Abstract

Remote ATtestation ProcedureS (RATS) architecture facilitates the attestation of device characteristics that, in general, are based on specific trustworthiness qualities intrinsic to a device or service. It includes trusted computing functionality provided by device hardware and software that allows trustworthiness qualities to be asserted and verified as part of, or pre-requisite to, the device’s normal operation. The RATS architecture maps corresponding attestation functions and capabilities to specific RATS Roles. The goal is to enable an appropriate conveyance of evidence about device trustworthiness via network protocols. RATS Roles provide the endpoint context for understanding the various interaction semantics of the attestation lifecycle. The RATS architecture provides the building block concepts, semantics, syntax and framework for interoperable attestation while remaining hardware-agnostic. This flexibility is intended to address a significant variety of use-cases and scenarios involving interoperable attestation. Example usages include, but are not limited to: financial transactions, voting machines, critical safety systems, network equipment health, or trustworthy end-user device management. Existing industry attestation efforts may be helpful toward informing RATS architecture. Such as: Remote Integrity VERification (RIVER), the creation of Entity Attestation Tokens (EAT), software integrity Measurement And ATtestation (MAAT).

Status of This Memo

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

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

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

This Internet-Draft will expire on September 13, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction ............................................. 3
   1.1. What is Remote Attestation ......................... 4
   1.2. The purpose of RATS Architecture and Terminology ... 4
   1.3. Requirements notation ............................. 4
2. RATS Architecture ......................................... 4
3. Architectural Components ................................. 5
   3.1. RATS Roles ........................................... 5
4. RATS Actors ................................................ 6
   4.1. RATS Duties .......................................... 7
       4.1.1. Attester Duties ............................... 8
       4.1.2. Verifier Duties ............................... 8
       4.1.3. Claimant Duties ............................... 8
       4.1.4. Relying Party Duties .......................... 8
       4.1.5. RATS Interactions ............................. 9
5. Application of RATS ........................................ 10
   5.1. Trust and Trustworthiness .......................... 11
   5.2. Claims and Evidence ............................... 12
   5.3. RATS Information Flows ............................. 12
6. Exemplary Composition of Roles ............................ 13
   6.1. Conveyance of Trusted Claim Sets Validated by Signature .... 13
   6.2. Conveyance of Attestation Evidence Appraised by a
1. Introduction

In general, this document provides normative guidance how to use, create or adopt network protocols that facilitate remote attestation procedures. The RATS Architecture anticipates broad deployment contexts that range from IoT to Cloud and Edge ecosystems. The foundation of the RATS architecture is the specification of RATS Roles that can be chained via RATS Interactions and - as a result - may be composed into use-case specific Remote Attestation Procedures. RATS Actors establish an ecosystem neutral context where RATS Roles are hosted and where a variety of Remote Attestation Procedure interactions are defined independent of specific conveyance protocols or message formats. In summary, the goal of the RATS Architecture is to enable interoperable interaction between the RATS Roles. Hence, the RATS Architecture is designed to enable interoperability via well-defined semantics of the information model (attestation assertions/claims), associated with RATS Roles following a conveyance
model (RATS Interactions) that may be used to compose domain-specific remote attestation solutions.

1.1. What is Remote Attestation

Unfortunately, the term Attestation itself is an overloaded term. In consequence, the term Remote Attestation covers a spectrum of meanings. The common denominator encompasses the creation, conveyance, and appraisal of evidence pertaining to the trustworthiness characteristics of the creator of the evidence. In essence, RATS are used to enable the assessment of the trustworthiness of a communication partner.

1.2. The purpose of RATS Architecture and Terminology

To consolidate the utilization of existing and emerging network protocols in the context of RATS, this document provides a detailed definition of Attestation Terminology that enables interoperability between different types of RATS. Specifically, this document illustrates and remediates the impedance mismatch of terms related to Remote Attestation Procedures used in different domains today. As an additional contribution, new terms defined by this document provide a common basis that simplifies future work on RATS in the IETF and beyond.

