT so the CWT Claims Registry is re used. No new IANA registry is created. All EAT claims should be registered in the CWT and JWT Claims Registries.

7.2. Claim Characteristics

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

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

7.2.1. Interoperability and Relying Party Orientation

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

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

7.2.2. Operating System and Technology Neutral

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

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

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

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

7.2.3. Security Level Neutral

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

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

7.2.4. Reuse of Extant Data Formats

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

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

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

7.2.5. Proprietary Claims

EAT allows the definition and use of proprietary claims.

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

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

7.3. Claims Registered by This Document

This specification adds the following values to the "JSON Web Token Claims" registry established by [RFC7519] and the "CBOR Web Token Claims Registry" established by [RFC8392]. Each entry below is an addition to both registries (except for the nonce claim which is already registered for JWT, but not registered for CWT).

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

7.3.1. Claims for Early Assignment

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

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

- Claim Name: Nonce

- Claim Description: Nonce

- JWT Claim Name: "nonce" (already registered for JWT)

- Claim Key: 10

- Claim Value Type(s): byte string

- Change Controller: IESG

- Specification Document(s): [OpenIDConnectCore], *this document*

- Claim Name: UEID

- Claim Description: The Universal Entity ID

- JWT Claim Name: "ueid"

- CWT Claim Key: 11

- Claim Value Type(s): byte string

- Change Controller: IESG

- Specification Document(s): *this document*

- Claim Name: OEMID

- Claim Description: IEEE-based OEM ID

- JWT Claim Name: "oemid"

- o Claim Key: 13
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Security Level
- o Claim Description: Characterization of the security of an Attester or submodule
- o JWT Claim Name: "seclevel"
- o Claim Key: 14
- o Claim Value Type(s): integer
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Secure Boot
- o Claim Description: Indicate whether the boot was secure
- o JWT Claim Name: "secboot"
- o Claim Key: 15
- o Claim Value Type(s): Boolean
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Debug Status
- o Claim Description: Indicate status of debug facilities
- o JWT Claim Name: "dbgstat"
- o Claim Key: 16
- o Claim Value Type(s): integer
- o Change Controller: IESG

- o Specification Document(s): *this document*

- o Claim Name: Location

- o Claim Description: The geographic location

- o JWT Claim Name: "location"

- o Claim Key: 17

- o Claim Value Type(s): map

- o Change Controller: IESG

- o Specification Document(s): *this document*

- o Claim Name: Profile

- o Claim Description: Indicates the EAT profile followed

- o JWT Claim Name: "eat_profile"

- o Claim Key: 18

- o Claim Value Type(s): map

- o Change Controller: IESG

- o Specification Document(s): *this document*

- o Claim Name: Submodules Section

- o Claim Description: The section containing submodules (not actually a claim)

- o JWT Claim Name: "submods"

- o Claim Key: 20

- o Claim Value Type(s): map

- o Change Controller: IESG

- o Specification Document(s): *this document*

7.3.2. To be Assigned Claims

TODO: add the rest of the claims in here

8. 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.

8.1. 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:

- o 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.

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

- o 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.

8.2. Location Privacy Considerations

Geographic location is most always considered personally identifiable information. Implementers should consider laws and regulations governing the transmission of location data from end user devices to servers and services. Implementers should consider using location management facilities offered by the operating system on the device generating the attestation. For example, many mobile phones prompt the user for permission when before sending location data.

9. Security Considerations

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

9.1. Key Provisioning

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

9.1.1. Transmission of Key Material

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

9.2. Transport Security

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

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

9.3. Multiple EAT Consumers

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

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

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

10. References

10.1. Normative References

[CBOR-OID]
Bormann, C. and S. Leonard, "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", draft-ietf-cbor-tags-oid-04 (work in progress), January 2021.

[CoSWID] "Concise Software Identification Tags", November 2020, <https://tools.ietf.org/html/draft-ietf-sacm-coswid-16>.

[EAN-13] GS1, "International Article Number - EAN/UPC barcodes", 2019, <https://www.gs1.org/standards/barcodes/ean-upc>.

[FIDO.AROE]
The FIDO Alliance, "FIDO Authenticator Allowed Restricted Operating Environments List", November 2019, <https://fidoalliance.org/specs/fido-uaf-v1.0-fd-20191115/fido-allowed-AROE-v1.0-fd-20191115.html>.

[IANA.CWT.Claims]
IANA, "CBOR Web Token (CWT) Claims", <http://www.iana.org/assignments/cwt>.

[IANA.JWT.Claims]
IANA, "JSON Web Token (JWT) Claims", <https://www.iana.org/assignments/jwt>.

[OpenIDConnectCore]
Sakimura, N., Bradley, J., Jones, M., Medeiros, B. D., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/info/rfc8747>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[ThreeGPP.IMEI]
3GPP, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification", 2019, <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729>.

[UCCS.Draft]
Birkholz, H., "A CBOR Tag for Unprotected CWT Claims Sets", 2020, <https://tools.ietf.org/html/draft-birkholz-rats-uccs-01>.

[WGS84] National Imagery and Mapping Agency, "National Imagery and Mapping Agency Technical Report 8350.2, Third Edition", 2000, <http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf>.

10.2. Informative References

[BirthdayAttack]
"Birthday attack", <https://en.wikipedia.org/wiki/Birthday_attack.>.

[Common.Criteria]
"Common Criteria for Information Technology Security Evaluation", April 2017, <https://www.commoncriteriaportal.org/cc/>.

[ECMAScript]
"Ecma International, "ECMAScript Language Specification, 5.1 Edition", ECMA Standard 262", June 2011, <http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf>.

[FIDO.Registry]
The FIDO Alliance, "FIDO Registry of Predefined Values", December 2019, <https://fidoalliance.org/specs/common-specs/fido-registry-v2.1-ps-20191217.html>.

[FIPS-140]
National Institue of Standards, "Security Requirements for Cryptographic Modules", May 2001, <https://csrc.nist.gov/publications/detail/fips/140/2/final>.

[IDevID] "IEEE Standard, "IEEE 802.1AR Secure Device Identifier"", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

[IEEE.802-2001]
"IEEE Standard For Local And Metropolitan Area Networks Overview And Architecture", 2007, <https://webstore.ansi.org/standards/ieee/ieee8022001r2007>.

[IEEE.RA] "IEEE Registration Authority", <https://standards.ieee.org/products-services/regauth/index.html>.

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

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

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

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

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

[W3C.GeoLoc] Worldwide Web Consortium, "Geolocation API Specification 2nd Edition", January 2018, <https://www.w3.org/TR/geolocation-API/#coordinates_interface>.

Appendix A. Examples

A.1. Very Simple EAT

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

```
{
    / issuer /           1: "joe",
    / nonce /           10: h'948f8860d13a463e8e',
    / UEID /            11: h'0198f50a4ff6c05861c8860d13a638ea',
    / secure-boot /     15: true,
    / debug-disable /   16: 3, / permanent-disable /
    / timestamp (iat) /  6: 1(1526542894)
}
```

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

```
{
    / nonce /                   10: h'948f8860d13a463e8e',
    / UEID /                    11: h'0198f50a4ff6c05861c8860d13a638ea',
    / secure-boot /             15: true,
    / debug-disable /           16: 3, / permanent-disable  /
    / timestamp (iat) /          6: 1(1526542894),
    / security-level /          14: 3, / secure restricted OS /
    / submods / 20: {
        / first submod, an Android Application /
        "Android App Foo" :  {
            / security-level /  14: 1 / unrestricted /
        },

        / 2nd submod, A nested EAT from a secure element /
        "Secure Element Eat" :
            / an embedded EAT, bytes of which are not shown /
            h'420123',

        / 3rd submod, information about Linux Android /
        "Linux Android": {
            / security-level /  14: 1 / unrestricted /
        }
    }
}
```

Appendix B. UEID Design Rationale

B.1. Collision Probability

This calculation is to determine the probability of a collision of UEIDs given the total possible entity population and the number of entities in a particular entity management database.

Three different sized databases are considered. The number of devices per person roughly models non-personal devices such as traffic lights, devices in stores they shop in, facilities they work in and so on, even considering individual light bulbs. A device may have individually attested subsystems, for example parts of a car or a mobile phone. It is assumed that the largest database will have at most 10% of the world's population of devices. Note that databases that handle more than a trillion records exist today.

The trillion-record database size models an easy-to-imagine reality over the next decades. The quadrillion-record database is roughly at the limit of what is imaginable and should probably be accommodated. The 100 quadrillion datadbase is highly speculative perhaps involving nanorobots for every person, livestock animal and domesticated bird. It is included to round out the analysis.

Note that the items counted here certainly do not have IP address and are not individually connected to the network. They may be connected to internal buses, via serial links, Bluetooth and so on. This is not the same problem as sizing IP addresses.

People	Devices / Person	Subsystems / Device	Database Portion	Database Size
10 billion	100	10	10%	trillion (10^12)
10 billion	100,000	10	10%	quadrillion (10^15)
100 billion	1,000,000	10	10%	100 quadrillion (10^17)

This is conceptually similar to the Birthday Problem where m is the number of possible birthdays, always 365, and k is the number of people. It is also conceptually similar to the Birthday Attack where collisions of the output of hash functions are considered.

The proper formula for the collision calculation is

$$p = 1 - e^{-k^2/(2n)}$$

p Collision Probability
n Total possible population
k Actual population

However, for the very large values involved here, this formula requires floating point precision higher than commonly available in calculators and SW so this simple approximation is used. See [BirthdayAttack].

$$p = k^2 / 2n$$

For this calculation:

p Collision Probability
n Total population based on number of bits in UEID
k Population in a database

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	2 * 10^-15	8 * 10^-35	5 * 10^-55
quadrillion (10^15)	2 * 10^-09	8 * 10^-29	5 * 10^-49
100 quadrillion (10^17)	2 * 10^-05	8 * 10^-25	5 * 10^-45

Next, to calculate the probability of a collision occurring in one year's operation of a database, it is assumed that the database size is in a steady state and that 10% of the database changes per year. For example, a trillion record database would have 100 billion states per year. Each of those states has the above calculated probability of a collision.

This assumption is a worst-case since it assumes that each state of the database is completely independent from the previous state. In reality this is unlikely as state changes will be the addition or deletion of a few records.

The following tables gives the time interval until there is a probability of a collision based on there being one tenth the number of states per year as the number of records in the database.

t = 1 / ((k / 10) * p)

t Time until a collision
p Collision probability for UEID size
k Database size

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	60,000 years	10^24 years	10^44 years
quadrillion (10^15)	8 seconds	10^14 years	10^34 years
100 quadrillion (10^17)	8 microseconds	10^11 years	10^31 years

Clearly, 128 bits is enough for the near future thus the requirement that UEIDs be a minimum of 128 bits.

There is no requirement for 256 bits today as quadrillion-record databases are not expected in the near future and because this time-to-collision calculation is a very worst case. A future update of the standard may increase the requirement to 256 bits, so there is a requirement that implementations be able to receive 256-bit UEIDs.

B.2. No Use of UUID

A UEID is not a UUID [RFC4122] by conscious choice for the following reasons.

UUIDs are limited to 128 bits which may not be enough for some future use cases.

Today, cryptographic-quality random numbers are available from common CPUs and hardware. This hardware was introduced between 2010 and 2015. Operating systems and cryptographic libraries give access to this hardware. Consequently, there is little need for implementations to construct such random values from multiple sources on their own.

Version 4 UUIDs do allow for use of such cryptographic-quality random numbers, but do so by mapping into the overall UUID structure of time and clock values. This structure is of no value here yet adds complexity. It also slightly reduces the number of actual bits with entropy.

UUIDs seem to have been designed for scenarios where the implementor does not have full control over the environment and uniqueness has to be constructed from identifiers at hand. UEID takes the view that hardware, software and/or manufacturing process directly implement UEID in a simple and direct way. It takes the view that cryptographic quality random number generators are readily available as they are implemented in commonly used CPU hardware.

Appendix C. Changes from Previous Drafts

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

C.1. From draft-rats-eat-01

- Added UEID design rationale appendix

C.2. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but no new claims have been added.

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

C.3. From draft-ietf-rats-eat-01

- Clarifications and corrections for OEMID claim
- Minor spelling and other fixes
- Add the nonce claim, clarify jti claim

C.4. From draft-ietf-rats-eat-02

- Roll all EUIs back into one UEID type
- UEIDs can be one of three lengths, 128, 192 and 256.
- Added appendix justifying UEID design and size.
- Submods part now includes nested eat tokens so they can be named and there can be more tha one of them
- Lots of fixes to the CDDL
- Added security considerations

C.5. From draft-ietf-rats-eat-03

- Split boot_state into secure-boot and debug-disable claims
- Debug disable is an enumerated type rather than Booleans

C.6. From draft-ietf-rats-eat-04

- Change IMEI-based UEIDs to be encoded as a 14-byte string
- CDDL cleaned up some more
- CDDL allows for JWTs and UCCSs
- CWT format submodules are byte string wrapped
- Allows for JWT nested in CWT and vice versa
- Allows UCCS (unsigned CWTs) and JWT unsecured tokens
- Clarify tag usage when nesting tokens
- Add section on key inclusion
- Add hardware version claims
- Collected CDDL is now filled in. Other CDDL corrections.
- Rename debug-disable to debug-status; clarify that it is not extensible
- Security level claim is not extensible
- Improve specification of location claim and added a location privacy section
- Add intended use claim

C.7. From draft-ietf-rats-05

- CDDL format issues resolved
- Corrected reference to Location Privacy section

C.8. From draft-ietf-rats-06

- o Added boot-seed claim
- o Rework CBOR interoperability section
- o Added profiles claim and section

C.9. From draft-ietf-rats-07

- o Filled in IANA and other sections for possible preassignment of claim keys for well understood claims

C.10. From draft-ietf-rats-08

- o Change profile claim to be either a URL or an OID rather than a test string

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

H. Birkholz
M. Eckel
Fraunhofer SIT
W. Pan
Huawei Technologies
E. Voit
Cisco

26 April 2021

Reference Interaction Models for Remote Attestation Procedures
draft-ietf-rats-reference-interaction-models-02

Abstract

This document describes interaction models for remote attestation procedures (RATS). Three conveying mechanisms -- Challenge/Response, Uni-Directional, and Streaming Remote Attestation -- are illustrated and defined. Analogously, a general overview about the information elements typically used by corresponding conveyance protocols are highlighted.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Remote ATtestation procedureS (RATS, [I-D.ietf-rats-architecture]) are workflows composed of roles and interactions, in which Verifiers create Attestation Results about the trustworthiness of an Attester's system component characteristics. The Verifier's assessment in the form of Attestation Results is created based on Attestation Policies and Evidence -- trustable and tamper-evident Claims Sets about an Attester's system component characteristics -- generated by an Attester. The roles _Attester_ and _Verifier_, as well as the Conceptual Messages _Evidence_ and _Attestation Results_ are concepts defined by the RATS Architecture [I-D.ietf-rats-architecture]. This document defines interaction models that can be used in specific RATS-related solution documents. The primary focus of this document is the conveyance of attestation Evidence. The reference models defined can also be applied to the conveyance of other Conceptual Messages in RATS. Specific goals of this document are to:

1.) prevent inconsistencies in descriptions of interaction models in other documents (due to text cloning and evolution over time), and to 2.) enable to highlight an exact delta/divergence between the core set of characteristics captured here in this document and variants of these interaction models used in other specifications or solutions.

In summary, this document enables the specification and design of trustworthy and privacy preserving conveyance methods for attestation Evidence from an Attester to a Verifier. While the conveyance of other Conceptual Messages is out-of-scope the methods described can also be applied to the conveyance of, for example, Endorsements or Attestation Results.

2. Terminology

This document uses the following set of terms, roles, and concepts as defined in [I-D.ietf-rats-architecture]: Attester, Verifier, Relying Party, Conceptual Message, Evidence, Endorsement, Attestation Result, Appraisal Policy, Attesting Environment, Target Environment

A PKIX Certificate is an X.509v3 format certificate as specified by [RFC5280].

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

2.1. Disambiguation

The term "Remote Attestation" is a common expression and often associated or connoted with certain properties. The term "Remote" in this context does not necessarily refer to a remote entity in the scope of network topologies or the Internet. It rather refers to decoupled systems or entities that exchange the payload of the Conceptual Message type called Evidence [I-D.ietf-rats-architecture]. This conveyance can also be "Local", if the Verifier role is part of the same entity as the Attester role, e.g., separate system components of the same Composite Device (a single RATS entity). Even if an entity takes on two or more different roles, the functions they provide typically reside in isolated environments that are components of the same entity. Examples of such isolated environments include: a Trusted Execution Environment (TEE), Baseboard Management Controllers (BMCs), as well as other physical or logical protected/isolated/shielded Computing Environments (e.g. embedded Secure Elements (eSE) or Trusted Platform Modules (TPM)). Readers of this document should be familiar with the concept of Layered Attestation as described in Section 3.1 Two Types of Environments of an Attester in [I-D.ietf-rats-architecture] and the definition of Attestation as described in [I-D.ietf-rats-tpm-based-network-device-attest].

3. Scope and Intent

This document focuses on generic interaction models between Attesters and Verifiers in order to convey Evidence. Complementary procedures, functions, or services that are required for a complete semantic binding of the concepts defined in [I-D.ietf-rats-architecture] are out-of-scope of this document. Examples include: identity establishment, key distribution and enrollment, time synchronization, as well as certificate revocation.

Furthermore, any processes and duties that go beyond carrying out remote attestation procedures are out-of-scope.

For instance, using the results of a remote attestation procedure that are created by the Verifier, e.g., how to triggering remediation actions or recovery processes, as well as such remediation actions and recovery processes themselves, are also out-of-scope.

The interaction models illustrated in this document are intended to provide a stable basis and reference for other solutions documents inside or outside the IETF. Solution documents of any kind can reference the interaction models in order to avoid text clones and to avoid the danger of subtle discrepancies. Analogously, deviations from the generic model descriptions in this document can be illustrated in solutions documents to highlight distinct contributions.

4. Essential Requirements

In order to ensure appropriate conveyance of Evidence, there exist essential requirements which MUST be fulfilled:

Integrity: Information provided by an Attester MUST be integral. This may be achieved by means of a digital signature over Attestation Evidence. The signature may be symmetric, such as an HMAC, or asymmetric, such as ECDSA.

Authentication: The information provided by the Attester MUST be authentic. For that purpose, the Attester should authenticate itself to the Verifier. This may be an implicit authentication by means of a digital signature over the Attestation Evidence, which does not require additional protocol steps, or may be achieved by using a confidential channel by means of encryption.

4.1. Endorsement of Attesting Environments

Via its Attesting Environments, an Attester only generates Evidence about its Target Environments. After being appraised to be trustworthy, a Target Environment may become a new Attesting Environment in charge of generating Evidence for further Target Environments. [I-D.ietf-rats-architecture] explains this as Layered Attestation. Layered Attestation has to start with an initial Attesting Environment. In essence, there cannot be turtles all the way down [turtles]. At this rock bottom of Layered Attestation, the Attesting Environments are always called Roots of Trust (RoT). An Attester cannot generate Evidence about its own RoTs by design. As a consequence, a Verifier requires trustable statements about this subset of Attesting Environments from a different source than the Attester itself. The corresponding trustable statements are called Endorsements and originate from external, trustable entities that take on the role of an Endorser (e.g., supply chain entities).

5. Normative Prerequisites

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

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

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

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

Attestation Evidence Authenticity: Attestation Evidence MUST be authentic.

In order to provide proofs of authenticity, Attestation Evidence SHOULD be cryptographically associated with an identity document (e.g. an PKIX certificate or trusted key material, or a randomized DAA credential [DAA]), or SHOULD include a correct and unambiguous and stable reference to an accessible identity document.

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

The Attester MUST protect Claims with that Authentication Secret, thereby proving the authenticity of the Claims included in Evidence. The Authentication Secret MUST be established before RATS can take place.

Evidence Freshness: Evidence MUST include an indicator about its freshness that can be understood by a Verifier. Analogously, interaction models MUST support the conveyance of proofs of freshness in a way that is useful to Verifiers and their appraisal procedures.

