RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2021

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

July 13, 2020

Time-Based Uni-Directional Attestation
draft-birkholz-rats-tuda-03

Abstract

This documents defines the method and bindings used to conduct Time-based Uni-Directional Attestation (TUDA) between two RATS (Remote ATtestation procedureS) entities over the Internet. TUDA does not require a challenge-response handshake and thereby does not rely on the conveyance of a nonce to prove freshness of remote attestation Evidence. Conversely, TUDA enables the creation of Secure Audit Logs that can constitute Evidence about current and past operational states of an Attester. Every RATS entity requires access to a trustable and synchronized time-source. A Handle Distributor takes on the corresponding role of a Time Stamp Authority (TSA) to provide Time Stamp Tokens (TST) to all RATS entities.

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 communication partner - the Attester - over the Internet to another communication partner - the Verifier - without direct access. TUDA uses the architectural constituents of the RATS Architecture [I-D.ietf-rats-architecture] that defines the roles Attester and Verifier in detail. The RATS Architecture also defines Conceptual Messages conveyed between these roles.

TUDA creates and conveys a specific type of Conceptual Message called Evidence: an integrity protected and tamper-evident composition of trustworthiness Claims that are collected from an Attester's Target Environments (defined in section 4.2. of [I-D.ietf-rats-architecture]). Evidence is consumed and appraised by a Verifier in order to eventually create Attestation Results.

Attestation Results are the output of RATS to support security decisions made by Relying Parties.

In contrast to traditional bi-directional challenge-response protocols (as defined in section 8.1 of [I-D.birkholz-rats-reference-interaction-model]), TUDA enables a uni-directional conveyance of attestation Evidence that allows for providing attestation information without solicitation (as defined in section 8.2 of [I-D.birkholz-rats-reference-interaction-model]). Exemplary applications of TUDA protocol family are the creation of beacons in vehicular environments ([IEEE1609]) or the notification content of event streams created by YANG Push ([RFC8641], [RFC8640], [RFC8639]).

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.birkholz-rats-reference-interaction-model]. 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. FA is achieved by timely recording of assessments of the authenticity from Target Environments (via "audit logs" during "audit sessions") that are authorized for this purpose and trustworthy (e.g via endorsed Roots of Trusts), 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. Handle Distributor

A trusted binding of a global timescale with Evidence is enabled in TUDA via a third party: the Handle Distributor. A Handle Distributor can provide centrally generated nonces, signed timestamps, or other handles used as qualifying data in Evidence generation. In TUDA, both the Attester and the Verifier receive signed timestamps from the Handle Distributor. These trusted timestamps replace nonces in Evidence generation and Evidence appraisal as specified in the remainder of this document.

1.3. Terminology

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

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. Evidence in TUDA

Remote attestation Evidence is a set of trustworthiness claims (assertions about the Target Environments of an Attester) that are accompanied by a proof of their veracity - typically a signature based on shielded, private and potentially restricted key material (used as an Authentication Secret as specified in section 6 of [I-D.birkholz-rats-reference-interaction-model]). As key material alone is typically not self-descriptive with respect to its intended use (its semantics), the Evidence created via TUDA is accompanied by two kinds of certificates that are cryptographically associated with a Trust Anchor (TA) [RFC4949] via a certification path:

- o 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 creates Evidence, and

- o an Endorsement Key (EK) Certificate (EK-Cert) that represents the protection characteristics 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 assertions about the system characteristics describing the Attester's Target Environments - the Evidence created via TUDA by the Attester is

considered trustable and believable. Ultimately, all trustable and believable Evidence is appraised by a Verifier in order to assess the trustworthiness of the corresponding Attester.

3. Generating Evidence about Software Component Integrity

The TUDA protocol family uses hash values of all started software components as a basis to provide and create Evidence about the integrity of the software components of an Attester's Target Environments (see section 4.2. in [I-D.ietf-rats-architecture]). This concept is implemented, for example, in the Linux kernel and is called the Linux Integrity Measurement Architecture (IMA) [Safford]. Open source solutions, for example, based on [RFC5209] use IMA to enable remote attestation [Steffens]. This section defines the processed data items, the requirements for system components, and corresponding operations to enable the generation of Evidence about software component integrity for TUDA.

3.1. 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 represent Claims that typically occur in large quantities and are referred to as a measurements. The Linux IMA feature is one way to generate these measurements on an Attester. Measurements are chained by Attesters using a rolling hash function. Each measurement added to the sequence of all measurements results in a new current hash value. The most recent (i.e., up-to-date) output of the rolling hash function that is chaining the created measurements is referred to as a digest.