1.3. Requirements notation

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

2. RATS Architecture

One of the goals of the RATS Architecture is to provide the building blocks - the roles defined by the RATS Architecture - to enable the composition of service-chains/hierarchies and work-flows that can create and appraise evidence about the trustworthiness of devices and services.

The RATS Architecture is based on the use-cases defined in [I-D.richardson-rats-usecases].

The RATS architecture specifies:

- The building blocks to create remote attestation procedures applicable Actors, Roles, Duties, and Interactions,
Mandatory and optional trust relationships between its Roles, that may assume a Root-of-Trust context,

The interaction between Roles that reside on separate Actors and interact via network protocols,

Protocol/message framing that allows for well-defined and opaque payloads,

The means to prove, preserve and convey trust properties, such as identity, varacity, freshness, or provenance, and

Primitives necessary for the construction of interoperable attestation payloads.

3. Architectural Components

The basic architectural components defined in this document are:

- RATS Roles
- RATS Actors
- RATS Duties
- RATS Interactions

The following sub-section define and elaborate on these terms:

3.1. RATS Roles

A Role in the context of usage scenarios for remote attestation procedures is providing a service to other Roles. Roles are building blocks that can be providers and consumers of information. In the RATS architecture, devices or services can take on RATS roles. They are composites of internal functions (RATS Duties) and external functions (RATS Interactions) that facilitate a required (sometimes optional) task in a remote attestation procedure.

The base set of RATS roles is:

Claimant: The producer of trustworthiness assertions pertaining to an Attester; that may or not have a root-of-trust for measurement.

It is not guaranteed that a Verifier Role can appraise the output of a Claimant via reference values (in contrast to the output of an Attester).
Examples of Claimant assertions include: * The hardware, firmware and software components of the Attester. * The manufacturer of Attester components. * The Attester’s current configuration. * The Attester’s current location – e.g. GPS coordinates. * The method by which binding of an attester to an RTR. * The identifier(s) available for identifying and authenticating the Attester – e.g. Universal Entity ID (UEID).

Typically, claimant role are taken on by RATS Actors that supply chain entities (SCE). Various assertions (often represented as Claims or Trusted Claims Sets, e.g. [I-D.mandyam-eat] or [I-D.tschofenig-rats-tpa-token]).

Attester: The producer of attestation evidence that has a root of trust for reporting (RTR) and implements a conveyance protocol, authenticates using an attestation credential, consumes assertions about itself and presents it to a consumer of evidence (e.g. a relying party or a verifier). Every output of an attester can be appraised via reference values.

Authentication Checker: The consumer of signed assertions such as trusted claim sets or attestation evidence that assesses the trustworthiness or other trust relationships of the information consumed via trusted third parties or external trust authorities, such as a privacy certificate authority. In certain environments, an Authentication Checker can assess a system’s trustworthiness via external trust anchors, implicitly.

Verifier: The consumer of attestation evidence that has a root of trust for verification (RTV), implements conveyance protocols, appraises attestation evidence against reference values or policies, and makes verification results available to relying parties.

Relying Party: The consumer and assessor of verifier or Authentication Checker results for the purpose of improved risk management, operational efficiency, security, privacy (natural or legal person) or safety. The verifier and/or authentication checker roles and the relying party role may be tightly integrated.

4. RATS Actors

RATS Actors may be any entity, such as an user, organization, execution environment, device or service provider, that takes on (implements) one or more RATS Roles and performs RATS Duties and/or RATS Interactions. RATS Interactions occur between RATS Actors. The methods whereby RATS Actors are identified, discovered, and
connectivity established are out-of-scope for this architecture. In contrast, if multiple RATS Roles reside on a single RATS Actor, the definition of RATS Interactions is out-of-scope of the RATS architecture, if no network protocols are required.

![Figure 1: RATS Actor-Role Interactions](image)

RATS Actors have the following properties:

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

* Decomposability - A singleton RATS Actor possessing multiple RATS Roles can be separated into multiple RATS Actors. RATS Interactions may occur between them.