Evidence Protection: Evidence MUST be a set of well-formatted and well-protected Claims that an Attester can create and convey to a Verifier in a tamper-evident manner.

6. Generic Information Elements

This section defines the information elements that are vital to all kinds interaction models. Varying from solution to solution, generic information elements can be either included in the scope of protocol messages (instantiating Conceptual Messages) or can be included in additional protocol parameters or payload. Ultimately, the following information elements are required by any kind of scalable remote attestation procedure using one or more of the interaction models provided.

Attester Identity ('attesterIdentity'): _mandatory_

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

Authentication Secret IDs ('authSecIDs'): _mandatory_

A statement representing an identifier list that MUST be associated with corresponding Authentication Secrets used to protect Claims included in Evidence.

Each Authentication Secret is uniquely associated with a distinguishable Attesting Environment. Consequently, an Authentication Secret ID also identifies an Attesting Environment.

Handle ('handle'): _mandatory_

A statement that is intended to uniquely distinguish received Evidence and/or determine the freshness of Evidence.

A Verifier can also use a Handle as an indicator for authenticity or attestation provenance, as only Attesters and Verifiers that are intended to exchange Evidence should have knowledge of the corresponding Handles. Examples include Nonces or signed timestamps.

Claims ('claims'): _mandatory_

Claims are assertions that represent characteristics of an Attester's Target Environment.

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

Event Logs ('eventLogs'): _optional_

Event Logs accompany Claims by providing event trails of security-critical events in a system. The primary purpose of Event Logs is to support Claim reproducibility by providing information on how Claims originated.

Reference Values ('refValues') _mandatory_

Reference Values as defined in [I-D.ietf-rats-architecture]. This specific type of Claims is used to appraise Claims incorporated in Evidence. For example, Reference Values MAY be Reference Integrity Measurements (RIM) or assertions that are implicitly trusted because they are signed by a trusted authority (see Endorsements in [I-D.ietf-rats-architecture]). Reference Values typically represent (trusted) Claim sets about an Attester's intended platform operational state.

Claim Selection ('claimSelection'): _optional_

A (sub-)set of Claims which can be created by an Attester.

Claim Selections act as filters to specify the exact set of Claims to be included in Evidence. In a remote attestation process, a Verifier sends a Claim Selection, among other elements, to an Attester. An Attester MAY decide whether or not to provide all requested Claims from a Claim Selection to the Verifier.

Collected Claims ('collectedClaims'): _mandatory_

Collected Claims represent a (sub-)set of Claims created by an Attester.

Collected Claims are gathered based on the Claims selected in the Claim Selection. If a Verifier does not provide a Claim Selection, then all available Claims on the Attester are part of the Collected Claims.

Evidence ('evidence'): _mandatory_

A set of Claims that consists of a list of Authentication Secret IDs that each identifies an Authentication Secret in a single Attesting Environment, the Attester Identity, Claims, and a Handle. Attestation Evidence MUST cryptographically bind all of these information elements. Evidence MUST be protected via an Authentication Secret. The Authentication Secret MUST be trusted by the Verifier as authoritative.

Attestation Result ('attestationResult'): _mandatory_

An Attestation Result is produced by the Verifier as the output of the appraisal of Evidence. Attestation Results include condensed assertions about integrity or other characteristics of the corresponding Attester that are processible by Relying Parties.

7. Interaction Models

The following subsections introduce and illustrate the interaction models:

1. Challenge/Response Remote Attestation

2. Uni-Directional Remote Attestation

3. Streaming Remote Attestation

Each section starts with a sequence diagram illustrating the interactions between Attester and Verifier. While the presented interaction models focus on the conveyance of Evidence, the intention of this document is in support of future work that applies the presented models to the conveyance of other Conceptual Messages, namely Attestation Results, Endorsements, Reference Values, or Appraisal Policies.

All interaction models have a strong focus on the use of a handle to incorporate a type of proof of freshness and to prevent replay attacks. The way these handles are processed is the most prominent difference between the three interaction models.

7.1. Challenge/Response Remote Attestation

```
.----------.                                                .----------.
| Attester |                                                | Verifier |
'----------'                                                '----------'
     |                                                            |
  generateClaims(attestingEnvironment)                            |
     | => claims, eventLogs                                       |
     |                                                            |
     | <-- requestAttestation(handle, authSecIDs, claimSelection) |
     |                                                            |
  collectClaims(claims, claimSelection)                           |
     | => collectedClaims                                         |
     |                                                            |
  generateEvidence(handle, authSecIDs, collectedClaims)           |
     | => evidence                                                |
     |                                                            |
     | evidence, eventLogs -------------------------------------> |
     |                                                            |
     |                appraiseEvidence(evidence, eventLogs, refValues)
     |                                       attestationResult <= |
     |                                                            |
```

The Attester boots up and thereby produces claims about its boot state and its operational state. Event Logs accompany the produced claims by providing an event trail of security-critical events in a system. Claims are produced by all attesting Environments of an Attester system.

The Challenge/Response remote attestation procedure is initiated by the Verifier by sending a remote attestation request to the Attester. A request includes a Handle, a list of Authentication Secret IDs, and a Claim Selection.

In the Challenge/Response model, the handle is composed of qualifying data in the form of a practically infeasible to guess nonce, such as a cryptographically strong random number. The Verifier-generated nonce is intended to guarantee Evidence freshness and to prevent replay attacks.

The list of Authentication Secret IDs selects the attestation keys with which the Attester is requested to sign the Attestation Evidence. Each selected key is uniquely associated with an Attesting Environment of the Attester. As a result, a single Authentication Secret ID identifies a single Attesting Environment. Correspondingly, a particular set of Evidence originating from a particular Attesting Environment in a composite device can be requested via multiple Authentication Secret IDs. Methods to acquire Authentication Secret IDs or mappings between Attesting Environments to Authentication Secret IDs are out-of-scope of this document.

The Attester collects Claims based on the Claim Selection. With the Claim Selection the Verifier defines the set of Claims it requires. Correspondingly, collected Claims can be a subset of the produced Claims. This could be all available Claims, depending on the Claim Selection. If the Claim Selection is omitted, then by default all Claims that are known and available on the Attester MUST be used to create corresponding Evidence. For example, when performing a boot integrity evaluation, a Verifier may only be requesting a particular subset of claims about the Attester, such as Evidence about BIOS/UEFI and firmware that the Attester booted up, and not include information about all currently running software.

With the Handle, the Authentication Secret IDs, and the collected Claims, the Attester produces signed Evidence. That is, it digitally signs the Handle and the collected Claims with a cryptographic secret identified by the Authentication Secret ID. This is done once per Attesting Environment which is identified by the particular Authentication Secret ID. The Attester communicates the signed Evidence as well as all accompanying Event Logs back to the Verifier.

While it is crucial that Claims, the Handle, and the Attester Identity information MUST be cryptographically bound to the signature of Evidence, they MAY be presented obfuscated, encrypted, or cryptographically blinded. For further reference see section Section 10.

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

7.2. Uni-Directional Remote Attestation

```
.----------.                       .--------------------.   .----------.
| Attester |                       | Handle Distributor |   | Verifier |
'----------'                       '--------------------'   '----------'
     |                                       |                    |
     |                                generateHandle()            |
     |                                       | => handle          |
     |                                       |                    |
     | <---------------------------- handle  | handle ----------> |
     |                                       |                    |
  generateClaims(attestingEnvironment)       x                    |
     | => claims, eventLogs                                       |
     |                                                            |
  collectClaims(claims, claimSelection)                           |
     | => collectedClaims                                         |
     |                                                            |
  generateEvidence(handle, authSecIDs, collectedClaims)           |
     | => evidence                                                |
     |                                                            |
     | evidence, eventLogs -------------------------------------> |
     |                                                            |
     |               appraiseEvidence(evidence, eventLogs, refValues)
     |                                       attestationResult <= |
     ~                                                            ~
     |                                                            |
*********[loop]********************************************************
*    |                                                            |    *
* generateClaims(attestingEnvironment)                            |    *
*    | => claimsDelta, eventLogsDelta                             |    *
*    |                                                            |    *
* collectClaims(claimsDelta, claimSelection)                      |    *
*    | => collectedClaimsDelta                                    |    *
*    |                                                            |    *
* generateEvidence(handle, authSecIDs, collectedClaimsDelta)      |    *
*    | => evidence                                                |    *
*    |                                                            |    *
*    | evidence, eventLogsDelta --------------------------------> |    *
*    |                                                            |    *
*    |          appraiseEvidence(evidence, eventLogsDelta, refValues) *
*    |                                       attestationResult <= |    *
*    |                                                            |    *
***********************************************************************
     |                                                            |
```

Uni-Directional Remote Attestation procedures can be initiated both by the Attester and by the Verifier. Initiation by the Attester can result in unsolicited pushes of Evidence to the Verifier. Initiation by the Verifier always results in solicited pushes to the Verifier.

The Uni-Directional model uses the same information elements as the Challenge/Response model. In the sequence diagram above, the Attester initiates the conveyance of Evidence (comparable with a RESTful POST operation or the emission of a beacon). While a request of Evidence from the Verifier would result in a sequence diagram more similar to the Challenge/Response model (comparable with a RESTful GET operation). The specific manner how Handles are created and used always remains as the distinguishing quality of this model.

In the Uni-Directional model, handles are composed of cryptographically signed trusted timestamps as shown in [I-D.birkholz-rats-tuda], potentially including other qualifying data. The Handles are created by an external 3rd entity -- the Handle Distributor -- which includes a trustworthy source of time, and takes on the role of a Time Stamping Authority (TSA, as initially defined in [RFC3161]). Timestamps created from local clocks (absolute clocks using a global timescale, as well as relative clocks, such as tick-counters) of Attesters and Verifiers MUST be cryptographically bound to fresh Handles received from the Handle Distributor. This binding provides a proof of synchronization that MUST be included in all produced Evidence. Correspondingly, conveyed Evidence in this model provides a proof that it was fresh at a certain point in time.

While periodically pushing Evidence to the Verifier, the Attester only needs to generate and convey evidence generated from Claim values that have changed and new Event Logs entries since the previous conveyance. This updates reflecting the differences are called "delta" in the sequence diagram above.

Effectively, the Uni-Directional model allows for a series of Evidence to be pushed to multiple Verifiers simultaneously. Methods to detect excessive time drift that would mandate a fresh Handle to be received by the Handle Distributor as well as timing of Handle distribution are out-of-scope of this document.

7.3. Streaming Remote Attestation

```
.----------.                                                .----------.
| Attester |                                                | Verifier |
'----------'                                                '----------'
     |                                                            |
  generateClaims(attestingEnvironment)                            |
     | => claims, eventLogs                                       |
     |                                                            |
     | <----------- subscribe(handle, authSecIDs, claimSelection) |
     | subscriptionResult --------------------------------------> |
     |                                                            |
  collectClaims(claims, claimSelection)                           |
     | => collectedClaims                                         |
     |                                                            |
  generateEvidence(handle, authSecIDs, collectedClaims)           |
     | => evidence                                                |
     |                                                            |
     | evidence, eventLogs -------------------------------------> |
     |                                                            |
     |                appraiseEvidence(evidence, eventLogs, refValues)
     |                                       attestationResult <= |
     ~                                                            ~

     |                                                            |
*********[loop]*********************************************************
*    |                                                            |    *
* generateClaims(attestingEnvironment)                            |    *
*    | => claimsDelta, eventLogsDelta                             |    *
*    |                                                            |    *
* collectClaims(claimsDelta, claimSelection)                      |    *
*    | => collectedClaimsDelta                                    |    *
*    |                                                            |    *
* generateEvidence(handle, authSecIDs, collectedClaimsDelta)      |    *
*    | => evidence                                                |    *
*    |                                                            |    *
*    | evidence, eventLogsDelta --------------------------------> |    *
*    |                                                            |    *
*    |           appraiseEvidence(evidence, eventLogsDelta, refValues) *
*    |                                       attestationResult <= |    *
*    |                                                            |    *
************************************************************************
     |                                                            |
```

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

8. Additional Application-Specific Requirements

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

8.1. 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 which provides confidentiality protection. In private networks, such as carrier management networks, it must be evaluated whether or not the transport medium is considered confidential.

8.2. 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.

8.3. Hardware-Enforcement/Support

Depending on given usage scenarios, hardware support for secure storage of cryptographic keys, crypto accelerators, as well as protected or isolated execution environments can be mandatory requirements. Well-known technologies in support of these requirements are roots of trusts, such as Hardware Security Modules (HSM), Physically Unclonable Functions (PUFs), Shielded Secrets, or Trusted Executions Environments (TEEs).

9. Implementation Status

Note to RFC Editor: Please remove this section as well as references to [BCP205] before AUTH48.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [BCP205]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [BCP205], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. Implementer

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

9.2. Implementation Name

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

9.3. Implementation URL

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

9.4. Maturity

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

9.5. Coverage and Version Compatibility

The current version ('1bcb469') implements a challenge/response interaction model and is aligned with the exemplary specification of the CoAP FETCH bodies defined in Section Appendix A of this document.

9.6. License

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

9.7. Implementation Dependencies

The implementation requires the use of the official Trusted Computing Group (TCG) open-source Trusted Software Stack (TSS) for the Trusted Platform Module (TPM) 2.0. The corresponding code and data is also maintained on GitHub and the project resources can be located via: https://github.com/tpm2-software/tpm2-tss/

The implementation uses the Constrained Application Protocol [RFC7252] (http://coap.technology/) and the Concise Binary Object Representation [RFC7049] (https://cbor.io/).

9.8. Contact

Michael Eckel (michael.eckel@sit.fraunhofer.de)

10. Security and Privacy Considerations

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

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

11. Acknowledgments

Olaf Bergmann, Michael Richardson, and Ned Smith

12. References

12.1. Normative References

[BCP205] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

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

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/info/rfc3161>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

12.2. Informative References

[DAA] Brickell, E., Camenisch, J., and L. Chen, "Direct Anonymous Attestation", page 132-145, ACM Proceedings of the 11rd ACM conference on Computer and Communications Security, 2004.

[I-D.birkholz-rats-tuda] Fuchs, A., Birkholz, H., McDonald, I. E., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-04, 13 January 2021, <https://www.ietf.org/archive/id/draft-birkholz-rats-tuda-04.txt>.

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-12, 23 April 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-12.txt>.

[I-D.ietf-rats-tpm-based-network-device-attest]
Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", Work in Progress, Internet-Draft, draft-ietf-rats-tpm-based-network-device-attest-06, 7 December 2020, <https://www.ietf.org/archive/id/draft-ietf-rats-tpm-based-network-device-attest-06.txt>.

[turtles] Rudnicki, R., "Turtles All the Way Down: Foundation, Edifice, and Ruin in Faulkner and McCarthy", DOI 10.1353/fau.2010.0002, The Faulkner Journal 25.2, 2010, <https://doi.org/10.1353/fau.2010.0002>.

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

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

In this example, evidence is created via the root-of-trust for reporting primitive operation "quote" that is provided by a TPM 2.0.

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

   CoAP-FETCH-Body = [ hello: bool, ; if true, the AK-Cert is conveyed
                       nonce: bytes,
                       pcr-selection: [ + [ tcg-hash-alg-id: uint .size 2, ; TPM2
_ALG_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
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Michael Eckel
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: michael.eckel@sit.fraunhofer.de

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

RATS Working Group G. Fedorkow, Ed.
Internet-Draft Juniper Networks, Inc.
Intended status: Informational E. Voit
Expires: June 10, 2021 Cisco Systems, Inc.
J. Fitzgerald-McKay
National Security Agency
December 07, 2020

TPM-based Network Device Remote Integrity Verification
draft-ietf-rats-tpm-based-network-device-attest-06

Abstract

This document describes a workflow for remote attestation of the integrity of firmware and software installed on network devices that contain Trusted Platform Modules [TPM1.2], [TPM2.0], as defined by the Trusted Computing Group (TCG).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

There are many aspects to consider in fielding a trusted computing device, from operating systems to applications. Mechanisms to prove that a device installed at a customer's site is authentic (i.e., not counterfeit) and has been configured with authorized software, all as part of a trusted supply chain, are just a few of the many aspects which need to be considered concurrently to have confidence that a device is truly trustworthy.

A generic architecture for remote attestation has been defined in [I-D.ietf-rats-architecture]. Additionally, the use cases for remotely attesting networking devices are discussed within Section 6 of [I-D.richardson-rats-usecases]. However, these documents do not provide sufficient guidance for network equipment vendors and operators to design, build, and deploy interoperable devices.

The intent of this document is to provide such guidance. It does this by outlining the Remote Integrity Verification (RIV) problem, and then identifies elements that are necessary to get the complete, scalable attestation procedure working with commercial networking products such as routers, switches and firewalls. An underlying assumption will be the availability within the device of a Trusted Platform Module [TPM1.2], [TPM2.0] compliant cryptoprocessor to enable the trustworthy remote assessment of the device's software and hardware.

1.1. Terminology

A number of terms are reused from [I-D.ietf-rats-architecture]. These include: Appraisal Policy for Evidence, Attestation Result, Attester, Evidence, Reference Value, Relying Party, Verifier, and Verifier Owner.

Additionally, this document defines the following term:

Attestation: the process of generating, conveying and appraising claims, backed by evidence, about device trustworthiness characteristics, including supply chain trust, identity, device provenance, software configuration, device composition, compliance to test suites, functional and assurance evaluations, etc.

The goal of attestation is simply to assure an administrator that the device configuration and software that was launched when the device was last started is authentic and untampered-with.

Within the Trusted Computing Group (TCG) context, attestation is the process by which an independent Verifier can obtain cryptographic proof as to the identity of the device in question, and evidence of the integrity of software loaded on that device when it started up, and then verify that what's there matches the intended configuration. For network equipment, a Verifier capability can be embedded in a Network Management Station (NMS), a posture collection server, or other network analytics tool (such as a software asset management solution, or a threat detection and mitigation tool, etc.). While informally referred to as attestation, this document focuses on a subset defined here as Remote Integrity Verification (RIV). RIV takes a network equipment centric perspective that includes a set of protocols and procedures for determining whether a particular device was launched with authentic software, starting from Roots of Trust. While there are many ways to accomplish attestation, RIV sets out a specific set of protocols and tools that work in environments commonly found in network equipment. RIV does not cover other device characteristics that could be attested (e.g., geographic location, connectivity; see [I-D.richardson-rats-usecases]), although it does provide evidence of a secure infrastructure to increase the level of trust in other device characteristics attested by other means (e.g., by Entity Attestation Tokens [I-D.ietf-rats-eat]).

In line with [I-D.ietf-rats-architecture] definitions, this document uses the term Endorser to refer to the role that signs identity and attestation certificates used by the Attester, while Reference Values are signed by a Reference Value Provider. Typically, the manufacturer of an embedded device would be accepted as both the Endorser and Reference Value Provider, although the choice is ultimately up to the Verifier Owner.

1.2. Document Organization

The remainder of this document is organized into several sections:

- The remainder of this section covers goals and requirements, plus a top-level description of RIV.
- The Solution Overview section outlines how Remote Integrity Verification works.
- The Standards Components section links components of RIV to normative standards.
- Privacy and Security shows how specific features of RIV contribute to the trustworthiness of the Attestation Result.
- Supporting material is in an appendix at the end.

1.3. Goals

Network operators benefit from a trustworthy attestation mechanism that provides assurance that their network comprises authentic equipment, and has loaded software free of known vulnerabilities and unauthorized tampering. In line with the overall goal of assuring integrity, attestation can be used to assist in asset management, vulnerability and compliance assessment, plus configuration management.

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

- Provable Device Identity - This specification requires that an Attester (i.e., the attesting device) includes a cryptographic identifier unique to each device. Effectively this means that the TPM must be so provisioned during the manufacturing cycle.

- Software Inventory - A key goal is to identify the software release(s) installed on the Attester, and to provide evidence that the software stored within hasn't been altered without authorization.

- Verifiability - Verification of software and configuration of the device shows that the software that was authorized for installation by the administrator has actually been launched.