3.2. Attesting Environments and Roots of Trust

The primitive operations used to generate the 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, Evidence based on values generated by the RTM is considered invalid. RTMs MUST be accessible to the first Attesting Environment in Layered Attestation (see section 4.3. in [I-D.ietf-rats-architecture]). A RTM MAY retain measurements until the first RTS becomes available in a Layered Attestation procedure.

The primitive operations used to store and chain measurements via a rolling hash function MUST be provided by Roots of Trust for Storage (RTS) that are system components 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 is considered invalid. RTS MUST be accessible to all Attesting Environments that are chained in a Layered Attestation procedure.

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 is considered invalid. All Attesting Environments of an Attester MUST be RTRs.

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. Analogously, a Trusted Platform Module (TPM, see [TPM12] and [TPM2]) takes on the roles of an RTS and an RTR. The specification in this document is based on the use of a TPM. The protocol part of this specification can also be used with other RTS and RTR as long as essential 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.

3.3. Primitive Operations for Remote Attestation

The primitive operation for processing measurements and adding them to a chain of measurements via a rolling hash function is called extend. Analogously, extend operations are provided by an RTS.

The primitive operation for retrieving the current available digest value that is the result of the rolling hash function including a signature based on an Attestation Key is called quote. Analogously, quote operations are provided by an RTR.

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

4. Remote Attestation Principles

In essence, the TUDA protocol family is composed of three operations that are defined in [I-D.birkholz-rats-reference-interaction-model]. These generic definitions align with the definitions presented in [PRIRA] and [TCGGLOSS].

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 Integrity Measurements (RIM).

4.1. Indeterministic Measurements

The sequence of measurements that is extended into the RTS may not be deterministic (e.g., due to parallelization of or race conditions between starting software components in user space). In order to enable appraisal in these case, 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.birkholz-rats-reference-interaction-model]. The Verifier has to replay the event log using its own extend operation with an identical rolling hash function in order to generate reference value as outlined in section 2.4.1. of [I-D.fedorkow-rats-network-device-attestation]. 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. This appraisal process requires Reference Integrity Measurements as are provided via [I-D.birkholz-rats-coswid-rim] or [TCGRIM]. Each RIM includes the information required to associate hash values of software components included in an event log with nominal reference hash values in the RIM. A Verifier requires an appropriate set of RIMs to compare every record in an event log successfully. The acquisition of RIMs and corresponding procedures are out-of-scope of this document.

4.2. Principles of Uni-Directional and Time-Based Attestation

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.birkholz-rats-reference-interaction-model] tightly couples the three operations of generating, conveying and appraising Evidence.

The Time-Based Uni-directional Attestation family of protocols can decouple these three operations. As a result, TUDA provides additional capabilities, such as:

- o remote attestation for Attesters that might not always be able to reach the Internet by enabling the appraisal of past states,

- o 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,

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

4.3. TUDA Attesting Environment Requirements

Evidence generation in TUDA requires certain features implemented by theTarget Environment

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

- o Platform Configuration Registers (PCR) that can extend measurements consecutively and represent the sequence of measurements as a single digest,

- o Restricted Signing Keys (RSK) that can only be accessed, if a specific signature about a set of measurements can be provided as authentication, and

- o a dedicated source of (relative) time, e.g. a tick counter (a tick being a specific time interval, for example 10 ms).

4.4. TUDA Handle Distributor: Time Stamp Authority

Both Evidence Generation and Evidence Appraisal of TUDA require a Handle Distributor that can take on the role of a trusted Time Stamp Authority (TSA) as an additional third party. This TUDA specification uses a Time Stamp Authority based on [RFC3161]. The combination of the local source of time provided by a RoT (located on the Attester) and the Time Stamp Tokens provided by the Handle Distributor (to both the Attester and the Verifier) enable an appropriate assertion of freshness.

4.5. Information Elements and Conveyance

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

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

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

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

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

4.6. TUDA Objectives

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

- increase the confidence in authentication and authorization procedures,

- address the requirements of constrained-node networks,

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

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

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

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

o provide trustworthy audit logs of past endpoint states.

4.7. Hardware Dependencies

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

5. TUDA Core Concept

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

In general, remote attestation using a bi-directional communication scheme includes sending a nonce-challenge within a signed Evidence token. Using the TPM 1.2 as an example, a corresponding nonce-challenge would be included within the signature created by the TPM_Quote command in order to prove the freshness of 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
- the typical round-trip time between the Verifier and the Attester.

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

- the time-synchronization between the Attester and the 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
- 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.birkholz-rats-reference-interaction-model] 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>.

[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>.