* Composability - RATS Actors possessing different RATS Roles can be combined into a singleton RATS Actor possessing the union of RATS Roles. RATS Interactions between combined RATS Actors ceases.

Interactions between RATS Roles belonging to the same RATS Actor are generally believed to be uninteresting. Actor operations that apply resiliency, scaling, load balancing or replication are generally believed to be uninteresting.

4.1. RATS Duties

A RATS Role can take on one or more duties. RATS Duties are role-internal functions that do not require interaction with other RATS Roles. In general, and RATS Duties are typically associated with a RATS Role. The list presented in this document is exhaustive. Also, there can be usage scenario where RATS Duties are associated with other RATS Roles than illustrated below:
4.1.1. Attester Duties

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

4.1.2. Verifier Duties

- Acquisition and storage of assertion semantics
- Acquisition and storage of appraisal policies
- Verification of Attester Identity (attestation provenance)
- Comparing assertions or evidence with reference values according to appraisal policies
- Validate authentication information based on public keys, signatures, secrets that are shielded, or secrets that are access restricted via protection profiles

4.1.3. Claimant Duties

- Hardens the device or service that implements the Attester role
- Provisions device identities and/or key material accessible to the Attester role
- Evaluates trustworthiness during manufacturing, supply chain and onboarding
- Produces trustworthiness assertions applicable to the Attestor role
- Embeds trustworthiness assertions about the Attester role in the device or service during manufacturing, supply chain or onboarding

4.1.4. Relying Party Duties

- Evaluate assertions/evidence locally as far as possible
- Compare trust policies to attestation-results based on assertions or evidence
- Enforce policies or create input for risk engines
4.1.5. RATS Interactions

The flow of information between RATS Roles located on RATS Actors compose individual remote attestation procedures. The RATS Architecture provides a set of standard interactions between the RATS Roles defined in this document in order to enable this composability. In this section, common interactions between roles are specified. This list of interactions is not exhaustive, but provides the basis to create various standard RATS.

Every RATS Interaction specified below is based on the information flow between two RATS Roles defined above. Every RATS Interaction is conducted via an Interconnect between corresponding RATS Roles that RATS Actors take on. If more than one RATS Role resides on the same RATS Actor, a network protocol might not be required. If RATS Roles are collapsed into a singular RATS Actor in this way, the method of conveying information is out-of-scope of this document. If network protocols are used to convey corresponding information between RATS Roles (collapsed on a singular RATS Actor or not), the definitions and requirements defined in this document apply.

In essence, an Interconnect is an abstract "distance-less" channel between RATS Actors that can range from General Purpose Input Output (GPIO) interfaces to the Internet.

Attester/Verifier: The most basic RATS interaction is between the creator of evidence (Attester) and its complementary remote attestation service (Verifier). In order to convey evidence (or assertions that are not accompanied by a proof of their validity) this RATS Interaction is required.

Attester/Relying-Party: A Relying Party typically requires external help to either validate authentication information or to appraise evidence presented by an Attester. In most cases, a Relying Party requires a corresponding Verifier to process the assertions/evidence received. In consequence, (a subset of) the information received by an Attester must be relayed securely to a Verifier.

Relying-Party/Verifier: Typically, trusted assertions or evidence are conveyed from an Attester to a Relying Party. In an open ecosystem, such as the Internet, the appraisal of the evidence presented by an Attester provided in order to assess its trustworthiness requires a remote attestation service. Hence, either the RATS roles of Verifier and Relying Party are collapsed and compose a single RATS Actor, or - if they reside on separate RATS Actors - a Relying Party requires appropriate configuration or a discovery/join/rendezvous service to initiate a RATS Interaction with an appropriate and trusted Verifier.
Attestation information originating from an Attester that is relayed via a Relying Party must be protected from replay or relay attacks, accordingly. In a closed ecosystem, trustworthiness with respect to the Attester can be achieved via a simply query to the Verifier. In an open ecosystem, the information conveyed in this interaction can include integrity measurements of every distinguishable software component that has been executed since its last boot cycle.