In addition, RIV is designed to operate either in a centralized environment, such as with a central authority that manages and configures a number of network devices, or 'peer-to-peer', where network devices independently verify one another to establish a trust relationship. (See Section 3.3 below, and also [I-D.voit-rats-trusted-path-routing])

1.4. Description of Remote Integrity Verification (RIV)

Attestation requires two interlocking mechanisms between the Attester network device and the Verifier:

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

o Software Measurement is the mechanism that reports the state of mutable software components on the device, and can assure network managers that they have known, authentic software configured to run in their network.

Using these two interlocking mechanisms, RIV is a component in a chain of procedures that can assure a network operator that the equipment in their network can be reliably identified, and that authentic software of a known version is installed on each device. Equipment in the network includes devices that make up the network itself, such as routers, switches and firewalls.

Software used to boot a device can be described as recording a chain of measurements, anchored at the start by a Root of Trust for Measurement (see Section 9.2), each measuring the next stage, that normally ends when the system software is loaded. A measurement signifies the identity, integrity and version of each software component registered with an Attester's TPM [TPM1.2], [TPM2.0], so that a subsequent verification stage can determine if the software installed is authentic, up-to-date, and free of tampering.

RIV includes several major processes:

1. Generation of Evidence is the process whereby an Attester generates cryptographic proof (Evidence) of claims about device properties. In particular, the device identity and its software configuration are both of critical importance.

2. Device Identification refers to the mechanism assuring the Relying Party (ultimately, a network administrator) of the identity of devices that make up their network, and that their manufacturers are known.

3. Conveyance of Evidence reliably transports the collected Evidence from Attester to a Verifier to allow a management station to perform a meaningful appraisal in Step 4. The transport is typically carried out via a management network. The channel must provide integrity and authenticity, and, in some use cases, may also require confidentiality.

4. Finally, Appraisal of Evidence occurs. This is the process of verifying the Evidence received by a Verifier from the Attester, and using an Appraisal Policy to develop an Attestation Result, used to inform decision making. In practice, this means comparing the Attester's measurements reported as Evidence with the device configuration expected by the Verifier. Subsequently the Appraisal Policy for Evidence might match Evidence found

against Reference Values (aka Golden Measurements), which represent the intended configured state of the connected device.

All implementations supporting this RIV specification require the support of the following three technologies:

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

2. Platform Attestation provides evidence of configuration of software elements present in the device. This form of attestation can be implemented with TPM Platform Configuration Registers (PCRs), Quote and Log mechanisms, which provide cryptographically authenticated evidence to report what software was started on the device through the boot cycle. Successful attestation requires an unbroken chain from a boot-time root of trust through all layers of software needed to bring the device to an operational state, in which each stage measures components of the next stage, updates the attestation log, and extends hashes into a PCR. The TPM can then report the hashes of all the measured hashes as signed evidence called a Quote (see Section 9.1 for an overview of TPM operation, or [TPM1.2] and [TPM2.0] for many more details).

3. Signed Reference Values (aka Reference Integrity Measurements) must be conveyed from the Reference Value Provider (the entity accepted as the software authority, often the manufacturer for embedded systems) to the Verifier.

1.5. Solution Requirements

Remote Integrity Verification must address the "Lying Endpoint" problem, in which malicious software on an endpoint may subvert the intended function, and also prevent the endpoint from reporting its compromised status. (See Section 5 for further Security Considerations.)

RIV attestation is designed to be simple to deploy at scale. RIV should work "out of the box" as far as possible, that is, with the fewest possible provisioning steps or configuration databases needed at the end-user's site. Network equipment is often required to "self-configure", to reliably reach out without manual intervention to prove its identity and operating posture, then download its own configuration, a process which precludes pre-installation configuration. See [RFC8572] for an example of Secure Zero Touch Provisioning.

1.6. Scope

The need for assurance of software integrity, addressed by Remote Attestation, is a very general problem that could apply to most network-connected computing devices. However, this document includes several assumptions that limit the scope to network equipment (e.g., routers, switches and firewalls):

- o This solution is for use in non-privacy-preserving applications (for example, networking, Industrial IoT), avoiding the need for a Privacy Certificate Authority for attestation keys [AK-Enrollment] or TCG Platform Certificates [Platform-Certificates].

- o This document assumes network protocols that are common in network equipment such as YANG [RFC7950] and NETCONF [RFC6241], but not generally used in other applications.

- o The approach outlined in this document mandates the use of a compliant TPM [TPM1.2], [TPM2.0].

1.6.1. Out of Scope

- o Run-Time Attestation: The Linux Integrity Measurement Architecture attests each process launched after a device is started (and is in scope for RIV), but continuous run-time attestation of Linux or other multi-threaded operating system processes after they've started considerably expands the scope of the problem. Many researchers are working on that problem, but this document defers the problem of continuous, in-memory run-time attestation.

- o Multi-Vendor Embedded Systems: Additional coordination would be needed for devices that themselves comprise hardware and software from multiple vendors, integrated by the end user. Although out of scope for this document, these issues are accommodated in [I-D.ietf-rats-architecture].

- o Processor Sleep Modes: Network equipment typically does not "sleep", so sleep and hibernate modes are not considered. Although out of scope for RIV, Trusted Computing Group specifications do encompass sleep and hibernate states.

- Virtualization and Containerization: In a non-virtualized system, the host OS is responsible for measuring each User Space file or process, but that's the end of the boot process. For virtualized systems, the host OS must verify the hypervisor, which then manages its own chain of trust through the virtual machine. Virtualization and containerization technologies are increasingly used in network equipment, but are not considered in this document.

2. Solution Overview

2.1. RIV Software Configuration Attestation using TPM

RIV Attestation is a process which can be used to determine the identity of software running on a specifically-identified device. Remote Attestation is broken into two phases, shown in Figure 1:

- During system startup, each distinct software object is "measured" by the Attester. The object's identity, hash (i.e., cryptographic digest) and version information are recorded in a log. Hashes are also extended, or cryptographically folded, into the TPM, in a way that can be used to validate the log entries. The measurement process generally follows the layered chain-of-trust model used in Measured Boot, where each stage of the system measures the next one, and extends its measurement into the TPM, before launching it. See [I-D.ietf-rats-architecture], section "Layered Attestation Environments," for an architectural definition of this model.

- Once the device is running and has operational network connectivity, a separate, trusted Verifier will interrogate the network device to retrieve the logs and a copy of the digests collected by hashing each software object, signed by an attestation private key secured by, but never released by, the TPM. The YANG model described in [I-D.ietf-rats-yang-tpm-charra] facilitates this operation.

The result is that the Verifier can verify the device's identity by checking the Subject Field and signature of certificate containing the TPM's attestation public key, and can validate the software that was launched by verifying the correctness of the logs by comparing with the signed digests from the TPM, and comparing digests in the log with Reference Values.

It should be noted that attestation and identity are inextricably linked; signed Evidence that a particular version of software was loaded is of little value without cryptographic proof of the identity of the Attester producing the Evidence.

```
       +-------------------------------------------------------+
       | +--------+    +--------+   +--------+    +---------+  |
       | | BIOS   |--->| Loader |-->| Kernel |--->|Userland |  |
       | +--------+    +--------+   +--------+    +---------+  |
       |     |            |           |                        |
       |     |            |           |                        |
       |     +------------+-----------+-+                      |
       |                        Step 1  |                      |
       |                                V                      |
       |                            +--------+                 |
       |                            |  TPM   |                 |
       |                            +--------+                 |
       |   Router                       |                      |
       +--------------------------------|----------------------+
                                        |
                                        |  Step 2
                                        |    +-----------+
                                        +--->| Verifier  |
                                             +-----------+

       Reset---------------flow-of-time-during-boot--...------->
```

Figure 1: Layered RIV Attestation Model

In Step 1, measurements are "extended", or hashed, into the TPM as processes start, with the result that the PCR ends up containing a hash of all the measured hashes. In Step 2, signed PCR digests are retrieved from the TPM for off-box analysis after the system is operational.

2.1.1. What Does RIV Attest?

TPM attestation is strongly focused on Platform Configuration Registers (PCRs), but those registers are only vehicles for certifying accompanying Evidence, conveyed in log entries. It is the hashes in log entries that are extended into PCRs, where the final PCR values can be retrieved in the form of a structure called a Quote, signed by an Attestation key known only to the TPM. The use of multiple PCRs serves only to provide some independence between different classes of object, so that one class of objects can be updated without changing the extended hash for other classes. Although PCRs can be used for any purpose, this section outlines the objects within the scope of this document which may be extended into the TPM.

In general, assignment of measurements to PCRs is a policy choice made by the device manufacturer, selected to independently attest three classes of object:

- Code, (i.e., instructions) to be executed by a CPU.

- Configuration - Many devices offer numerous options controlled by non-volatile configuration variables which can impact the device's security posture. These settings may have vendor defaults, but often can be changed by administrators, who may want to verify via attestation that the operational state of the settings match their intended state.

- Credentials - Administrators may wish to verify via attestation that public keys (and other credentials) outside the Root of Trust have not been subject to unauthorized tampering. (By definition, keys protecting the root of trust can't be verified independently.)

The TCG PC Client Platform Firmware Profile Specification [PC-Client-BIOS-TPM-2.0] gives considerable detail on what is to be measured during the boot phase of platform startup using a UEFI BIOS (www.uefi.org), but the goal is simply to measure every bit of code executed in the process of starting the device, along with any configuration information related to security posture, leaving no gap for unmeasured code to remain undetected, potentially subverting the chain.

For devices using a UEFI BIOS, [PC-Client-BIOS-TPM-2.0] gives detailed normative requirements for PCR usage. For other platform architectures, the table in Figure 2 gives non-normative guidance for PCR assignment that generalizes the specific details of [PC-Client-BIOS-TPM-2.0].

By convention, most PCRs are assigned in pairs, which the even-numbered PCR used to measure executable code, and the odd-numbered PCR used to measure whatever data and configuration are associated with that code. It is important to note that each PCR may contain results from dozens (or even thousands) of individual measurements.

Function	Assigned PCR # Code	Assigned PCR # Configuration
Firmware Static Root of Trust, (i.e., initial boot firmware and drivers)	0	1
Drivers and initialization for optional or add-in devices	2	3
OS Loader code and configuration, (i.e., the code launched by firmware) to load an operating system kernel. These PCRs record each boot attempt, and an identifier for where the loader was found	4	5
Vendor Specific Measurements during boot	6	6
Secure Boot Policy. This PCR records keys and configuration used to validate the OS loader		7
Measurements made by the OS Loader (e.g GRUB2 for Linux)	8	9
Measurements made by OS (e.g., Linux IMA)	10	10

Figure 2: Attested Objects

2.1.2. Notes on PCR Allocations

It is important to recognize that PCR[0] is critical. The first measurement into PCR[0] is taken by the Root of Trust for Measurement, code which, by definition, cannot be verified by measurement. This measurement establishes the chain of trust for all subsequent measurements. If the PCR[0] measurement cannot be trusted, the validity of the entire chain is put into question.

Distinctions Between PCR[0], PCR[2], PCR[4] and PCR[8] are summarized below:

- PCR[0] typically represents a consistent view of rarely-changed Host Platform boot components, allowing Attestation policies to be defined using the less changeable components of the transitive trust chain. This PCR typically provides a consistent view of the platform regardless of user selected options.

- PCR[2] is intended to represent a "user configurable" environment where the user has the ability to alter the components that are measured into PCR[2]. This is typically done by adding adapter cards, etc., into user-accessible PCI or other slots. In UEFI systems these devices may be configured by Option ROMs measured into PCR[2] and executed by the BIOS.

- PCR[4] is intended to represent the software that manages the transition between the platform's Pre-Operating System start and the state of a system with the Operating System present. This PCR, along with PCR[5], identifies the initial operating system loader (e.g., GRUB for Linux).

- PCR[8] is used by the OS loader (e.g. GRUB) to record measurements of the various components of the operating system.

Although the TCG PC Client document specifies the use of the first eight PCRs very carefully to ensure interoperability among multiple UEFI BIOS vendors, it should be noted that embedded software vendors may have considerably more flexibility. Verifiers typically need to know which log entries are consequential and which are not (possibly controlled by local policies) but the Verifier may not need to know what each log entry means or why it was assigned to a particular PCR. Designers must recognize that some PCRs may cover log entries that a particular Verifier considers critical and other log entries that are not considered important, so differing PCR values may not on their own constitute a check for authenticity. For example, in a UEFI system, some administrators may consider booting an image from a removable drive, something recorded in a PCR, to be a security violation, while others might consider that operation an authorized recovery procedure.

Designers may allocate particular events to specific PCRs in order to achieve a particular objective with local attestation, (e.g., allowing a procedure to execute, or releasing a paricular decryption key, only if a given PCR is in a given state). It may also be important to designers to consider whether streaming notification of PCR updates is required (see [I-D.birkholz-rats-network-device-subscription]). Specific log entries can only be validated if the Verifier receives every log entry affecting the relevant PCR, so (for example) a designer might want to separate rare, high-value events such as configuration changes, from high-volume, routine measurements such as IMA [IMA] logs.

2.2. RIV Keying

RIV attestation relies on two credentials:

- o An identity key pair and matching certificate is required to certify the identity of the Attester itself. RIV specifies the use of an IEEE 802.1AR Device Identity (DevID) [IEEE-802-1AR], signed by the device manufacturer, containing the device serial number.

- o An Attestation key pair and matching certificate is required to sign the Quote generated by the TPM to report evidence of software configuration.

In TPM application, both the Attestation private key and the DevID private key MUST be protected by the TPM. Depending on other TPM configuration procedures, the two keys are likely be different; some of the considerations are outlined in TCG "TPM 2.0 Keys for Device Identity and Attestation" [Platform-DevID-TPM-2.0].

The TCG TPM 2.0 Keys document [Platform-DevID-TPM-2.0] specifies further conventions for these keys:

- o When separate Identity and Attestation keys are used, the Attestation Key (AK) and its X.509 certificate should parallel the DevID, with the same device ID information as the DevID certificate (that is, the same Subject Name and Subject Alt Name, even though the key pairs are different). This allows a quote from the device, signed by an AK, to be linked directly to the device that provided it, by examining the corresponding AK certificate. If the Subject and Subject Alt Names in the AK certificate don’t match the corresponding DevID certifcate, or they’re signed by differing authorities the Verifier may signal the detection of an Asokan-style man-in-the-middle attack (see Section 5.2).

- o Network devices that are expected to use secure zero touch provisioning as specified in [RFC8572]) MUST be shipped by the manufacturer with pre-provisioned keys (Initial DevID and Initial AK, called IDevID and IAK). IDevID and IAK certificates MUST both be signed by the Endorser (typically the device manufacturer). Inclusion of an IDevID and IAK by a vendor does not preclude a mechanism whereby an administrator can define Local Identity and Attestation Keys (LDevID and LAK) if desired.

2.3. RIV Information Flow

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

1. The Attester, the device which the network operator wants to examine.

2. A Verifier (which might be a network management station) somewhere separate from the Device that will retrieve the signed evidence and measurement logs, and analyze them to pass judgment on the security posture of the device.

3. A Relying Party, which can act on Attestation Results. Interaction between the Relying Party and the Verifier is considered out of scope for RIV.

4. Signed Reference Integrity Manifests (RIMs), containing Reference Values, can either be created by the device manufacturer and shipped along with the device as part of its software image, or alternatively, could be obtained several other ways (direct to the Verifier from the manufacturer, from a third party, from the owner's observation of what's thought to be a "known good system", etc.). Retrieving RIMs from the device itself allows attestation to be done in systems that may not have access to the public internet, or by other devices that are not management stations per se (e.g., a peer device; see Section 3.1.3). If Reference Values are obtained from multiple sources, the Verifier may need to evaluate the relative level of trust to be placed in each source in case of a discrepancy.

These components are illustrated in Figure 3.

A more-detailed taxonomy of terms is given in [I-D.ietf-rats-architecture]

```
+----------------+        +-------------+        +---------+--------+
|Reference Value |        | Attester    | Step 1 | Verifier|        |
|Provider        |        | (Device     |<-------| (Network| Relying|
|(Device         |        | under       |------->| Mngmt   | Party  |
|Manufacturer    |        | attestation)| Step 2 | Station)|        |
|or other        |        |             |        |         |        |
|authority)      |        |             |        |         |        |
+----------------+        +-------------+        +---------+--------+
       |                                             /\
       |                  Step 0                      |
       -----------------------------------------------
```

Figure 3: RIV Reference Configuration for Network Equipment

- In Step 0, The Reference Value Provider (the device manufacturer or other authority) makes one or more Reference Integrity Manifests (RIMs), corresponding to the software image expected to be found on the device, signed by the Reference Value Provider, available to the Verifier (see Section 3.1.3 for "in-band" and "out of band" ways to make this happen).

- In Step 1, the Verifier (Network Management Station), on behalf of a Relying Party, requests Identity, Measurement Values, and possibly RIMs, from the Attester.

- In Step 2, the Attester responds to the request by providing a DevID, quotes (measured values, signed by the Attester), and optionally RIMs.

To achieve interoperability, the following standards components SHOULD be used:

1. TPM Keys MUST be configured according to [Platform-DevID-TPM-2.0], or [Platform-ID-TPM-1.2].

2. For devices using UEFI and Linux, measurements of firmware and bootable modules SHOULD be taken according to TCG PC Client [PC-Client-BIOS-TPM-1.2] or [PC-Client-BIOS-TPM-2.0], and Linux IMA [IMA]

3. Device Identity MUST be managed as specified in IEEE 802.1AR Device Identity certificates [IEEE-802-1AR], with keys protected by TPMs.

4. Attestation logs SHOULD be formatted according to the Canonical Event Log format [Canonical-Event-Log], although other

specialized formats may be used. UEFI-based systems MAY use the TCG UEFI BIOS event log [EFI-TPM].

5. Quotes are retrieved from the TPM according to TCG TAP Information Model [TAP]. While the TAP IM gives a protocol-independent description of the data elements involved, it's important to note that quotes from the TPM are signed inside the TPM, so MUST be retrieved in a way that does not invalidate the signature, as specified in [I-D.ietf-rats-yang-tpm-charra], to preserve the trust model. (See Section 5 Security Considerations).

6. Reference Values SHOULD be encoded as SWID or CoSWID tags, as defined in the TCG RIM document [RIM], compatible with NIST IR 8060 [NIST-IR-8060] and the IETF CoSWID draft [I-D.ietf-sacm-coswid]. See Section 2.4.1.

2.4. RIV Simplifying Assumptions

This document makes the following simplifying assumptions to reduce complexity:

- The product to be attested MUST be shipped with an IEEE 802.1AR Device Identity and an Initial Attestation Key (IAK) with certificate. The IAK certificate MUST contain the same identity information as the DevID (specifically, the same Subject Name and Subject Alt Name, signed by the manufacturer), but it's a type of key that can be used to sign a TPM Quote, but not other objects (i.e., it's marked as a TCG "Restricted" key; this convention is described in "TPM 2.0 Keys for Device Identity and Attestation" [Platform-DevID-TPM-2.0]). For network equipment, which is generally non-privacy-sensitive, shipping a device with both an IDevID and an IAK already provisioned substantially simplifies initial startup. Privacy-sensitive applications may use the TCG Platform Certificate or other procedures to install identity credentials into the device after manufacture.

- The product MUST be equipped with a Root of Trust for Measurement (RTM), Root of Trust for Storage and Root of Trust for Reporting (as defined in [SP800-155]) that are capable of conforming to TCG Trusted Attestation Protocol (TAP) Information Model [TAP].

- The authorized software supplier MUST make available Reference Values in the form of signed SWID or CoSWID tags [I-D.ietf-sacm-coswid], [SWID], as described in TCG Reference Integrity Measurement Manifest Information Model [RIM].

2.4.1. Reference Integrity Manifests (RIMs)

[I-D.ietf-rats-yang-tpm-charra] focuses on collecting and transmitting evidence in the form of PCR measurements and attestation logs. But the critical part of the process is enabling the Verifier to decide whether the measurements are "the right ones" or not.

While it must be up to network administrators to decide what they want on their networks, the software supplier should supply the Reference Values, in signed Reference Integrity Manifests, that may be used by a Verifier to determine if evidence shows known good, known bad or unknown software configurations.