[I-D.birkholz-rats-coswid-rim]
Birkholz, H., Uiterwijk, P., Fitzgerald-McKay, J., and D. Waltermire, "Reference Integrity Measurement Extension for Concise Software Identities", draft-birkholz-rats-coswid-rim-00 (work in progress), March 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.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-05 (work in progress), July 2020.

[I-D.ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-15 (work in progress), May 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 Platform Module Library Specification, Family 2.0, Level 00, Revision 01.16 ed., Trusted Computing Group", 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    "202007130000Z" --  13 July 2020
        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 (2020)."

        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 (2017).";

  revision "2017-10-30" {
    description
```

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

     container tudaV1General {
     description
       "TBD";

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

           DEFVAL intentionally omitted - counter object.";
       }

       leaf tudaV1GeneralVersionInfo {
         type snmp-framework:SnmpAdminString {
           length "0..255";
         }
         config false;
```

```
      description
       "Version information for TUDA MIB, e.g., specific release
        version of TPM 1.2 base specification and release version
        of TPM 1.2 errata specification and manufacturer and model
        TPM module itself.";
    }
  }

  container tudaV1AIKCert {
  description
    "TBD";

    leaf tudaV1AIKCertCycles {
      type yang:counter32;
      config false;
      description
       "Count of AIK Certificate chain update cycles that have
        occurred.

        DEFVAL intentionally omitted - counter object.";
    }


    /* XXX table comments here XXX */

    list tudaV1AIKCertEntry {

      key "tudaV1AIKCertCycleIndex tudaV1AIKCertInstanceIndex
           tudaV1AIKCertFragmentIndex";
        config false;
      description
       "An entry for one fragment of AIK Certificate data.";


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

          DEFVAL intentionally omitted - index object.";
      }

      leaf tudaV1AIKCertInstanceIndex {
```

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

          DEFVAL intentionally omitted - index object.";
      }

      leaf tudaV1AIKCertFragmentIndex {
        type int32 {
          range "1..2147483647";
        }
        config false;
        description
         "Low-order index of this AIK Certificate fragment.

          DEFVAL intentionally omitted - index object.";
      }

      leaf tudaV1AIKCertData {
        type binary {
          length "0..1024";
        }
        config false;
        description
         "A fragment of CBOR encoded AIK Certificate data.";
      }
    }
  }

  container tudaV1TSACert {
  description
    "TBD";

    leaf tudaV1TSACertCycles {
      type yang:counter32;
      config false;
      description
       "Count of TSA Certificate chain update cycles that have
        occurred.

        DEFVAL intentionally omitted - counter object.";
    }
```

```
    /* XXX table comments here XXX */

    list tudaV1TSACertEntry {

      key "tudaV1TSACertCycleIndex tudaV1TSACertInstanceIndex
           tudaV1TSACertFragmentIndex";
      config false;
      description
       "An entry for one fragment of TSA Certificate data.";


      leaf tudaV1TSACertCycleIndex {
        type int32 {
          range "1..2147483647";
        }
        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:

- 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:

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

```
TUDA-RestrictionInfo = [Composite,
                        restrictedKey_Pub: Pubkey,
                        CertifyInfo]
```

```
PCRSelection = bytes .size (2..4) ; used as bit string

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

Pubkey = bytes ; may be extended to COSE pubkeys

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

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

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

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

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

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

Figure 6: TUDA-Key element in CDDL

Required TPM functions:

```
| 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:

- hash() : description TBD

- sig() : description TBD

- X.509-Certificate() : description TBD

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

D.3.1. AIK and AIK Certificate

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

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

```
TUDA-Cert = [AIK-Cert, TSA-Cert]; maybe split into two for SNMP
AIK-Certificate = X.509-Certificate(AIK-Key,Restricted-Flag)
TSA-Certificate = X.509-Certificate(TSA-Key, TSA-Flag)
```

Figure 15: TUDA-Cert element for TPM 2.0

D.3.2. Synchronization Token

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

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

Figure 16: TUDA-Sync element for TPM 2.0

D.3.3. Measurement Log

The creation procedure is identical to Appendix D.2.4.

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

Figure 17: TUDA-Log element for TPM 2.0

D.3.4. Explicit time-based Attestation

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

```
TUDA-AttestationToken = TUDA-AttestationToken_quote / TUDA-
AttestationToken_audit

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

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

Figure 18: TUDA-Attest element for TPM 2.0

D.3.5. Sync Proof

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

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

Figure 19: TUDA-Proof element for TPM 2.0

Acknowledgements

Authors' Addresses

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

Email: andreas.fuchs@sit.fraunhofer.de

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

Email: henk.birkholz@sit.fraunhofer.de

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

Email: blueroofmusic@gmail.com

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

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