In the scope of RATS, this interaction encompasses the largest variety of information conveyed.

Claimant/Verifier: The intended operational state an Attester is intended to be in, is defined by the supply chain entities that manufacture and maintain the Attestor. In order to appraise trusted assertions or evidence conveyed by the Attester, every distinguishable system component the Attester is composed of can provide trusted assertions or evidence about its trustworthiness. A corresponding verifier that is tasked with assessing the trustworthiness of the Attester potentially requires a multitude of sources of reference values according to policies and the information provided. As Relying Parties often have to discover an appropriate Verifier, a Verifier has to obtain and potentially store appropriate reference values in order to assess assertions or evidence about trustworthiness.

Claimant/Attester: To enable RATS, trustworthy assertions have to be embedded in an Attester by its manufacturer. In some cases this involves various types of roots of trust. In other cases shielded pre-master secrets in combination with key derivation functions (KDF) provide this binding of trusted information to an Attester. A supply chain entity can embed additional trusted assertions to an Attester. These assertion can also be used to assert the trustworthiness on behalf of a separate RATS Actor or they can originate from an external entity (e.g. a security certification authority).

5. Application of RATS

Attester are typically composite devices (in the case of atomically integrated devices that would result in a composite device with one component) or services. Services are software components - e.g. a daemon, a virtual network function (vnf) or a network security function (nsf) - that can reside on one or more Attester and are not necessarily bound to a specific set of hardware devices.
Relevant decision-factors that influence the composition of RATS Roles on RATS Actors, which result in specific work-flows are (amongst others):

- which RATS Role (or correspondingly, which RATS Actore that is taking on specific RATS roles) is triggering a Remote Attestation Procedure
- which entities are involved in a Remote Attestation Procedure (e.g. the Attester itself, trusted third parties, specific trust anchors, or other sources of assertions)
- the capabilities of the protocols used (e.g. challenge-response based, RESTful, or uni-directional)
- the security requirements and security capabilities of systems in a domain of application
- the risks and corresponding threats that are intended to be mitigated

5.1. Trust and Trustworthiness

[RFC4949] provides definitions that highlight the difference between a "trusted system" and a "trustworthy system". The following definitions exclude the explicit specialization of concepts that are "environmental disruption" as well as "human user and operator errors".

A trusted system in the context of RATS "operates as expected, according to design and policy, doing what is required and not doing other things" [RFC4949]. A trustworthy system is a system "that not only is trusted, but also warrants that trust because the system’s behavior can be validated in some convincing way, such as through formal analysis or code review" [RFC4949].

The goal of RATS is to convey information about system component characteristics, such as integrity or authenticity, that can be appraised in a convincing way.

RATS require trust relationships with third parties that qualify assertions about, for example, origin of data, the manufacturer or the capabilities of a system, or the origination of attestation evidence (attestation provenance). Without trusted authorities (e.g. a certificate authority) it is virtually impossible to assess the level of assurance (or resulting level of confidence, correspondingly) of information produced by RATS. Trusting a system does not make it trustworthy. Assessing trustworthiness requires the
conveyance of evidence that a system is a trustworthy system, which has to originate from the system itself and has to be convincing. If the convincing information is not originating from the system itself, it comprises trusted claim sets and not evidence. In essence, the attestation provenance of attestation evidence is the system that intends to present its trustworthiness in a believable manner.

The essential basis for trust in the information created via RATS are roots of trust.

Roots of trust are defined by the NIST special publication 800-164 draft as "security primitives composed of hardware, firmware and/or software that provide a set of trusted, security-critical functions. They must always behave in an expected manner because their misbehavior cannot be detected. As such, RoTs need to be secured by their design. Hardware RoTs are preferred over software RoTs due to their immutability, smaller attack surface, and more reliable behavior."

If the root of trust involved is a root of trust for measurement (RTM), the producer of information takes on the role of an asserter. An asserter can also make use of a root of trust for integrity (RTI) in order to increase the level of assurance in the assertions produced. If the root of trust involved is a root of trust for reporting (RTR), the producer of information takes on the role of an attester.