In general, there are two kinds of reference measurements:

1. Measurements of early system startup (e.g., BIOS, boot loader, OS kernel) are essentially single-threaded, and executed exactly once, in a known sequence, before any results could be reported. In this case, while the method for computing the hash and extending relevant PCRs may be complicated, the net result is that the software (more likely, firmware) vendor will have one known good PCR value that "should" be present in the relevant PCRs after the box has booted. In this case, the signed reference measurement could simply list the expected hashes for the given version. However, a RIM that contains the intermediate hashes can be useful in debugging cases where the expected final hash is not the one reported.

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

In both cases, the expected values can be expressed as signed SWID or CoSWID tags, but the SWID structure in the second case is somewhat more complex, as reconstruction of the extended hash in a PCR may involve thousands of files and other objects.

TCG has published an information model defining elements of Reference Integrity Manifests under the title TCG Reference Integrity Manifest Information Model [RIM]. This information model outlines how SWID tags should be structured to allow attestation, and defines "bundles" of SWID tags that may be needed to describe a complete software release. The RIM contains metadata relating to the software release it belongs to, plus hashes for each individual file or other object that could be attested.

TCG has also published the PC Client Reference Integrity Measurement specification [PC-Client-RIM], which focuses on a SWID-compatible format suitable for expressing expected measurement values in the specific case of a UEFI-compatible BIOS, where the SWID focus on files and file systems is not a direct fit. While the PC Client RIM is not directly applicable to network equipment, many vendors do use a conventional UEFI BIOS to launch their network OS.

2.4.2. Attestation Logs

Quotes from a TPM can provide evidence of the state of a device up to the time the evidence was recorded, but to make sense of the quote in most cases an event log that identifies which software modules contributed which values to the quote during startup MUST also be provided. The log MUST contain enough information to demonstrate its integrity by allowing exact reconstruction of the digest conveyed in the signed quote (that is, calculating the hash of all the hashes in the log should produce the same values as contained in the PCRs; if they don't match, the log may have been tampered with. See Section 9.1).

There are multiple event log formats which may be supported as viable formats of Evidence between the Attester and Verifier:

- IMA Event log file exports [IMA]
- TCG UEFI BIOS event log (TCG EFI Platform Specification for TPM Family 1.1 or 1.2, Section 7) [EFI-TPM]
- TCG Canonical Event Log [Canonical-Event-Log]

Attesters which use UEFI BIOS and Linux SHOULD use TCG Canonical Event Log [Canonical-Event-Log] and TCG UEFI BIOS event log [EFI-TPM], although the CHARRA YANG model [I-D.ietf-rats-yang-tpm-charra] has no dependence on the format of the log.

3. Standards Components

3.1. Prerequisites for RIV

The Reference Interaction Model for Challenge-Response-based Remote Attestation ([I-D.birkholz-rats-reference-interaction-model]) is based on the standard roles defined in [I-D.ietf-rats-architecture]. However additional prerequisites have been established to allow for interoperable RIV use case implementations. These prerequisites are intended to provide sufficient context information so that the

Verifier can acquire and evaluate measurements collected by the Attester.

3.1.1. Unique Device Identity

A secure Device Identity (DevID) in the form of an IEEE 802.1AR DevID certificate [IEEE-802-1AR] MUST be provisioned in the Attester's TPMs.

3.1.2. Keys

The Attestation Key (AK) and certificate MUST also be provisioned on the Attester according to [Platform-DevID-TPM-2.0], [PC-Client-BIOS-TPM-1.2], or [Platform-ID-TPM-1.2].

The Attester's TPM Keys MUST be associated with the DevID on the Verifier (see [Platform-DevID-TPM-2.0] and Section 5 Security Considerations, below).

3.1.3. Appraisal Policy for Evidence

The Verifier MUST obtain trustworthy Reference Values (encoded as SWID or CoSWID tags [I-D.birkholz-yang-swid]). These reference measurements will eventually be compared to signed PCR Evidence ('quotes') acquired from an Attester's TPM using Attestation Policies chosen by the administrator or owner of the device.

This document does not specify the format or contents for the Appraisal Policy for Evidence, but Reference Values may be acquired in one of two ways:

1. a Verifier may obtain reference measurements directly from an Reference Value Provider chosen by the Verifier administrator (e.g., through a web site).

2. Signed reference measurements may be distributed by the Reference Value Provider to the Attester, as part of a software update. From there, the reference measurement may be acquired by the Verifier.

In either case, the Verifier Owner MUST select the source of trusted Reference Values through the Appraisal Policy for Evidence.

```
   *****************           .-------------.           .-----------.
   *Reference Value *          | Attester    |           | Verifier/ |
   *Provider        *          |             |           | Relying   |
   *(Device         *----2--->|  (Network   |----2--->| Party     |
   *config          *          |  Device)    |           |(Ntwk Mgmt |
   *Authority)      *          |             |           |  Station) |
   *****************           '-------------'           '-----------'
           |                                                  ^
           |                                                  |
           '---------------1----------------------------'
```

Figure 4: Appraisal Policy for Evidence Prerequisites

In either case the Reference Values must be generated, acquired and delivered in a secure way, including reference measurements of firmware and bootable modules taken according to TCG PC Client [PC-Client-BIOS-TPM-2.0] and Linux IMA [IMA]. Reference Values MUST be encoded as SWID or CoSWID tags, signed by the device manufacturer, as defined in the TCG RIM document [RIM], compatible with NIST IR 8060 [NIST-IR-8060] or the IETF CoSWID draft [I-D.ietf-sacm-coswid].

3.2. Reference Model for Challenge-Response

Once the prerequisites for RIV are met, a Verifier is able to acquire Evidence from an Attester. The following diagram illustrates a RIV information flow between a Verifier and an Attester, derived from Section 8.1 of [I-D.birkholz-rats-reference-interaction-model]. Event times shown correspond to the time types described within Appendix A of [I-D.ietf-rats-architecture]:

```
   .----------.                            .--------------------------.
   | Attester |                            | Relying Party / Verifier |
   '----------'                            '--------------------------'
      time(VG)                                                    |
   valueGeneration(targetEnvironment)                             |
        | => claims                                               |
        |                                                         |
        | <-----------requestEvidence(nonce, PcrSelection)----time(NS)
        |                                                         |
      time(EG)                                                    |
   evidenceGeneration(nonce, PcrSelection, collectedClaims)       |
        | => SignedPcrEvidence(nonce, PcrSelection)               |
        | => LogEvidence(collectedClaims)                         |
        |                                                         |
        | returnSignedPcrEvidence-------------------------------> |
        | returnLogEvidence-------------------------------------> |
        |                                                         |
        |                                                 time(RG,RA)
        |    evidenceAppraisal(SignedPcrEvidence, eventLog, refClaims)
        |                                    attestationResult <= |
        ~                                                         ~
        |                                                    time(RX)
```

Figure 5: IETF Attestation Information Flow

o Step 1 (time(VG)): One or more Attesting Network Device PCRs are extended with measurements. RIV provides no direct link between the time at which the event takes place and the time that it's attested, although streaming attestation as in [I-D.birkholz-rats-network-device-subscription] could.

o Step 2 (time(NS)): The Verifier generates a unique random nonce ("number used once"), and makes a request attestation data for one or more PCRs from an Attester. For interoperability, this MUST be accomplished via a YANG [RFC7950] interface that implements the TCG TAP model (e.g., YANG Module for Basic Challenge-Response-based Remote Attestation Procedures [I-D.ietf-rats-yang-tpm-charra]).

o Step 3 (time(EG)): On the Attester, measured values are retrieved from the Attester's TPM. This requested PCR evidence, along with the Verifier's nonce, called a Quote, is signed by the Attestation Key (AK) associated with the DevID. Quotes are retrieved according to TCG TAP Information Model [TAP]. At the same time, the Attester collects log evidence showing the values have been extended into that PCR. Section 9.1 gives more detail on how this works.

- Step 4: Collected Evidence is passed from the Attester to the Verifier

- Step 5 (time(RG,RA)): The Verifier reviews the Evidence and takes action as needed. As the interaction between Relying Party and Verifier is out of scope for RIV, this can be described as one step.

  * If the signature covering TPM Evidence is not correct, the device SHOULD NOT be trusted.

  * If the nonce in the response doesn't match the Verifier's nonce, the response may be a replay, and device SHOULD NOT be trusted.

  * If the signed PCR values do not match the set of log entries which have extended a particular PCR, the device SHOULD NOT be trusted.

  * If the log entries that the Verifier considers important do not match known good values, the device SHOULD NOT be trusted. We note that the process of collecting and analyzing the log can be omitted if the value in the relevant PCR is already a known-good value.

  * If the set of log entries are not seen as acceptable by the Appraisal Policy for Evidence, the device SHOULD NOT be trusted.

  * If time(RG)-time(NS) is greater than the Appraisal Policy for Evidence's threshold for assessing freshness, the Evidence is considered stale and SHOULD NOT be trusted.

3.2.1. Transport and Encoding

Network Management systems may retrieve signed PCR based Evidence as shown in Figure 5, and can be accomplished via NETCONF or RESTCONF, with XML, JSON, or CBOR encoded Evidence.

Implementations that use NETCONF MUST do so over a TLS or SSH secure tunnel. Implementations that use RESTCONF transport MUST do so over a TLS or SSH secure tunnel.

Retrieval of Log Evidence SHOULD be done via log interfaces specified in [I-D.ietf-rats-yang-tpm-charra]).

3.3. Centralized vs Peer-to-Peer

Figure 5 above assumes that the Verifier is trusted, while the Attester is not. In a Peer-to-Peer application such as two routers negotiating a trust relationship [I-D.voit-rats-trusted-path-routing], the two peers can each ask the other to prove software integrity. In this application, the information flow is the same, but each side plays a role both as an Attester and a Verifier. Each device issues a challenge, and each device responds to the other's challenge, as shown in Figure 6. Peer-to-peer challenges, particularly if used to establish a trust relationship between routers, require devices to carry their own signed reference measurements (RIMs). Devices may also have to carry Appraisal Policy for Evidence for each possible peer device so that each device has everything needed for remote attestation, without having to resort to a central authority.

```
   +----------------+                             +----------------+
   | RefVal         |                             | RefVal         |
   | Provider A     |                             | Provider B     |
   | Firmware       |                             | Firmware       |
   | Configuration  |                             | Configuration  |
   | Authority      |                             | Authority      |
   |                |                             |                |
   +----------------+                             +----------------+
         |                                               |
         |       +------------+        +------------+    |
         |       |            | Step 1 |            |    |   \
         |       |  Attester  |<------>|  Verifier  |    |   |
         |       |            |<------>|            |    |   |  Router B
         +------>|            | Step 2 |            |    |   |- Challenges
          Step 0A|            |        |            |    |   |  Router A
                 |            |------->|            |    |   |
                 |- Router A -| Step 3 |- Router B -|    |   /
                 |            |        |            |    |
                 |            |        |            |    |
                 |            | Step 1 |            |    |   \
                 |  Verifier  |<------>|  Attester  |<-+     |  Router A
                 |            |<------>|            |        |- Challenges
                 |            | Step 2 |            |        |  Router B
                 |            |        |            |        |
                 |            |<-------|            |        |
                 +------------+ Step 3 +------------+        /
```

Figure 6: Peer-to-Peer Attestation Information Flow

In this application, each device may need to be equipped with signed RIMs to act as an Attester, and also an Appraisal Policy for Evidence and a selection of trusted X.509 root certificates, to allow the device to act as a Verifier. An existing link layer protocol such as 802.1X [IEEE-802.1X] or 802.1AE [IEEE-802.1AE], with Evidence being enclosed over a variant of EAP [RFC3748] or LLDP [LLDP] are suitable methods for such an exchange.

4. Privacy Considerations

Network equipment, such as routers, switches and firewalls, has a key role to play in guarding the privacy of individuals using the network. Network equipment generally adheres to several rules to protect privacy:

- o Packets passing through the device must not be sent to unauthorized destinations. For example:

  * Routers often act as Policy Enforcement Points, where individual subscribers may be checked for authorization to access a network. Subscriber login information must not be released to unauthorized parties.

  * Network equipment is often called upon to block access to protected resources from unauthorized users.

- o Routing information, such as the identity of a router's peers, must not be leaked to unauthorized neighbors.

- o If configured, encryption and decryption of traffic must be carried out reliably, while protecting keys and credentials.

Functions that protect privacy are implemented as part of each layer of hardware and software that makes up the networking device. In light of these requirements for protecting the privacy of users of the network, the network equipment must identify itself, and its boot configuration and measured device state (for example, PCR values), to the equipment's administrator, so there's no uncertainty as to what function each device and configuration is configured to carry out. Attestation is a component that allows the administrator to ensure that the network provides individual and peer privacy guarantees, even though the device itself may not have a right to keep its identity secret.

RIV specifically addresses the collection of information from enterprise network devices by authorized agents of the enterprise. As such, privacy is a fundamental concern for those deploying this solution, given EU GDPR, California CCPA, and many other privacy regulations. The enterprise SHOULD implement and enforce their duty of care.

See [NetEq] for more context on privacy in networking devices.

5. Security Considerations

Attestation Evidence from the RIV procedure are subject to a number of attacks:

- Keys may be compromised.

- A counterfeit device may attempt to impersonate (spoof) a known authentic device.

- Man-in-the-middle attacks may be used by a counterfeit device to attempt to deliver responses that originate in an actual authentic device.

- Replay attacks may be attempted by a compromised device.

5.1. Keys Used in RIV

Trustworthiness of RIV attestation depends strongly on the validity of keys used for identity and attestation reports. RIV takes full advantage of TPM capabilities to ensure that evidence can be trusted.

Two sets of key-pairs are relevant to RIV attestation:

- A DevID key-pair is used to certify the identity of the device in which the TPM is installed.

- An Attestation Key-pair (AK) key is used to certify attestation Evidence (called 'quotes' in TCG documents), used to provide evidence for integrity of the software on the device

TPM practices usually require that these keys be different, as a way of ensuring that a general-purpose signing key cannot be used to spoof an attestation quote.

In each case, the private half of the key is known only to the TPM, and cannot be retrieved externally, even by a trusted party. To ensure that's the case, specification-compliant private/public key-pairs are generated inside the TPM, where they're never exposed, and cannot be extracted (See [Platform-DevID-TPM-2.0]).

Keeping keys safe is a critical enabler of trustworthiness, but it's just part of attestation security; knowing which keys are bound to the device in question is just as important in an environment where private keys are never exposed.

While there are many ways to manage keys in a TPM (see [Platform-DevID-TPM-2.0]), RIV includes support for "zero touch" provisioning (also known as zero-touch onboarding) of fielded devices (e.g., Secure ZTP, [RFC8572]), where keys which have predictable trust properties are provisioned by the device vendor.

Device identity in RIV is based on IEEE 802.1AR Device Identity (DevID). This specification provides several elements:

- o A DevID requires a unique key pair for each device, accompanied by an X.509 certificate,

- o The private portion of the DevID key is to be stored in the device, in a manner that provides confidentiality (Section 6.2.5 [IEEE-802-1AR])

The X.509 certificate contains several components:

- o The public part of the unique DevID key assigned to that device allows a challenge of identity.

- o An identifying string that's unique to the manufacturer of the device. This is normally the serial number of the unit, which might also be printed on a label on the device.

- o The certificate must be signed by a key traceable to the manufacturer's root key.

With these elements, the device's manufacturer and serial number can be identified by analyzing the DevID certificate plus the chain of intermediate certificates leading back to the manufacturer's root certificate. As is conventional in TLS or SSH connections, a random nonce must be signed by the device in response to a challenge, proving possession of its DevID private key.

RIV uses the DevID to validate a TLS or SSH connection to the device as the attestation session begins. Security of this process derives from TLS or SSH security, with the DevID providing proof that the session terminates on the intended device. See [RFC8446], [RFC4253].

Evidence of software integrity is delivered in the form of a quote signed by the TPM itself. Because the contents of the quote are signed inside the TPM, any external modification (including reformatting to a different data format) after measurements have been taken will be detected as tampering. An unbroken chain of trust is essential to ensuring that blocks of code that are taking measurements have been verified before execution (see Figure 1).

Requiring measurements of the operating software to be signed by a key known only to the TPM also removes the need to trust the device's operating software (beyond the first measurement in the RTM; see below); any changes to the quote, generated and signed by the TPM itself, made by malicious device software, or in the path back to the Verifier, will invalidate the signature on the quote.

A critical feature of the YANG model described in [I-D.ietf-rats-yang-tpm-charra] is the ability to carry TPM data structures in their native format, without requiring any changes to the structures as they were signed and delivered by the TPM. While alternate methods of conveying TPM quotes could compress out redundant information, or add an additional layer of signing using external keys, the implementation MUST preserve the TPM signing, so that tampering anywhere in the path between the TPM itself and the Verifier can be detected.

5.2. Prevention of Spoofing and Man-in-the-Middle Attacks

Prevention of spoofing attacks against attestation systems is also important. There are two cases to consider:

- o The entire device could be spoofed. If the Verifier goes to appraise a specific Attester, it might be redirected to a different Attester. Use of the 802.1AR Device Identity (DevID) in the TPM ensures that the Verifier's TLS or SSH session is in fact terminating on the right device.

- o A device with a compromised OS could return a fabricated quote providing spoofed attestation Evidence.

Protection against spoofed quotes from a device with valid identity is a bit more complex. An identity key must be available to sign any kind of nonce or hash offered by the Verifier, and consequently, could be used to sign a fabricated quote. To block a spoofed Attestation Result, the quote generated inside the TPM must be signed by a key that's different from the DevID, called an Attestation Key (AK).

Given separate Attestation and DevID keys, the binding between the AK and the same device must also be proven to prevent a man-in-the-middle attack (e.g., the 'Asokan Attack' [RFC6813]).

This is accomplished in RIV through use of an AK certificate with the same elements as the DevID (same manufacturer's serial number, signed by the same manufacturer's key), but containing the device's unique AK public key instead of the DevID public key.

The TCG document TPM 2.0 Keys for Device Identity and Attestation [Platform-DevID-TPM-2.0] specifies OIDs for Attestation Certificates that allow the CA to mark a key as specifically known to be an Attestation key.

These two key-pairs and certificates are used together:

- o The DevID is used to validate a TLS connection terminating on the device with a known serial number.

- o The AK is used to sign attestation quotes, providing proof that the attestation evidence comes from the same device.

5.3. Replay Attacks

Replay attacks, where results of a previous attestation are submitted in response to subsequent requests, are usually prevented by inclusion of a random nonce in the request to the TPM for a quote. Each request from the Verifier includes a new random number (a nonce). The resulting quote signed by the TPM contains the same nonce, allowing the Verifier to determine freshness, (i.e., that the resulting quote was generated in response to the Verifier's specific request). Time-Based Uni-directional Attestation [I-D.birkholz-rats-tuda] provides an alternate mechanism to verify freshness without requiring a request/response cycle.

5.4. Owner-Signed Keys

Although device manufacturers MUST pre-provision devices with easily verified DevID and AK certificates if zero-touch provisioning such as described in [RFC8572] is to be supported, use of those credentials is not mandatory. IEEE 802.1AR incorporates the idea of an Initial Device ID (IDevID), provisioned by the manufacturer, and a Local Device ID (LDevID) provisioned by the owner of the device. RIV and [Platform-DevID-TPM-2.0] extends that concept by defining an Initial Attestation Key (IAK) and Local Attestation Key (LAK) with the same properties.

Device owners can use any method to provision the Local credentials.

- o TCG document [Platform-DevID-TPM-2.0] shows how the initial Attestation keys can be used to certify LDevID and LAK keys. Use of the LDevID and LAK allows the device owner to use a uniform identity structure across device types from multiple manufacturers (in the same way that an "Asset Tag" is used by many enterprises to identify devices they own). TCG document [Provisioning-TPM-2.0] also contains guidance on provisioning Initial and Local identity keys in TPM 2.0.

- o Device owners, however, can use any other mechanism they want to assure themselves that local identity certificates are inserted into the intended device, including physical inspection and programming in a secure location, if they prefer to avoid placing trust in the manufacturer-provided keys.

Clearly, local keys can't be used for secure Zero Touch provisioning; installation of the local keys can only be done by some process that runs before the device is installed for network operation.

On the other end of the device life cycle, provision should be made to wipe local keys when a device is decommissioned, to indicate that the device is no longer owned by the enterprise. The manufacturer's Initial identity keys must be preserved, as they contain no information that's not already printed on the device's serial number plate.

5.5. Other Factors for Trustworthy Operation

In addition to trustworthy provisioning of keys, RIV depends on a number of other factors for trustworthy operation.

- o Secure identity depends on mechanisms to prevent per-device secret keys from being compromised. The TPM provides this capability as a Root of Trust for Storage.

- o Attestation depends on an unbroken chain of measurements, starting from the very first measurement. See Section 9.1 for background on TPM practices.

- o That first measurement is made by code called the Root of Trust for Measurement, typically done by trusted firmware stored in boot flash. Mechanisms for maintaining the trustworthiness of the RTM are out of scope for RIV, but could include immutable firmware, signed updates, or a vendor-specific hardware verification technique. See Section 9.2 for background on roots of trust.

- o The device owner SHOULD provide some level of physical defense for the device. If a TPM that has already been programmed with an authentic DevID is stolen and inserted into a counterfeit device, attestation of that counterfeit device may become indistinguishable from an authentic device.

RIV also depends on reliable Reference Values, as expressed by the RIM [RIM]. The definition of trust procedures for RIMs is out of scope for RIV, and the device owner is free to use any policy to validate a set of reference measurements. RIMs may be conveyed out-of-band or in-band, as part of the attestation process (see Section 3.1.3). But for embedded devices, where software is usually shipped as a self-contained package, RIMs signed by the manufacturer and delivered in-band may be more convenient for the device owner.

The validity of RIV attestation results is also influenced by procedures used to create Reference Values:

o While the RIM itself is signed, supply-chains SHOULD be carefully scrutinized to ensure that the values are not subject to unexpected manipulation prior to signing. Insider-attacks against code bases and build chains are particularly hard to spot.

o Designers SHOULD guard against hash collision attacks. Reference Integrity Manifests often give hashes for large objects of indeterminate size; if one of the measured objects can be replaced with an implant engineered to produce the same hash, RIV will be unable to detect the substitution. TPM1.2 uses SHA-1 hashes only, which have been shown to be susceptible to collision attack. TPM2.0 will produce quotes with SHA-256, which so far has resisted such attacks. Consequently, RIV implementations SHOULD use TPM2.0.

6. Conclusion

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

o Platform identity can be based on IEEE 802.1AR Device Identity, coupled with careful supply-chain management by the manufacturer.

o Complex supply chains can be certified using TCG Platform Certificates [Platform-Certificates].

o The TCG TAP mechanism couple with [I-D.ietf-rats-yang-tpm-charra] can be used to retrieve attestation evidence.

o Reference Values must be conveyed from the software authority (e.g., the manufacturer) in Reference Integrity Manifests, to the system in which verification will take place. IETF and TCG SWID and CoSWID work ([I-D.birkholz-yang-swid], [RIM])) forms the basis for this function.

7. IANA Considerations

This memo includes no request to IANA.

8. Acknowledgements

The authors wish to thank numerous reviewers for generous assistance, including William Bellingrath, Mark Baushke, Ned Smith, Henk Birkholz, Tom Laffey, Dave Thaler, Wei Pan, Michael Eckel, Thomas Hardjono, Bill Sulzen, Monty Wiseman, Kathleen Moriarty, Nancy Cam-Winget and Shwetha Bhandari

9. Appendix

9.1. Using a TPM for Attestation

The Trusted Platform Module and surrounding ecosystem provide three interlocking capabilities to enable secure collection of evidence from a remote device, Platform Configuration Registers (PCRs), a Quote mechanism, and a standardized Event Log.

Each TPM has at least eight and at most twenty-four PCRs (depending on the profile and vendor choices), each one large enough to hold one hash value (SHA-1, SHA-256, and other hash algorithms can be used, depending on TPM version). PCRs can't be accessed directly from outside the chip, but the TPM interface provides a way to "extend" a new security measurement hash into any PCR, a process by which the existing value in the PCR is hashed with the new security measurement hash, and the result placed back into the same PCR. The result is a composite fingerprint of all the security measurements extended into each PCR since the system was reset.

Every time a PCR is extended, an entry should be added to the corresponding Event Log. Logs contain the security measurement hash plus informative fields offering hints as to which event generated the security measurement. The Event Log itself is protected against accidental manipulation, but it is implicitly tamper-evident - any verification process can read the security measurement hash from the log events, compute the composite value and compare that to what ended up in the PCR. If there's no discrepancy, the logs do provide an accurate view of what was placed into the PCR.

Note that the composite hash-of-hashes recorded in PCRs is order-dependent, resulting in different PCR values for different ordering of the same set of events (e.g. Event A followed by Event B yields a different PCR value than B followed by A). For single-threaded code, where both the events and their order are fixed, a Verifier may validate a single PCR value, and use the log only to diagnose a mismatch from Reference Values. However, operating system code is usually non-deterministic, meaning that there may never be a single "known good" PCR value. In this case, the Verifier may have to verify that the log is correct, and then analyze each item in the log to determine if it represents an authorized event.

In a conventional TPM Attestation environment, the first measurement must be made and extended into the TPM by trusted device code (called the Root of Trust for Measurement, RTM). That first measurement should cover the segment of code that is run immediately after the RTM, which then measures the next code segment before running it, and so on, forming an unbroken chain of trust. See [TCGRoT] for more on Mutable vs Immutable roots of trust.

The TPM provides another mechanism called a Quote that can read the current value of the PCRs and package them, along with the Verifier's nonce, into a TPM-specific data structure signed by an Attestation private key, known only to the TPM.

As noted above in Section 5 Security Considerations, it's important to note that the Quote data structure is signed inside the TPM. The trust model is preserved by retrieving the Quote in a way that does not invalidate the signature, as specified in [I-D.ietf-rats-yang-tpm-charra].

The Verifier uses the Quote and Log together. The Quote contains the composite hash of the complete sequence of security measurement hashes, signed by the TPM's private Attestation Key. The Log contains a record of each measurement extended into the TPM's PCRs. By computing the composite hash of all the measurements, the Verifier can verify the integrity of the Event Log, even though the Event Log itself is not signed. Each hash in the validated Event Log can then be compared to corresponding expected values in the set of Reference Values to validate overall system integrity.

A summary of information exchanged in obtaining quotes from TPM1.2 and TPM2.0 can be found in [TAP], Section 4. Detailed information about PCRs and Quote data structures can be found in [TPM1.2], [TPM2.0]. Recommended log formats include [PC-Client-BIOS-TPM-2.0] and [Canonical-Event-Log].

9.2. Root of Trust for Measurement

The measurements needed for attestation require that the device being attested is equipped with a Root of Trust for Measurement, that is, some trustworthy mechanism that can compute the first measurement in the chain of trust required to attest that each stage of system startup is verified, a Root of Trust for Storage (i.e., the TPM PCRs) to record the results, and a Root of Trust for Reporting to report the results [TCGRoT], [SP800-155], [SP800-193].

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

o The first measurement computed by the Root of Trust for Measurement, and stored in the TPM's Root of Trust for Storage, must be assumed to be correct.

o There must not be a way to reset the Root of Trust for Storage without re-entering the Root of Trust for Measurement code.

The first measurement must be computed by code that is implicitly trusted; if that first measurement can be subverted, none of the remaining measurements can be trusted. (See [SP800-155])

It's important to note that the trustworthiness of the RTM code cannot be assured by the TPM or TPM supplier - code or procedures external to the TPM must guarantee the security of the RTM.

9.3. Layering Model for Network Equipment Attester and Verifier

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

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

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

       .......................            .......................
       . Reference Integrity .            .  TAP (PTS2.0) Info  .
       .      Manifest       .            . Model and Canonical .
       .                     .            .     Log Format      .
       .......................            .......................

       *************************  .............. **********************
       * YANG SWID Module      *  . TCG        . * YANG Attestation   *
       * I-D.birkholz-yang-swid*  . Attestation. * Module             *
       *                       *  . MIB        . * I-D.ietf-rats-     *
       *                       *  .            . * yang-tpm-charra    *
       *************************  .............. **********************

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

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

       *************************               ************************
       *       TLS, SSH        *               *       TLS, SSH       *
       *************************               ************************
```

Figure 7: RIV Protocol Stacks

IETF documents are captured in boxes surrounded by asterisks. TCG documents are shown in boxes surrounded by dots.

9.4. Implementation Notes

Figure 8 summarizes many of the actions needed to complete an Attestation system, with links to relevant documents. While documents are controlled by several standards organizations, the implied actions required for implementation are all the responsibility of the manufacturer of the device, unless otherwise noted. It should be noted that, while the YANG model is RECOMMENDED for attestation, this table identifies an optional SNMP MIB as well, [Attest-MIB].

Component	Controlling Specification
Make a Secure execution environment o Attestation depends on a secure root of trust for measurement outside the TPM, as well as roots for storage and reporting inside the TPM. o Refer to TCG Root of Trust for Measurement. o NIST SP 800-193 also provides guidelines on Roots of Trust	TCG RoT UEFI.org
Provision the TPM as described in TCG documents.	[Platform-DevID-TPM-2.0] TCG Platform Certificate
Put a DevID or Platform Cert in the TPM o Install an Initial Attestation Key at the same time so that Attestation can work out of the box o Equipment suppliers and owners may want to implement Local Device ID as well as Initial Device ID	TCG TPM DevID TCG Platform Certificate IEEE 802.1AR
Connect the TPM to the TLS stack o Use the DevID in the TPM to authenticate TAP connections, identifying the device	Vendor TLS stack (This action is simply configuring TLS to use the DevID as its client certificate)
Make CoSWID tags for BIOS/LoaderLKernel objects o Add reference measurements into SWID tags o Manufacturer should sign the SWID tags	IETF CoSWID ISO/IEC 19770-2 NIST IR 8060

```
|    o  The TCG RIM-IM identifies further         |                 |
|       procedures to create signed RIM           |                 |
|       documents that provide the necessary      |                 |
|       reference information                     |                 |
-----------------------------------------------------------------------
|  Package the SWID tags with a vendor software   | Retrieve tags   |
|  release                                        | with            |
|    o  A tag-generator plugin such      | draft-birkholz-yang-swid|
|     as [SWID-Gen] can be used                   |-----------------|
|                                                 | TCG PC Client   |
|                                                 | RIM             |
-----------------------------------------------------------------------
|  Use PC Client measurement definitions          | TCG PC Client   |
|  to define the use of PCRs                      | BIOS            |
|  (although Windows  OS is rare on Networking    |                 |
|  Equipment, UEFI BIOS is not)                   |                 |
-----------------------------------------------------------------------
|  Use TAP to retrieve measurements               |                 |
|    o  Map TAP to SNMP                           | TCG SNMP MIB    |
|    o  Map to YANG                               | YANG Module for |
|  Use Canonical Log Format                       |   Basic         |
|                                                 |   Attestation   |
|                                                 | TCG Canonical   |
|                                                 |   Log Format    |
-----------------------------------------------------------------------
| Posture Collection Server (as described in IETF |                 |
|  SACMs ECP) should request the                  |                 |
|  attestation and analyze the result             |                 |
| The Management application might be broken down|                 |
|  to several more components:                    |                 |
|    o  A Posture Manager Server                  |                 |
|       which collects reports and stores them in |                 |
|       a database                                |                 |
|    o  One or more Analyzers that can look at the|                 |
|       results and figure out what it means.     |                 |
-----------------------------------------------------------------------
```

Figure 8: Component Status

10. References

10.1. Normative References

[Canonical-Event-Log]
Trusted Computing Group, "DRAFT Canonical Event Log Format Version: 1.0, Revision: .12", October 2018.

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

[I-D.ietf-rats-yang-tpm-charra]
Birkholz, H., Eckel, M., Voit, E., Bhandari, S., Sulzen, B., Xia, L., Laffey, T., and G. Fedorkow, "A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs", draft-ietf-rats-yang-tpm-charra-03 (work in progress), September 2020.

[I-D.ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-16 (work in progress), November 2020.

[IEEE-802-1AR]
Seaman, M., "802.1AR-2018 - IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity, IEEE Computer Society", August 2018.

[PC-Client-BIOS-TPM-1.2]
Trusted Computing Group, "TCG PC Client Specific Implementation Specification for Conventional BIOS, Specification Version 1.21 Errata, Revision 1.00", February 2012, <https://trustedcomputinggroup.org/resource/pc-client-work-group-specific-implementation-specification-for-conventional-bios/>.

[PC-Client-BIOS-TPM-2.0]
Trusted Computing Group, "PC Client Specific Platform Firmware Profile Specification Family "2.0", Level 00 Revision 1.04", June 2019, <https://trustedcomputinggroup.org/pc-client-specific-platform-firmware-profile-specification>.

[PC-Client-RIM]
Trusted Computing Group, "DRAFT: TCG PC Client Reference Integrity Manifest Specification, v.09", December 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_PC_Client_RIM_r0p15_15june2020.pdf>.

[Platform-DevID-TPM-2.0]
Trusted Computing Group, "TPM 2.0 Keys for Device Identity and Attestation, Specification Version 1.0, Revision 2", September 2020, <https://trustedcomputinggroup.org/resource/tpm-2-0-keys-for-device-identity-and-attestation/>.

[Platform-ID-TPM-1.2]
Trusted Computing Group, "TPM Keys for Platform Identity for TPM 1.2, Specification Version 1.0, Revision 3", August 2015, <https://trustedcomputinggroup.org/resource/tpm-keys-for-platform-identity-for-tpm-1-2-2/>.

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

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

[RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <https://www.rfc-editor.org/info/rfc8572>.

[RIM] Trusted Computing Group, "DRAFT: TCG Reference Integrity Manifest Information Model", June 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_RIM_Model_v1-r13_2feb20.pdf>.

[SWID] The International Organization for Standardization/ International Electrotechnical Commission, "Information Technology Software Asset Management Part 2: Software Identification Tag, ISO/IEC 19770-2", October 2015, <https://www.iso.org/standard/65666.html>.

[TAP] Trusted Computing Group, "TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0, Version 1.0, Revision 0.36", October 2018, <https://trustedcomputinggroup.org/resource/tcg-tap-information-model/>.

10.2. Informative References

[AK-Enrollment]
Trusted Computing Group, "TCG Infrastructure Working Group - A CMC Profile for AIK Certificate Enrollment Version 1.0, Revision 7", March 2011, <https://trustedcomputinggroup.org/resource/tcg-infrastructure-working-group-a-cmc-profile-for-aik-certificate-enrollment/>.

[Attest-MIB]
Trusted Computing Group, "SNMP MIB for TPM-Based Attestation, Version 0.8Revision 0.02", May 2018, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_SNMP_MIB_for_TPM-Based_Attestation_v0.8r2_PUBLIC_REVIEW.pdf>.

[EFI-TPM] Trusted Computing Group, "TCG EFI Platform Specification for TPM Family 1.1 or 1.2, Specification Version 1.22, Revision 15", January 2014, <https://trustedcomputinggroup.org/resource/tcg-efi-platform-specification/>.

[I-D.birkholz-rats-network-device-subscription]
Birkholz, H., Voit, E., and W. Pan, "Attestation Event Stream Subscription", draft-birkholz-rats-network-device-subscription-01 (work in progress), October 2020.

[I-D.birkholz-rats-reference-interaction-model]
Birkholz, H., Eckel, M., Newton, C., and L. Chen, "Reference Interaction Models for Remote Attestation Procedures", draft-birkholz-rats-reference-interaction-model-03 (work in progress), July 2020.

[I-D.birkholz-rats-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-rats-tuda-03 (work in progress), July 2020.

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-07 (work in progress), October 2020.

[I-D.ietf-rats-eat]
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-06 (work in progress), December 2020.

[I-D.richardson-rats-usecases]
Richardson, M., Wallace, C., and W. Pan, "Use cases for Remote Attestation common encodings", draft-richardson-rats-usecases-08 (work in progress), November 2020.

[I-D.voit-rats-trusted-path-routing]
Voit, E., "Trusted Path Routing", draft-voit-rats-trusted-path-routing-02 (work in progress), June 2020.

[IEEE-802.1AE]
Seaman, M., "802.1AE MAC Security (MACsec)", 2018, <https://1.ieee802.org/security/802-1ae/>.

[IEEE-802.1X]
IEEE Computer Society, "802.1X-2020 - IEEE Standard for Local and Metropolitan Area Networks--Port-Based Network Access Control", February 2020, <https://standards.ieee.org/standard/802_1X-2020.html>.

[IMA] and , "Integrity Measurement Architecture", June 2019, <https://sourceforge.net/p/linux-ima/wiki/Home/>.

[LLDP] IEEE Computer Society, "802.1AB-2016 - IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery", March 2016, <https://standards.ieee.org/standard/802_1AB-2016.html>.

[NetEq] Trusted Computing Group, "TCG Guidance for Securing Network Equipment, Version 1.0, Revision 29", January 2018, <https://trustedcomputinggroup.org/resource/tcg-guidance-securing-network-equipment/>.

[NIST-IR-8060]
National Institute for Standards and Technology, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags", April 2016, <https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf>.

[Platform-Certificates]
Trusted Computing Group, "TCG Platform Attribute Credential Profile, Specification Version 1.0, Revision 16", January 2018, <https://trustedcomputinggroup.org/resource/tcg-platform-attribute-credential-profile/>.

[Provisioning-TPM-2.0]
Trusted Computing Group, "TCG TPM v2.0 Provisioning Guidance, Version 1.0, Revision 1.0", March 2015, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-TPM-v2.0-Provisioning-Guidance-Published-v1r1.pdf>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <https://www.rfc-editor.org/info/rfc3748>.

[RFC6813] Salowey, J. and S. Hanna, "The Network Endpoint Assessment (NEA) Asokan Attack Analysis", RFC 6813, DOI 10.17487/RFC6813, December 2012, <https://www.rfc-editor.org/info/rfc6813>.

[SP800-155]
National Institute of Standards and Technology, "BIOS Integrity Measurement Guidelines (Draft)", December 2011, <https://csrc.nist.gov/csrc/media/publications/sp/800-155/draft/documents/draft-sp800-155_dec2011.pdf>.

[SP800-193]
National Institute for Standards and Technology, "NIST Special Publication 800-193: Platform Firmware Resiliency Guidelines", April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-193.pdf>.

[SWID-Gen]
Labs64, Munich, Germany, "SoftWare IDentification (SWID) Tags Generator (Maven Plugin)", n.d., <https://github.com/Labs64/swid-maven-plugin>.

[TCGRoT] Trusted Computing Group, "DRAFT: TCG Roots of Trust Specification", October 2018, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_Roots_of_Trust_Specification_v0p20_PUBLIC_REVIEW.pdf>.

[TPM1.2] Trusted Computing Group, "TPM Main Specification Level 2 Version 1.2, Revision 116", March 2011, <https://trustedcomputinggroup.org/resource/tpm-main-specification/>.

[TPM2.0] Trusted Computing Group, "Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59", November 2019, <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

Authors' Addresses

Guy Fedorkow (editor)
Juniper Networks, Inc.
US

Email: gfedorkow@juniper.net

Eric Voit
Cisco Systems, Inc.
US

Email: evoit@cisco.com

Jessica Fitzgerald-McKay
National Security Agency
US

Email: jmfitz2@nsa.gov

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 16 October 2021

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

14 April 2021

A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs

draft-ietf-rats-yang-tpm-charra-07

Abstract

This document defines a YANG RPC and a small number of configuration node required to retrieve attestation evidence about integrity measurements from a device following the operational context defined in TPM-based Network Device Remote Integrity Verification. Complementary measurement logs are also provided by the YANG RPC originating from one or more roots of trust of measurement. The module defined requires at least one TPM 1.2 or 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

Table of Contents

1. Introduction

This document is based on the terminology defined in the [I-D.ietf-rats-architecture] and uses the operational context defined in [I-D.ietf-rats-tpm-based-network-device-attest] as well as the interaction model and information elements defined in [I-D.ietf-rats-reference-interaction-models]. The currently supported hardware security modules (HWM) are the Trusted Platform Module (TPM) [TPM1.2] and [TPM2.0] specified by the Trusted Computing Group (TCG). One ore more TPMs 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 is used as a root of trust for reporting (RTR) in order to retrieve attestation evidence from a composite device (quote primitive operation). Additionally, it is used as a root of trust for storage (RTS) in order to retain shielded secrets and store system measurements using a folding hash function (extend primitive operation).

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. The YANG Module for Basic Remote Attestation Procedures

One or more TPMs 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.ietf-rats-architecture] and the corresponding challenge-response interaction model defined in the [I-D.ietf-rats-reference-interaction-models] 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 TPMs per hardware component.

2.1. YANG Modules

2.1.1. ietf-tpm-remote-attestation

This YANG module imports modules from [RFC6991], [RFC8348], [I-D.ietf-netconf-keystore], ietf-tcg-algs.yang Section 2.1.2.3.

2.1.1.1. Features

This module supports the following features:

<TPMs> - Indicates that multiple TPMs on the device can support remote attestation, This feature is applicable in cases where multiple line cards, each with its own TPM.

<bios> - Indicates the device supports the retrieval of bios event logs.

<ima> - Indicates the device supports the retrieval of Integrity Measurement Architecture event logs.

<netequip_boot> - Indicates the device supports the retrieval of netequip boot event logs.

2.1.1.2. Identities

This module supports the following types of attestation event logs: <ima>, <bios>, and <netequip_boot>.

2.1.1.3. RPCs

2.1.1.3.1. <tpm20-challenge-response-attestation>

This RPC allows a Verifier to request a quote of PCRs from a TPM2.0 compliant cryptoprocessor. Where the feature <TPMs> is active, and one or more <certificate-name> is not provided, all TPM2.0 compliant cryptoprocessors will respond. A YANG tree diagram of this RPC is as follows:

```
+---x tpm20-challenge-response-attestation {taa:TPM20}?
   +---w input
   |  +---w tpm20-attestation-challenge
   |     +---w nonce-value            binary
   |     +---w tpm20-pcr-selection* []
   |     |  +---w TPM20-hash-algo?   identityref
   |     |  +---w pcr-index*         tpm:pcr
   |     +---w certificate-name*      certificate-name-ref {tpm:TPMs}?
   +--ro output
      +--ro tpm20-attestation-response* []
         +--ro certificate-name       certificate-name-ref
         +--ro TPMS_QUOTE_INFO        binary
         +--ro quote-signature?       binary
         +--ro up-time?               uint32
         +--ro unsigned-pcr-values* []
            +--ro TPM20-hash-algo?   identityref
            +--ro pcr-values* [pcr-index]
               +--ro pcr-index    pcr
               +--ro pcr-value?   binary
```

An example of an RPC challenge requesting PCRs 0-7 from a SHA256 bank could look like the following:

```
 <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <tpm20-challenge-response-attestation>
       xmlns="urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation">
     <nonce>110101010110011011111001010010100</nonce>
     <tpm20-pcr-selection>
       <TPM20-hash-algo
           xmlns:taa="urn:ietf:params:xml:ns:yang:ietf-tcg-algs">
         taa:TPM_ALG_SHA256
       </TPM20-hash-algo>
       <pcr-index>0</pcr-index>
       <pcr-index>1</pcr-index>
       <pcr-index>2</pcr-index>
       <pcr-index>3</pcr-index>
       <pcr-index>4</pcr-index>
       <pcr-index>5</pcr-index>
       <pcr-index>6</pcr-index>
       <pcr-index>7</pcr-index>
     </tpm20-pcr-selection>
   </tpm20-challenge-response-attestation>
 </rpc>
```

and a successful response might be formated as follows:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tpm12-attestation-response
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation">
    <certificate-name
        xmlns:ks=urn:ietf:params:xml:ns:yang:ietf-keystore>
       ks:(instance of Certificate name in the Keystore)
    </certificate-name>
    <TPMS_QUOTE_INFO>
       (raw information from the TPM Quote, this includes a digest
       across the requested PCRs, the nonce, TPM2 time counters.)
    </TPMS_QUOTE_INFO>
    <quote-signature>
        (signature across TPMS_QUOTE_INFO)
    </quote-signature>
  </tpm12-attestation-response>
</rpc-reply>
```

2.1.1.4. <tpm12-challenge-response-attestation>

This RPC allows a Verifier to request a quote of PCRs from a TPM1.2 compliant cryptoprocessor. Where the feature <TPMs> is active, and one or more <certificate-name> is not provided, all TPM1.2 compliant cryptoprocessors will respond. A YANG tree diagram of this RPC is as follows:

```
+---x tpm12-challenge-response-attestation {taa:TPM12}?
   +---w input
   |  +---w tpm12-attestation-challenge
   |     +---w pcr-index*          pcr
   |     +---w nonce-value         binary
   |     +---w certificate-name*   certificate-name-ref {tpm:TPMs}?
   +--ro output
      +--ro tpm12-attestation-response* []
         +--ro certificate-name    certificate-name-ref
         +--ro up-time?            uint32
         +--ro TPM_QUOTE2?         binary
```

2.1.1.5. <log-retrieval>

This RPC allows a Verifier to acquire the evidence which was extended into specific PCRs. A YANG tree diagram of this RPC is as follows:

```
+---x log-retrieval
   +---w input
   |  +---w log-selector* []
   |  |  +---w tpm-name*                      string
   |  |  +---w (index-type)?
   |  |  |  +--:(last-entry)
   |  |  |  |  +---w last-entry-value?    binary
   |  |  |  +--:(index)
   |  |  |  |  +---w last-index-number?   uint64
   |  |  |  +--:(timestamp)
   |  |  |     +---w timestamp?           yang:date-and-time
   |  |  +---w log-entry-quantity?        uint16
   |  +---w log-type        identityref
   +--ro output
      +--ro system-event-logs
         +--ro node-data* []
            +--ro tpm-name?     string
            +--ro up-time?      uint32
            +--ro log-result
               +--ro (attested_event_log_type)
                  +--:(bios)
                  |  +--ro bios-event-logs
                  |     +--ro bios-event-entry* [event-number]
                  |        +--ro event-number    uint32
                  |        +--ro event-type?     uint32
                  |        +--ro pcr-index?      pcr
                  |        +--ro digest-list* []
                  |        |  +--ro hash-algo?   identityref
                  |        |  +--ro digest*      binary
                  |        +--ro event-size?     uint32
                  |        +--ro event-data*     uint8
                  |
```

```
                  +--:(ima)
                  |  +--ro ima-event-logs
                  |     +--ro ima-event-entry* [event-number]
                  |        +--ro event-number               uint64
                  |        +--ro ima-template?              string
                  |        +--ro filename-hint?             string
                  |        +--ro filedata-hash?             binary
                  |        +--ro filedata-hash-algorithm?   string
                  |        +--ro template-hash-algorithm?   string
                  |        +--ro template-hash?             binary
                  |        +--ro pcr-index?                 pcr
                  |        +--ro signature?                 binary
                  +--:(netequip_boot)
                     +--ro boot-event-logs
                        +--ro boot-event-entry* [event-number]
                           +--ro event-number               uint64
                           +--ro filename-hint?             string
                           +--ro filedata-hash?             binary
                           +--ro filedata-hash-algorithm?   string
                           +--ro file-version?              string
                           +--ro file-type?                 string
                           +--ro pcr-index?                 pcr
```

2.1.1.6. Data Nodes

This section provides a high level description of the data nodes containing the configuration and operational objects with the YANG model. For more details, please see the YANG model itself in Section 2.1.1.7.

container <rats-support-structures> - This houses the set of information relating to a device's TPM(s).

container <tpms> - Provides configuration and operational details for each supported TPM, including the tpm-firmware-version, PCRs which may be quoted, certificates which are associated with that TPM, and the current operational status. Of note is the certificates which are associated with that TPM. As a certificate is associated with a single Attestation key, knowledge of the certificate allows a specific TPM to be identified.

```
   +--rw tpms
      +--rw tpm* [tpm-name]
         +--rw tpm-name                 string
         +--ro hardware-based?          boolean
         +--ro tpm-physical-index?      int32 {ietfhw:entity-mib}?
         +--ro tpm-path?                string
         +--ro compute-node             compute-node-ref {tpm:TPMs}?
         +--ro tpm-manufacturer?        string
         +--rw tpm-firmware-version     identityref
         +--rw TPM12-hash-algo?         identityref
         +--rw TPM12-pcrs*              pcr
         +--rw tpm20-pcr-bank* [TPM20-hash-algo]
         |  +--rw TPM20-hash-algo     identityref
         |  +--rw pcr-index*          tpm:pcr
         +--ro tpm-status               enumeration
         +--rw certificates
            +--rw certificate* [certificate-name]
               +--rw certificate-name            string
               +--rw certificate-keystore-ref?   -> /ks:keystore/asymmetric-keys/
asymmetric-key/certificates/certificate/name
               +--rw certificate-type?           enumeration
```

container <attester-supported-algos> - Identifies which TCG algorithms are available for use the Attesting platform. This allows an operator to limit algorithms available for use by RPCs to just a desired set from the universe of all allowed by TCG.

```
   +--rw attester-supported-algos
      +--rw tpm12-asymmetric-signing*   identityref {taa:TPM12}?
      +--rw tpm12-hash*                 identityref {taa:TPM12}?
      +--rw tpm20-asymmetric-signing*   identityref {taa:TPM20}?
      +--rw tpm20-hash*                 identityref {taa:TPM20}?
```

container <compute-nodes> - When there is more than one TPM supported, this container maintains the set of information related to the compute associated with a specific TPM. This allows each specific TPM to identify on which <compute-node> it belongs.

```
   +--rw compute-nodes {tpm:TPMs}?
      +--ro compute-node* [node-id]
         +--ro node-id                 string
         +--ro node-physical-index?    int32 {ietfhw:entity-mib}?
         +--ro node-name?              string
         +--ro node-location?          string
```

2.1.1.7. YANG Module

```
<CODE BEGINS> file "ietf-tpm-remote-attestation@2021-03-16.yang"
module ietf-tpm-remote-attestation {
  namespace "urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation";
  prefix "tpm";

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-hardware {
    prefix ietfhw;
  }
  import ietf-keystore {
    prefix ks;
  }
  import ietf-tcg-algs {
    prefix taa;
  }

  organization
   "IETF RATS (Remote ATtestation procedureS) Working Group";

  contact
   "WG Web  : <http://datatracker.ietf.org/wg/rats/>
    WG List : <mailto:rats@ietf.org>
    Author  : Eric Voit <evoit@cisco.com>
    Author  : Henk Birkholz <henk.birkholz@sit.fraunhofer.de>
    Author  : Michael Eckel <michael.eckel@sit.fraunhofer.de>
    Author  : Shwetha Bhandari <shwetha.bhandari@thoughtspot.com>
    Author  : Bill Sulzen <bsulzen@cisco.com>
    Author  : Liang Xia (Frank) <frank.xialiang@huawei.com>
    Author  : Tom Laffey <tom.laffey@hpe.com>
    Author  : Guy Fedorkow <gfedorkow@juniper.net>";

  description
    "A YANG module to enable a TPM 1.2 and TPM 2.0 based
     remote attestation procedure using a challenge-response
     interaction model and the TPM 1.2 and TPM 2.0 Quote
     primitive operations.

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

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Simplified
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
```

```
         This version of this YANG module is part of RFC XXXX
         (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
         itself for full legal notices.

         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 (RFC 2119)
         (RFC 8174) when, and only when, they appear in all
         capitals, as shown here.";

      revision "2021-03-16" {
        description
          "Initial version";
        reference
          "draft-ietf-rats-yang-tpm-charra";
      }

      /*****************/
      /*   Features    */
      /*****************/

      feature TPMs {
        description
          "The device supports the remote attestation of multiple
          TPM based cryptoprocessors.";
      }

      feature bios {
        description
          "The device supports the bios logs.";
        reference
          "https://trustedcomputinggroup.org/wp-content/uploads/
           PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf
           Section 9.4.5.2";
      }

      feature ima {
        description
          "The device supports Integrity Measurement Architecture logs.";
        reference
          "https://www.trustedcomputinggroup.org/wp-content/uploads/
           TCG_IWG_CEL_v1_r0p30_13feb2021.pdf  Section 4.3";
      }

      feature netequip_boot {
        description
          "The device supports the netequip_boot logs.";
```

```
     }

     /*****************/
     /*   Typedefs    */
     /*****************/

     typedef pcr {
       type uint8 {
         range "0..31";
       }
       description
         "Valid index number for a PCR.  At this point 0-31 is viable.";
     }

     typedef compute-node-ref {
       type leafref {
         path "/tpm:rats-support-structures/tpm:compute-nodes" +
              "/tpm:compute-node/tpm:node-name";
       }
       description
         "This type is used to reference a hardware node.  It is quite
          possible this leafref will eventually point to another YANG
          module's node.";
     }

     typedef certificate-name-ref {
       type leafref {
         path "/tpm:rats-support-structures/tpm:tpms/tpm:tpm" +
              "/tpm:certificates/tpm:certificate/tpm:certificate-name";
       }
       description
         "A type which allows identification of a TPM based certificate.";
     }


     /******************/
     /*   Identities   */
     /******************/

     identity attested_event_log_type {
       description
         "Base identity allowing categorization of the reasons why and
          attested measurement has been taken on an Attester.";
     }

     identity ima {
       base attested_event_log_type;
       description
```

```
        "An event type recorded in IMA.";
    }

    identity bios {
      base attested_event_log_type;
      description
        "An event type associated with BIOS/UEFI.";
    }

    identity netequip_boot {
      base attested_event_log_type;
      description
        "An event type associated with Network Equipment Boot.";
    }

    /*****************/
    /*   Groupings   */
    /*****************/

    grouping TPM20-hash-algo {
      description
        "The cryptographic algorithm used to hash the TPM2 PCRs.  This
         must be from the list of platform supported options.";
      leaf TPM20-hash-algo {
        type identityref {
          base taa:hash;
        }
        must "/tpm:rats-support-structures/tpm:attester-supported-algos"
           + "/tpm:tpm20-hash" {
           error-message "Not a platform supported TPM20-hash-algo";
        }
        default taa:TPM_ALG_SHA256;
        description
          "The hash scheme that is used to hash a TPM1.2 PCR. This
           must be one of those supported by a platform.";
      }
    }

    grouping TPM12-hash-algo {
      description
        "The cryptographic algorithm used to hash the TPM1.2 PCRs.";
      leaf TPM12-hash-algo {
        type identityref {
          base taa:hash;
        }
        must "/tpm:rats-support-structures/tpm:attester-supported-algos"
           + "/tpm:tpm12-hash" {
           error-message "Not a platform supported TPM12-hash-algo";
```

```
         }
         default taa:TPM_ALG_SHA1;
         description
           "The hash scheme that is used to hash a TPM1.2 PCR. This
            MUST be one of those supported by a platform.  This assumes
            that an algorithm other than SHA1 can be supported on some
            TPM1.2 cryptoprocessor variant.";
       }
     }

     grouping nonce {
       description
         "A random number intended to be used once to show freshness
          and to allow the detection of replay attacks.";
       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-index {
         /*  the following XPATH must be updated to ensure that only
               selectable PCRs are allowed in the RPC
         must "/tpm:rats-support-structures/tpm:tpms" +
              "/tpm:tpm[tpm-name = current()]" +
              "/tpm:tpm[TPM12-pcrs = current()]" {
           error-message "Acquiring this PCR index is not supported";
         }
         */
         type pcr;
         description
           "The numbers/indexes of the PCRs. At the moment this is limited
            to 32.";
```

```
      }
    }

    grouping tpm20-pcr-selection {
      description
        "A Verifier can acquire one or more PCR values, which are hashed
         together in a TPM2B_DIGEST coming from the TPM2.  The selection
         list of desired PCRs and the Hash Algorithm is represented in
         this grouping.";
      list tpm20-pcr-selection {
        unique "TPM20-hash-algo";
        description
          "Specifies the list of PCRs and Hash Algorithms that can be
           returned within a TPM2B_DIGEST.";
        reference
          "https://www.trustedcomputinggroup.org/wp-content/uploads/
           TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.9.7";
        uses TPM20-hash-algo;
        leaf-list pcr-index {
          /*  the following XPATH must be updated to ensure that only
              selectable PCRs are allowed in the RPC
          must "/tpm:rats-support-structures/tpm:tpms" +
               "/tpm:tpm[tpm-name = current()]" +
               "/tpm:tpm20-pcr-bank[pcr-index = current()]" {
            error-message "Acquiring this PCR index is not supported";
          }
          */
          type tpm:pcr;
          description
            "The numbers of the PCRs that which are being tracked
             with a hash based on the TPM20-hash-algo.";
        }
      }
    }

    grouping certificate-name-ref {
      description
        "Identifies a certificate in a keystore.";
      leaf certificate-name {
        type certificate-name-ref;
        mandatory true;
        description
          "Identifies a certificate in a keystore.";
      }
    }

    grouping tpm-name {
      description
```

```
          "A unique TPM on a device.";
        leaf tpm-name {
          type string;
          description
            "Unique system generated name for a TPM on a device.";
        }
      }

      grouping tpm-name-selector {
        description
          "One or more TPM on a device.";
        leaf-list tpm-name {
          type string;
          config false;
          description
            "Name of one or more unique TPMs on a device.  If this object
             exists, a selection should pull only the objects related to
             these TPM(s).  If it does not exist, all qualifying TPMs that
             are 'hardware-based' equals true on the device are selected.";
        }
      }

      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";
        }
      }

      grouping tpm12-attestation {
        description
          "Contains an instance of TPM1.2 style signed cryptoprocessor
           measurements.  It is supplemented by unsigned Attester
           information.";
        uses node-uptime;
        leaf TPM_QUOTE2 {
          type binary;
          description
            "Result of a TPM1.2 Quote2 operation. This includes PCRs,
             signatures, locality, the provided nonce and other data which
             can be further parsed to appraise the Attester.";
          reference
            "TPM1.2 commands rev116 July 2007, Section 16.5";
        }
      }
```

```
     grouping tpm20-attestation {
       description
         "Contains an instance of TPM2 style signed cryptoprocessor
          measurements.  It is supplemented by unsigned Attester
          information.";
       leaf TPMS_QUOTE_INFO {
         type binary;
         mandatory true;
         description
           "A hash of the latest PCR values (and the hash algorithm used)
            which have been returned from a Verifier for the selected PCRs
            and Hash Algorithms.";
         reference
           "https://www.trustedcomputinggroup.org/wp-content/uploads/
            TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.12.1";
       }
       leaf quote-signature {
         type binary;
         description
           "Quote signature returned by TPM Quote.  The signature was
            generated using the key associated with the
            certificate-name.";
         reference
           "https://www.trustedcomputinggroup.org/wp-content/uploads/
            TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 11.2.1";
       }
       uses node-uptime;
       list unsigned-pcr-values {
         description
           "PCR values in each PCR bank. This might appear redundant with
            the TPM2B_DIGEST, but that digest is calculated across multiple
            PCRs.  Having to verify across multiple PCRs does not
            necessarily make it easy for a Verifier to appraise just the
            minimum set of PCR information which has changed since the last
            received TPM2B_DIGEST.  Put another way, why should a Verifier
            reconstruct the proper value of all PCR Quotes when only a
            single PCR has changed?

            To help this happen, if the Attester does know specific PCR
            values, the Attester can provide these individual values via
            'unsigned-pcr-values'.   By comparing this information to the
            what has previously been validated, it is possible for a
            Verifier to confirm the Attester's signature while eliminating
            significant processing.";
         uses TPM20-hash-algo;
         list pcr-values {
           key pcr-index;
           description
```

```
            "List of one PCR bank.";
          leaf pcr-index {
            type pcr;
            description
              "PCR index number.";
          }
          leaf pcr-value {
            type binary;
            description
              "PCR value.";
            reference
             "https://www.trustedcomputinggroup.org/wp-content/uploads/
              TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.9.7";
          }
        }
      }
    }


    grouping log-identifier {
      description
        "Identifier for type of log to be retrieved.";
      leaf log-type {
        type identityref {
          base attested_event_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 pcr;
```

```
        description
          "Defines the PCR index that this event extended";
      }
      list digest-list {
        description
          "Hash of event data";
        leaf hash-algo {
          type identityref {
            base taa:hash;
          }
          description
            "The hash scheme that is used to compress the event data in
             each of the leaf-list digest items.";
        }
        leaf-list digest {
          type binary;
          description
            "The hash of the event data using the algorithm of the
             'hash-algo' against '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 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 {
      description
```

```
          "Defines an hash log extend event for IMA measurements";
        reference
          "https://www.trustedcomputinggroup.org/wp-content/uploads/
           TCG_IWG_CEL_v1_r0p30_13feb2021.pdf  Section 4.3";
        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, ima-sig";
        }
        leaf filename-hint {
          type string;
          description
            "File that was measured";
        }
        leaf filedata-hash {
          type binary;
          description
            "Hash of filedata";
        }
        leaf filedata-hash-algorithm {
          type string;
          description
            "Algorithm used for filedata-hash";
        }
        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 pcr;
          description
            "Defines the PCR index that this event extended";
        }
        leaf signature {
          type binary;
          description
```

```
          "The file signature";
      }
    }

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

    grouping netequip-boot-event {
      description
        "Defines an hash log extend event for Network Equipment Boot.";
      leaf event-number {
        type uint64;
        description
          "Unique number for this event for sequencing";
      }
      leaf filename-hint {
        type string;
        description
          "File that was measured";
      }
      leaf filedata-hash {
        type binary;
        description
          "Hash of filedata";
      }
      leaf filedata-hash-algorithm {
        type string;
        description
          "Algorithm used for filedata-hash.";
        }
      leaf file-version {
        type string;
        description
          "File version information.";
      }
      leaf file-type {
        type string;
        description
          "Indicating at which boot stage the file is loaded,
           such as BIOS, BootLoader, etc.";
```

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

     grouping network-equipment-boot-event-log {
       description
         "Measurement log created by Network Equipment Boot.";
       list boot-event-entry {
         key event-number;
         description
           "Ordered list of Network Equipment Boot event logs
            by event-number.";
         uses netequip-boot-event;
       }
     }

     grouping event-logs {
       description
         "A selector for the log and its type.";
       choice attested_event_log_type {
         mandatory true;
         description
           "Event log type determines the event logs content.";
         case bios {
           if-feature "bios";
           description
             "BIOS/UEFI event logs";
           container bios-event-logs {
             description
               "BIOS/UEFI event logs";
             uses bios-event-log;
           }
         }
         case ima {
           if-feature "ima";
           description
             "IMA event logs.";
           container ima-event-logs {
             description
               "IMA event logs.";
             uses ima-event-log;
           }
         }
         case netequip_boot {
```

```
          if-feature "netequip_boot";
          description
            "Network Equipment Boot event logs";
          container boot-event-logs {
            description
              "Network equipment boot event logs.";
            uses network-equipment-boot-event-log;
          }
        }
      }
    }

    /**********************/
    /*   RPC operations   */
    /**********************/

    rpc tpm12-challenge-response-attestation {
      if-feature "taa:TPM12";
      description
        "This RPC accepts the input for TSS TPM 1.2 commands made to the
         attesting device.";
      input {
        container tpm12-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;
          leaf-list certificate-name {
            if-feature "tpm:TPMs";
            type certificate-name-ref;
            must "/tpm:rats-support-structures/tpm:tpms" +
                 "/tpm:tpm[tpm:tpm-firmware-version='taa:tpm12']" +
                 "/tpm:certificates/" +
                 "/tpm:certificate[certificate-name-ref=current()]" {
              error-message "Not an available TPM1.2 AIK certificate.";
            }
            description
              "When populated, the RPC will only get a Quote for the
              TPMs associated with these certificate(s).";
          }
        }
      }
      output {
        list tpm12-attestation-response {
          unique "certificate-name";
```

```
          description
            "The binary output of TPM 1.2 TPM_Quote/TPM_Quote2, including
             the PCR selection and other associated attestation evidence
             metadata";
          uses certificate-name-ref {
            description
              "Certificate associated with this tpm12-attestation.";
          }
          uses tpm12-attestation;
        }
      }
    }

    rpc tpm20-challenge-response-attestation {
      if-feature "taa:TPM20";
      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 nonce;
          uses tpm20-pcr-selection;
          leaf-list certificate-name {
            if-feature "tpm:TPMs";
            type certificate-name-ref;
            must "/tpm:rats-support-structures/tpm:tpms" +
                 "/tpm:tpm[tpm:tpm-firmware-version='taa:tpm20']" +
                 "/tpm:certificates/" +
                 "/tpm:certificate[certificate-name-ref=current()]" {
              error-message "Not an available TPM2.0 AIK certificate.";
            }
            description
              "When populated, the RPC will only get a Quote for the
              TPMs associated with the certificates.";
          }
        }
      }
      output {
        list tpm20-attestation-response {
          unique "certificate-name";
          description
```

```
             "The binary output of TPM2b_Quote in one TPM chip of the
              node which identified by node-id. An TPMS_ATTEST structure
              including a length, encapsulated in a signature";
           uses certificate-name-ref {
             description
               "Certificate associated with this tpm20-attestation.";
           }
           uses tpm20-attestation;
         }
       }
     }

     rpc log-retrieval {
       description
         "Logs Entries are either identified via indices or via providing
          the last line received. The number of lines returned can be
          limited. The type of log is a choice that can be augmented.";
       input {
         list log-selector {
           description
             "Selection of log entries to be reported.";
           uses tpm-name-selector;
           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 last-index-number {
                 type uint64;
                 description
                   "The last 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.";
             }
           }
           leaf log-entry-quantity {
             type uint16;
             description
               "The number of log entries to be returned. If omitted, it
                means all of them.";
           }
         }
         uses log-identifier;
       }

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

     /**************************************/
     /*   Config & Oper accessible nodes   */
```

```
     /**************************************/

     container rats-support-structures {
       description
         "The datastore definition enabling verifiers or relying
          parties to discover the information necessary to use the
          remote attestation RPCs appropriately.";
       container compute-nodes {
         if-feature "tpm:TPMs";
         description
           "Holds the set device subsystems/components in this composite
            device that support TPM operations.";
         list compute-node {
           key node-id;
           config false;
           min-elements 2;
           description
             "A component within this composite device which
              supports TPM operations.";
           leaf node-id {
             type string;
             description
               "ID of the compute node, such as Board Serial Number.";
           }
           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";
           }
           leaf node-name {
             type string;
             description
               "Name of the compute node.";
           }
           leaf node-location {
             type string;
             description
               "Location of the compute node, such as slot number.";
           }
         }
       }
       container tpms {
```

```
        description
          "Holds the set of TPMs within an Attester.";
        list tpm {
          key tpm-name;
          unique "tpm-path";
            description
             "A list of TPMs in this composite device that RATS
              can be conducted with.";
          uses tpm-name;
          leaf hardware-based {
            type boolean;
            config false;
            description
              "Answers the question: is this TPM is a hardware based
               TPM?";
          }
          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";
          }
          leaf tpm-path {
            type string;
            config false;
            description
              "Path to a unique TPM on a device.  This can change across
               reboots.";
          }
          leaf compute-node {
            if-feature "tpm:TPMs";
            type compute-node-ref;
            config false;
            mandatory true;
            description
              "Indicates the compute node measured by this TPM.";
          }
          leaf tpm-manufacturer {
            type string;
            config false;
            description
              "TPM manufacturer name.";
          }
```

```
          leaf tpm-firmware-version {
            type identityref {
              base taa:cryptoprocessor;
            }
            mandatory true;
            description
              "Identifies the cryptoprocessor API set supported.  This
               is automatically configured by the device and should not
               be changed.";
          }
          uses TPM12-hash-algo {
            when "tpm-firmware-version = 'taa:tpm12'";
            refine TPM12-hash-algo {
              description
                "The hash algorithm overwrites the default used for PCRs
                 on this TPM1.2 compliant cryptoprocessor.";
            }
          }
          leaf-list TPM12-pcrs {
            when "../tpm-firmware-version = 'taa:tpm12'";
            type pcr;
            description
              "The PCRs which may be extracted from this TPM1.2
               compliant cryptoprocessor.";
          }
          list tpm20-pcr-bank {
            when "../tpm-firmware-version = 'taa:tpm20'";
            key "TPM20-hash-algo";
            description
              "Specifies the list of PCRs that may be extracted for
               a specific Hash Algorithm on this TPM2 compliant
               cryptoprocessor.  A bank is a set of PCRs which are
               extended using a particular hash algorithm.";
            reference
              "https://www.trustedcomputinggroup.org/wp-content/uploads/
               TPM-Rev-2.0-Part-2-Structures-01.38.pdf  Section 10.9.7";
            leaf TPM20-hash-algo {
              type identityref {
                base taa:hash;
              }
              must "/tpm:rats-support-structures"
                 + "/tpm:attester-supported-algos"
                 + "/tpm:tpm20-hash" {
                error-message "Not a platform supported TPM20-hash-algo";
              }
              description
                "The hash scheme actively being used to hash a
                 one or more TPM2.0 PCRs.";
```

```
            }
            leaf-list pcr-index {
              type tpm:pcr;
              description
                "Defines what TPM2 PCRs are available to be extracted.";
            }
          }
          leaf tpm-status {
            type enumeration {
              enum operational {
                value 0;
                description
                  "The TPM currently is currently running normally and
                   is ready to accept and process TPM quotes.";
                reference
                  "TPM-Rev-2.0-Part-1-Architecture-01.07-2014-03-13.pdf
                   Section 12";
              }
              enum non-operational {
                value 1;
                description
                  "TPM is in a state such as startup or shutdown which
                   precludes the processing of TPM quotes.";
              }
            }
            config false;
            mandatory true;
            description
              "TPM chip self-test status.";
          }
          container certificates {
            description
              "The TPM's certificates, including EK certificates
               and AK certificates.";
            list certificate {
              key "certificate-name";
              description
                "Three types of certificates can be accessed via
                 this statement, including Initial Attestation
                 Key Cert, Local Attestation Key Cert or
                 Endorsement Key Cert.";
              leaf certificate-name {
                type string;
                description
                  "An arbitrary name uniquely identifying a certificate
                   associated within key within a TPM.";
              }
              leaf certificate-keystore-ref {
```

```
                type leafref {
                  path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
                     + "/ks:certificates/ks:certificate/ks:name";
                }
                description
                  "A reference to a specific certificate of an
                   asymmetric key in the Keystore.";
                   /* Note: It is also possible to import a grouping which
                      allows local definition via an imported keystore
                      schema. */
              }
              leaf certificate-type {
                type enumeration {
                  enum endorsement-cert {
                    value 0;
                    description
                      "Endorsement Key (EK) Certificate type.";
                  }
                  enum initial-attestation-cert {
                    value 1;
                    description
                      "Initial Attestation key (IAK) Certificate type.";
                  }
                  enum local-attestation-cert {
                    value 2;
                    description
                      "Local Attestation Key (LAK) Certificate type.";
                  }
                }
                description
                  "Type of this certificate";
              }
            }
          }
        }
      }
      container attester-supported-algos {
        description
          "Identifies which TPM algorithms are available for use on an
           attesting platform.";
        leaf-list tpm12-asymmetric-signing {
          when "../../tpm:tpms" +
               "/tpm:tpm[tpm:tpm-firmware-version='taa:tpm12']";
          if-feature "taa:TPM12";
          type identityref {
            base taa:asymmetric;
          }
          description
```

```
            "Platform Supported TPM12 asymmetric algorithms.";
        }
        leaf-list tpm12-hash {
          when "../../tpm:tpms" +
               "/tpm:tpm[tpm:tpm-firmware-version='taa:tpm12']";
          if-feature "taa:TPM12";
          type identityref {
            base taa:hash;
          }
          description
            "Platform supported TPM12 hash algorithms.";
        }
        leaf-list tpm20-asymmetric-signing {
          when "../../tpm:tpms" +
               "/tpm:tpm[tpm:tpm-firmware-version='taa:tpm20']";
          if-feature "taa:TPM20";
          type identityref {
            base taa:asymmetric;
          }
          description
            "Platform Supported TPM20 asymmetric algorithms.";
        }
        leaf-list tpm20-hash {
          when "../../tpm:tpms" +
               "/tpm:tpm[tpm:tpm-firmware-version='taa:tpm20']";
          if-feature "taa:TPM20";
          type identityref {
            base taa:hash;
          }
          description
            "Platform supported TPM20 hash algorithms.";
        }
      }
    }
  }
  <CODE ENDS>
```

2.1.2. ietf-tcg-algs

Cryptographic algorithm types were initially included within -v14 NETCONF's iana-crypto-types.yang. Unfortunately all this content including the algorithms needed here failed to make the -v15 used WGLC. As a result this document has encoded the TCG Algorithm definitions of [TCG-Algos], revision 1.32. By including this full table as a separate YANG file within this document, it is possible for other YANG models to leverage the contents of this model.

2.1.2.1. Features

There are two types of features supported <TPM12> and <TPM20>. Support for either of these features indicates that a cryptoprocessor supporting the corresponding type of TCG API is present on an Attester. Most commonly, only one type of cryptoprocessor will be available on an Attester.

2.1.2.2. Identities

There are three types of identities in this model.

The first are the cryptographic functions supportable by a TPM algorithm, these include: <asymmetric>, <symmetric>, <hash>, <signing>, <anonymous_signing>, <encryption_mode>, <method>, and <object_type>. The definitions of each of these are in Table 2 of [TCG-Algos].

The second are API specifications for tpms: <tpm12> and <tpm2>.

The third are specific algorithm types. Each algorithm type defines what cryptographic functions may be supported, and on which type of API specification. It is not required that an implementation of a specific TPM will support all algorithm types. The contents of each specific algorithm mirrors what is in Table 3 of [TCG-Algos].

2.1.2.3. YANG Module

```
<CODE BEGINS> file "ietf-tcg-algs@2020-09-18.yang"
module ietf-tcg-algs {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcg-algs";
  prefix taa;

  organization
    "IETF RATS Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/rats/>
     WG List:  <mailto:rats@ietf.org>
     Author:   Eric Voit <mailto:evoit@cisco.com>";

  description
    "This module defines a identities for asymmetric algorithms.

     Copyright (c) 2020 IETF Trust and the persons identified
     as authors of the code. All rights reserved.
     Redistribution and use in source and binary forms, with
```

```
       or without modification, is permitted pursuant to, and
       subject to the license terms contained in, the Simplified
       BSD License set forth in Section 4.c of the IETF Trust's
       Legal Provisions Relating to IETF Documents
       (https://trustee.ietf.org/license-info).
       This version of this YANG module is part of RFC XXXX
       (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
       itself for full legal notices.
       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 (RFC 2119)
       (RFC 8174) when, and only when, they appear in all
       capitals, as shown here.";

    revision 2020-09-18 {
      description
        "Initial version";
      reference
        "RFC XXXX: tbd";
    }

    /*****************/
    /*   Features    */
    /*****************/

    feature TPM12 {
      description
        "This feature indicates algorithm support for the TPM 1.2 API
         as per TPM-main-1.2-Rev94-part-2, Section 4.8.";
    }

    feature TPM20 {
      description
        "This feature indicates algorithm support for the TPM 2.0 API
         as per TPM-Rev-2.0-Part-1-Architecture-01.38 Section 11.4.";
    }


    /*****************/
    /*  Identities   */
    /*****************/

    /*  There needs to be collasping/verification of some of the identity
        types between the various algorithm types listed below */

    identity asymmetric {
      description
```

```
          "A TCG recognized asymmetric algorithm with a public and
           private key.";
        reference
          "http://trustedcomputinggroup.org/resource/tcg-algorithm-registry/
           TCG_Algorithm_Registry_r1p32_pub Table 2";
      }

      identity symmetric {
        description
          "A TCG recognized symmetric algorithm with only a private key.";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 2";
      }

      identity hash {
        description
          "A TCG recognized hash algorithm that compresses input data to
           a digest value or indicates a method that uses a hash.";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 2";
      }

      identity signing {
        description
          "A TCG recognized signing algorithm";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 2";
      }

      identity anonymous_signing {
        description
          "A TCG recognized anonymous signing algorithm.";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 2";
      }

      identity encryption_mode {
        description
          "A TCG recognized encryption mode.";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 2";
      }

      identity method {
        description
          "A TCG recognized method such as a mask generation function.";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 2";
```

```
     }

     identity object_type {
       description
         "A TCG recognized object type.";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 2";
     }

     identity cryptoprocessor {
       description
         "Base identity identifying a crytoprocessor.";
     }

     identity tpm12 {
       if-feature "TPM12";
       base cryptoprocessor;
       description
         "Supportable by a TPM1.2.";
       reference
         "TPM-Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf
          TPM_ALGORITHM_ID values, page 18";
     }

     identity tpm20 {
       if-feature "TPM12";
       base cryptoprocessor;
       description
         "Supportable by a TPM2.";
       reference
         "TPM-Rev-2.0-Part-2-Structures-01.38.pdf
          The TCG Algorithm Registry. Table 9";
     }

     identity TPM_ALG_RSA {
       if-feature "TPM12 or TPM20";
       base tpm12;
       base tpm20;
       base asymmetric;
       base object_type;
       description
         "RSA algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          RFC 8017. ALG_ID: 0x0001";
     }

     identity TPM_ALG_TDES {
```

```
       if-feature "TPM12";
       base tpm12;
       base symmetric;
       description
         "Block cipher with various key sizes (Triple Data Encryption
          Algorithm, commonly called Triple Data Encryption Standard)
          Note: was banned in TPM1.2 v94";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 18033-3. ALG_ID: 0x0003";
     }

     identity TPM_ALG_SHA1 {
       if-feature "TPM12 or TPM20";
       base hash;
       base tpm12;
       base tpm20;
       description
         "SHA1 algorithm - Deprecated due to insufficient cryptographic
          protection.  However it is still useful for hash algorithms
          where protection is not required.";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 10118-3. ALG_ID: 0x0004";
     }

     identity TPM_ALG_HMAC  {
       if-feature "TPM12 or TPM20";
       base tpm12;
       base tpm20;
       base hash;
       base signing;
       description
         "Hash Message Authentication Code (HMAC) algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3,
          ISO/IEC 9797-2 and RFC2014. ALG_ID: 0x0005";
     }

     identity TPM_ALG_AES  {
       if-feature "TPM12";
       base tpm12;
       base symmetric;
       description
         "The AES algorithm with various key sizes";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 18033-3. ALG_ID: 0x0006";
```

```
     }

     identity TPM_ALG_MGF1 {
       if-feature "TPM20";
       base tpm20;
       base hash;
       base method;
       description
         "hash-based mask-generation function";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3,
          IEEE Std 1363-2000 and IEEE Std 1363a -2004.
          ALG_ID: 0x0007";
     }

     identity TPM_ALG_KEYEDHASH {
       if-feature "TPM20";
       base tpm20;
       base hash;
       base object_type;
       description
         "An encryption or signing algorithm using a keyed hash.  These
          may use XOR for encryption or an HMAC for signing and may
          also refer to a data object that is neither signing nor
          encrypting.";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          TCG TPM 2.0 library specification. . ALG_ID: 0x0008";
       }

     identity TPM_ALG_XOR {
       if-feature "TPM12 or TPM20";
       base tpm12;
       base tpm20;
       base hash;
       base symmetric;
       description
         "The XOR encryption algorithm.";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          TCG TPM 2.0 library specification. ALG_ID: 0x000A";
     }

     identity TPM_ALG_SHA256 {
       if-feature "TPM20";
       base tpm20;
       base hash;
       description
```

```
         "The SHA 256 algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 10118-3. ALG_ID: 0x000B";
     }

     identity TPM_ALG_SHA384  {
       if-feature "TPM20";
       base tpm20;
       base hash;
       description
         "The SHA 384 algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 10118-3. ALG_ID: 0x000C";
     }

     identity TPM_ALG_SHA512  {
       if-feature "TPM20";
       base tpm20;
       base hash;
       description
         "The SHA 512 algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 10118-3. ALG_ID: 0x000D";
     }

     identity TPM_ALG_NULL {
       if-feature "TPM20";
       base tpm20;
       description
         "NULL algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          TCG TPM 2.0 library specification. ALG_ID: 0x0010";
     }

     identity TPM_ALG_SM3_256 {
       if-feature "TPM20";
       base tpm20;
       base hash;
       description
         "The SM3 hash algorithm.";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          GM/T 0004-2012 - SM3_256. ALG_ID: 0x0012";
     }
```

```
     identity TPM_ALG_SM4 {
       if-feature "TPM20";
       base tpm20;
       base symmetric;
       description
         "SM4 symmetric block cipher";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          GB/T 32907-2016. ALG_ID: 0x0013";
     }

     identity TPM_ALG_RSASSA {
       if-feature "TPM20";
       base tpm20;
       base asymmetric;
       base signing;
       description
         "Signature algorithm defined in section 8.2 (RSASSAPKCS1-v1_5)";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and RFC 8017.
          ALG_ID: 0x0014";
     }

     identity TPM_ALG_RSAES {
       if-feature "TPM20";
       base tpm20;
       base asymmetric;
       base encryption_mode;
       description
         "Signature algorithm defined in section 7.2 (RSAES-PKCS1-v1_5)";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and RFC 8017
          ALG_ID: 0x0015";
     }

     identity TPM_ALG_RSAPSS {
       if-feature "TPM20";
       base tpm20;
       base asymmetric;
       base signing;
       description
         "Padding algorithm defined in section 8.1 (RSASSA PSS)";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and RFC 8017.
          ALG_ID: 0x0016";
     }

     identity TPM_ALG_OAEP {
```

```
         if-feature "TPM20";
         base tpm20;
         base asymmetric;
         base encryption_mode;
         description
           "Padding algorithm defined in section 7.1 (RSASSA OAEP)";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and RFC 8017.
            ALG_ID: 0x0017";
       }

       identity TPM_ALG_ECDSA {
         if-feature "TPM20";
         base tpm20;
         base asymmetric;
         base signing;
         description
           "Signature algorithm using elliptic curve cryptography (ECC)";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            ISO/IEC 14888-3. ALG_ID: 0x0018";
       }

       identity TPM_ALG_ECDH {
         if-feature "TPM20";
         base tpm20;
         base asymmetric;
         base method;
         description
           "Secret sharing using ECC";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            NIST SP800-56A and RFC 7748. ALG_ID: 0x0019";
       }

       identity TPM_ALG_ECDAA {
         if-feature "TPM20";
         base tpm20;
         base asymmetric;
         base signing;
         base anonymous_signing;
         description
           "Elliptic-curve based anonymous signing scheme";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            TCG TPM 2.0 library specification. ALG_ID: 0x001A";
       }
```

```
    identity TPM_ALG_SM2 {
      if-feature "TPM20";
      base tpm20;
      base asymmetric;
      base signing;
      base encryption_mode;
      base method;
      description
        "SM2 - depending on context, either an elliptic-curve based,
         signature algorithm, an encryption scheme, or a key exchange
         protocol";
      reference
        "TCG_Algorithm_Registry_r1p32_pub Table 3 and
         A GM/T 0003.1-2012, GM/T 0003.2-2012, GM/T 0003.3-2012,
         GM/T 0003.5-2012 SM2. ALG_ID: 0x001B";
    }

    identity TPM_ALG_ECSCHNORR {
      if-feature "TPM20";
      base tpm20;
      base asymmetric;
      base signing;
      description
        "Elliptic-curve based Schnorr signature";
      reference
        "TCG_Algorithm_Registry_r1p32_pub Table 3 and
         TCG TPM 2.0 library specification. ALG_ID: 0x001C";
    }

    identity TPM_ALG_ECMQV {
      if-feature "TPM20";
      base tpm20;
      base asymmetric;
      base method;
      description
        "Two-phase elliptic-curve key";
      reference
        "TCG_Algorithm_Registry_r1p32_pub Table 3 and
         NIST SP800-56A. ALG_ID: 0x001D";
    }

    identity TPM_ALG_KDF1_SP800_56A {
      if-feature "TPM20";
      base tpm20;
      base hash;
      base method;
      description
        "Concatenation key derivation function";
```

```
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            NIST SP800-56A  (approved alternative1) section 5.8.1.
            ALG_ID: 0x0020";
       }

       identity TPM_ALG_KDF2  {
         if-feature "TPM20";
         base tpm20;
         base hash;
         base method;
         description
           "Key derivation function";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            IEEE 1363a-2004 KDF2 section 13.2. ALG_ID: 0x0021";
       }

       identity TPM_ALG_KDF1_SP800_108 {
         base TPM_ALG_KDF2;
         description
           "A key derivation method";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            NIST SP800-108 - Section 5.1 KDF. ALG_ID: 0x0022";
       }

       identity TPM_ALG_ECC {
         if-feature "TPM20";
         base tpm20;
         base asymmetric;
         base object_type;
         description
           "Prime field ECC";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            ISO/IEC 15946-1. ALG_ID: 0x0023";
       }

       identity TPM_ALG_SYMCIPHER {
         if-feature "TPM20";
         base tpm20;
         description
           "Object type for a symmetric block cipher";
         reference
           "TCG_Algorithm_Registry_r1p32_pub Table 3 and
            TCG TPM 2.0 library specification. ALG_ID: 0x0025";
       }
```

```
     identity TPM_ALG_CAMELLIA {
       if-feature "TPM20";
       base tpm20;
       base symmetric;
       description
         "The Camellia algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 18033-3. ALG_ID: 0x0026";
     }

     identity TPM_ALG_SHA3_256  {
       if-feature "TPM20";
       base tpm20;
       base hash;
       description
         "ISO/IEC 10118-3 - the SHA 256 algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          NIST PUB FIPS 202. ALG_ID: 0x0027";
     }

     identity TPM_ALG_SHA3_384 {
       if-feature "TPM20";
       base tpm20;
       base hash;
       description
         "The SHA 384 algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          NIST PUB FIPS 202. ALG_ID: 0x0028";
     }

     identity TPM_ALG_SHA3_512 {
       if-feature "TPM20";
       base tpm20;
       base hash;
       description
         "The SHA 512 algorithm";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          NIST PUB FIPS 202. ALG_ID: 0x0029";
     }

     identity TPM_ALG_CMAC {
       if-feature "TPM20";
       base tpm20;
       base symmetric;
```

```
       base signing;
       description
         "block Cipher-based Message Authentication Code (CMAC)";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 9797-1:2011 Algorithm 5. ALG_ID: 0x003F";
     }

     identity TPM_ALG_CTR {
       if-feature "TPM20";
       base tpm20;
       base symmetric;
       base encryption_mode;
       description
         "Counter mode";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 10116. ALG_ID: 0x0040";
     }

     identity TPM_ALG_OFB {
       base tpm20;
       base symmetric;
       base encryption_mode;
       description
         "Output Feedback mode";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 10116. ALG_ID: 0x0041";
     }

     identity TPM_ALG_CBC {
       if-feature "TPM20";
       base tpm20;
       base symmetric;
       base encryption_mode;
       description
         "Cipher Block Chaining mode";
       reference
         "TCG_Algorithm_Registry_r1p32_pub Table 3 and
          ISO/IEC 10116. ALG_ID: 0x0042";
     }

     identity TPM_ALG_CFB {
       if-feature "TPM20";
       base tpm20;
       base symmetric;
       base encryption_mode;
```

```
        description
          "Cipher Feedback mode";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           ISO/IEC 10116. ALG_ID: 0x0043";
      }

      identity TPM_ALG_ECB  {
        if-feature "TPM20";
        base tpm20;
        base symmetric;
        base encryption_mode;
        description
          "Electronic Codebook mode";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           ISO/IEC 10116. ALG_ID: 0x0044";
      }

      identity TPM_ALG_CCM  {
        if-feature "TPM20";
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
          "Counter with Cipher Block Chaining-Message Authentication
          Code (CCM)";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           NIST SP800-38C. ALG_ID: 0x0050";
      }

      identity TPM_ALG_GCM {
        if-feature "TPM20";
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
          "Galois/Counter Mode (GCM)";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           NIST SP800-38D. ALG_ID: 0x0051";
      }

      identity TPM_ALG_KW {
        if-feature "TPM20";
```

```
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
          "AES Key Wrap (KW)";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           NIST SP800-38F. ALG_ID: 0x0052";
      }

      identity TPM_ALG_KWP {
        if-feature "TPM20";
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
          "AES Key Wrap with Padding (KWP)";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           NIST SP800-38F. ALG_ID: 0x0053";
      }

      identity TPM_ALG_EAX  {
        if-feature "TPM20";
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
          "Authenticated-Encryption Mode";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           NIST SP800-38F. ALG_ID: 0x0054";
      }

      identity TPM_ALG_EDDSA {
        if-feature "TPM20";
        base tpm20;
        base asymmetric;
        base signing;
        description
          "Edwards-curve Digital Signature Algorithm (PureEdDSA)";
        reference
          "TCG_Algorithm_Registry_r1p32_pub Table 3 and
           RFC 8032. ALG_ID: 0x0060";
      }
```

```
}
<CODE ENDS>
```

Note that not all cryptographic functions are required for use by ietf-tpm-remote-attestation.yang. However the full definition of Table 3 of [TCG-Algos] will allow use by additional YANG specifications.

3. IANA considerations

This document will include requests to IANA:

To be defined yet. But keeping up with changes to ietf-tcg-algs.yang will be necessary.

4. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

Container: </rats-support-structures/attester-supported-algos>

* <tpm12-asymmetric-signing>, <tpm12-hash>, <tpm20-asymmetric-signing>, and <tpm20-hash> all could be populated with algorithms which are not supported by the underlying physical TPM installed by the equipment vendor.

Container: </rats-support-structures/tpms>

* <tpm-name> - Although shown as 'rw', it is system generated

* <tpm20-pcr-bank> - It is possible to configure PCRs for extraction which are not being extended by system software. This could unnecessarily use TPM resources.

* <certificates> - It is possible to provision a certificate which does not correspond to a Attestation Identity Key (AIK) within the TPM.

RPC: <tpm12-challenge-response-attestation> - Need to verify that the certificate is for an active AIK.

RPC: <tpm20-challenge-response-attestation> - Need to verify that the certificate is for an active AIK.

RPC: <log-retrieval> - Pulling lots of logs can chew up system resources.

5. Acknowledgements

Not yet.

6. Change Log

Changes from version 04 to version 05:

* YANG Dr comments covered

Changes from version 03 to version 04:

* TPM1.2 Quote1 eliminated

* YANG model simplifications so redundant info isn't exposed

Changes from version 02 to version 03:

* moved to tcg-algs

* cleaned up model to eliminate sources of errors

* removed key establishment RPC

* added lots of XPATH which must all be scrubbed still

* Descriptive text added on model contents.

Changes from version 01 to version 02:

* Extracted Crypto-types into a separate YANG file

* Mades the algorithms explicit, not strings

* Hash Algo as key the selected TPM2 PCRs

* PCR numbers are their own type

* Eliminated nested keys for node-id plus tpm-name

* Eliminated TPM-Name of "ALL"

* Added TPM-Path

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.ietf-netconf-keystore]
Watsen, K., "A YANG Data Model for a Keystore", Work in Progress, Internet-Draft, draft-ietf-netconf-keystore-21, 10 February 2021, <https://www.ietf.org/archive/id/draft-ietf-netconf-keystore-21.txt>.

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-11, 30 March 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-11.txt>.

[I-D.ietf-rats-reference-interaction-models]
Birkholz, H., Eckel, M., Newton, C., and L. Chen, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-ietf-rats-reference-interaction-models-01, 23 October 2020, <https://www.ietf.org/archive/id/draft-ietf-rats-reference-interaction-models-01.txt>.

[I-D.ietf-rats-tpm-based-network-device-attest]
Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", Work in Progress, Internet-Draft, draft-ietf-rats-tpm-based-network-device-attest-06, 7 December 2020, <https://www.ietf.org/archive/id/draft-ietf-rats-tpm-based-network-device-attest-06.txt>.

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

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <https://www.rfc-editor.org/info/rfc8348>.

[TCG-Algos]
"TCG_Algorithm_Registry_r1p32_pub", n.d., <hhttp://trustedcomputinggroup.org/resource/tcg-algorithm-registry/>.

[TPM1.2] TCG, ., "TPM 1.2 Main Specification", 2 October 2003, <https://trustedcomputinggroup.org/resource/tpm-main-specification/>.

[TPM2.0] TCG, ., "TPM 2.0 Library Specification", 15 March 2013, <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

7.2. Informative References

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

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

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Michael Eckel
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: michael.eckel@sit.fraunhofer.de

Shwetha Bhandari
ThoughtSpot

Email: shwetha.bhandari@thoughtspot.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Bill Sulzen
Cisco Systems

Email: bsulzen@cisco.com

Liang Xia (Frank)
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing
Jiangsu, 210012
China

Email: Frank.Xialiang@huawei.com

Tom Laffey
Hewlett Packard Enterprise

Email: tom.laffey@hpe.com

Guy C. Fedorkow
Juniper Networks
10 Technology Park Drive
Westford

Email: gfedorkow@juniper.net

IETF K. Moriarty
Internet-Draft Center for Internet Security (CIS)
Intended status: Standards Track May 4, 2021
Expires: November 5, 2021

Scalable Remote Attestation for Systems, Containers, and Applications
draft-moriarty-attestationsets-02

Abstract

This document establishes an architectural pattern whereby a remote attestation could be issued for a complete set of benchmarks or controls that are defined and grouped by an external entity, preventing the need to send over individual attestations for each item within a benchmark or control framework. This document establishes a pattern to list sets of benchmarks and controls within CWT and JWT formats for use as an Entity Attestation Token (EAT).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Posture assessment has long been desired, but has been difficult to achieve due to complexities of customization requirements at each organization. By using policy and measurement sets that may be offered at various assurance levels, automating posture assessment through attestation becomes achievable for organizations of all sizes. The measurement and policy groupings may be provided by the vendor or by a neutral third party to enable ease of use and consistent implementations. This provides simpler options to enable posture assessment at selected levels by organizations without the need to have in-house expertise. The measurement and policy sets may also be customized, but not necessary to achieve posture assessment to predefined options. This document describes a method to use existing attestation formats and protocols while allowing for profiles of policies, benchmarks, and measurements at defined assurance levels that scale to provide transparency to posture assessment results with remote attestation.

By way of example, the Center for Internet Security (CIS) hosts recommended configuration settings to secure operating systems, applications, and devices in CIS Benchmarks developed with industry experts. Attestations aligned to the CIS Benchmarks or other configuration guide such as a DISA STIG could be used to assert the configuration meets expectations. This has already been done for multiple platforms to demonstrate assurance for firmware according to NIST SP 800-193, Firmware Resiliency Guidelines. In order to scale remote attestation, a single attestation for a set of Benchmarks or policies being met may be sent to the remote atteststaion management system.

On traditional servers, assurance to NIST SP 800-193 is provable through attestation from a root of trust (RoT), using the Trusted Computing Group (TCG) Trusted Platform Module (TPM) chip and attestation formats. At boot, policy and measurement expectations are verified against a set of "golden policies" from collected and attested evidence. Device identity and measurements can also be attestated at runtime. The attestations on evidence (e.g. hash of boot element) and verification of attestations are typically contained within a system and are limited to the control plane for management. The policy and measurement sets for comparison are protected to assure the result in the attestation verification process for boot element. Event logs and PCR values may be exposed to provide transparency into the verified attestations. Remote attestation on systems is intended to provide an assessment of posture for all managed systems and across various layers in each of these systems in an environment.

There is a balance of exposure and evidence needed to assess posture when providing assurance of controls and system state. Currently, logs and TPM PCR values may be passed to provide assurance of verification of attestation evidence meeting set requirements. Providing the assurance can be accomplished with a remote attestation format such as the Entity Attestation Token (EAT) [I-D.ietf-rats-eat] and a RESTful interface such as ROLIE or RedFish. Policy definition blocks may be scoped to control measurement sets, where the EAT asserts compliance to the policy or measurement block specified and may include claims with the log and PCR value evidence. Measurement and Policy sets may be published and maintained by separate entities (e.g. CIS Benchmarks, DISA STIGs). The policy and measurement sets should be maintained separately even if associated with the same benchmark or control set. This avoids the need to transition the verifying entity to a remote system for individual policy and measurements which are performed locally for more immediate remediation as well as other functions.

Examples of measurement and policy sets include, but are not limited to:

- Hardware attribute certificates

- Hardware Attribute Certificate Comparison Results

- Reference Integrity Measurements for firmware

- Operating system benchmarks at Specified Assurance Levels o Application hardening Benchmarks at Specified Assurance Levels

o Container security benchmarks at Specified Assurance Levels

Scale, ease of use, full automation, and consistency for customer consumption of a remote attestation function or service are essential toward the goal of consistently securing systems against known threats and vulnerabilities. Mitigations may be baked into policy. Measurement verification sets and the attestation that the sets meet expected policies and measurements are conveyed in an Entity Attestation Token made available to a RESTful interface in aggregate for the systems managed.

2. Policy and Measurement Set Definitions

This document defines EAT claims in the JWT [RFC7519] and CWT [RFC8392] registries to provide attestation to a set of verified claims within a defined grouping. The trustworthiness will be conveyed on original verified evidence as well as the attestation on the grouping.

{

Claim	Long Name	Description	Format
MPS	Measurement or Policy Set	Name for the MPS	
LEM	Log Evidence of MPS	Log File or URI	
PCR	TPM PCR Values		
FMA	Format of MPS Attestations	Format of included attestations	
HSH	Hash Value/Message Digest	Hash va,ue of configuration set	

}

3. Supportability and Re-Attestation

The remote attestation framework shall include provisions within the system and attestation authority to allow for Product modification.

Over its lifecycle, the Product may experience modification due to: maintenance, failures, upgrades, expansion, moves, etc..

The customer can chose to:

o Run remote attestation after product modification, or

o Not take action and remain un-protected

In the case of Re-Attestation:

- o framework needs to invalidate previous TPM PCR values and tokens,
- o framework needs to collect new measurements,
- o framework needs to maintain history or allow for history to be logged to enable change traceability attestation, and
- o framework needs to notify that the previous attestation has been invalidated

4. Configuration Sets

In some cases, it may be difficult to attest to configuration settings for the initial or subsequent attestation and verification processes. The use of an expected hash value for configuration settings can be used to compare the attested configuration set. In this case, the creator of the attestation verification measurements would define a set of values for which a message digest would be created and then signed by the attestor. The expected measurements would include the expected hash value for comparison. The configuration set could be the full attestation set to a Benchmark or a defined subset.

5. Remediation

If policy and configration settings or measurements attestated do not meet expected values, remediation is desireable. Automated remediation performed with alignment to zero trust architecture principles would require that the remeidation be performed prior to any relying component executing. The relying component would verifiy before continuing in a zero trust architecture.

Ideally, remediation would occur on system as part of the process to attest to a set of attestations, similar to how attestation is performed for firmware in the boot process. If automated remediation is not possible, an alert should be generated to allow for notification of the variance from expected values.

6. Security Considerations

This document establishes a pattern to list sets of benchmarks and controls within CWT and JWT formats. The contents of the benchmarks and controls are out of scope for this document. This establishes an architectural pattern whereby a remote attestation could be issued for a complete set of benchmarks or controls as defined and grouped by external entities, preventing the need to send over individual attestations for each item within a benchmark or control framework.

This document does not add security consideration over what has been described in the EAT, JWT, or CWT specifications.

7. IANA Considerations

This memo includes no request to IANA, yet. This will list the initial registration sets to the JWT and CWT registries if adopted.

8. Contributors

Thank you to reviewers and contributors who helped to improve this document. Thank you to Nick Grobelney, Dell Technologies, for your review and contribution to separate out the policy and measurement sets. Thank you, Samant Kakarla and Huijun Xie from Dell Technologies, for your detailed review and corrections on boot process details. Section 3 has been contributed by Rudy Bauer from Dell as well and an author will be added on the next reveision.

9. References

9.1. Normative References

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

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

9.2. Informative References

[I-D.ietf-rats-eat] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-09 (work in progress), March 2021.

Appendix A. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

Appendix B. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

Author's Address

Kathleen M. Moriarty
Center for Internet Security (CIS)
31 Tech Valley Drive
East Greenbush, NY
US

EMail: Kathleen.Moriarty.ietf@gmail.com