5.2. Claims and Evidence

The RATS asserter role produces measurements about the system’s characteristics in the form of signed (sometimes un-signed) claim sets in order to convey information. A secret signing key is required for this procedure, which is typically stored in a shielded location that can be trusted, for example, via a root of trust for storage (RTS).

The RATS attester role produces signed attestation evidence in order to convey information. The secret key required for this procedure is stored in a shielded location that only allows access to that key, if a specific operational state of the system is met. The trust with respect to this origination is based on a root of trust for reporting.

5.3. RATS Information Flows

There are six roles defined in the RATS architecture. iFigure 2 provides a simplified overview of the RATS Roles defined above,
illustrating a general Interconnect in the center that facilitates all RATS Interactions.

![Diagram](image)

**Figure 2: Overall Relationships of Roles in the RATS Architecture**

6. Exemplary Composition of Roles

In order to provide an intuitive understanding how the roles used in RATS can be composed into work-flows, this document provides a few example work-flows. Boxes in the following examples that include more than one role are systems that take on more than one role.

6.1. Conveyance of Trusted Claim Sets Validated by Signature

If there is a trust relationship between a trusted third party that can assert that signed claims created by a claimant guarantee a trustworthy origination of claim, the work-flow depicted in Figure 3 can facilitate a trust-based implicit remote attestation procedure. The information conveyed are signed claim sets that are trusted via an authoritative third party. In this work-flow claim emission is triggered by the claimant. Variations based on requests emitted by the relying party can be easily facilitated by the same set of roles.
6.2. Conveyance of Attestation Evidence Appraised by a Verifier

If there is trust in the root of trust for reporting based on the assertions of a trusted third party, the work-flow depicted in Figure 4 can facilitate an evidence-based explicit remote attestation procedure. The information conveyed is signed attestation evidence that is created by the trusted verifier. In this work-flow claims do not necessarily have to be signed and the work-flow is triggered by the attestor that aggregates claims from a root of trust of measurement. Variations based on requests emitted by the verifier can be easily facilitated by the same set of roles.

7. The Scope of RATS

During its evolution, the term Remote Attestation has been used in multiple contexts and multiple scopes and in consequence accumulated various connotations with slightly different semantic meaning.
Correspondingly, Remote Attestation Procedures (RATS) are employed in various usage scenarios and different environments.

In order to better understand and grasp the intent and meaning of specific RATS in the scope of the security area - including the requirements that are addressed by them - this document provides an overview of existing work, its background, and common terminology. As the contribution, from that state-of-the-art a set of terms that provides a stable basis for future work on RATS in the IETF is derived.

In essence, a prerequisite for providing an adequate set of terms and definitions for the RATS architecture is a general understanding and a common definitions of "what" RATS can accomplish "how" RATS can to be used.

Please note that this section is still missing various references and is considered "under construction". The majority of definitions is still only originating from IETF work. Future iterations will pull in more complementary definitions from other SDO (e.g. Global Platform, TCG, etc.) and a general structure template to highlight semantic relationships and capable of resolving potential discrepancies will be introduced. A section of context awareness will provide further insight on how Attestation procedures are vital to ongoing work in the IETF (e.g. I2NSF & tokbind). The definitions in the section about RATS are still self-describing in this version. Additional explanatory text will be added to provide more context and coherence.

7.1. The Lying Endpoint Problem

A very prominent goal of RATS is to address the "lying endpoint problem". The lying endpoint problem is characterized as a condition of a Computing Context where the information or behavior embedded, created, relayed, stored, or emitted by the Computing Context is not "correct" according to expectations of the authorized system designers, operators and users. There can be multiple reasons why these expectations are incorrect, either from malicious Activity, unanticipated conditions or accidental means. The observed behavior, nevertheless, appears to be a compromised Computing Context.

Attempts to "scrub" the data or "proxy" control elements implies the existence of a more fundamental trusted endpoint that is operating correctly. Therefore, Remote Attestation - the technology designed to detect and mitigate the "lying endpoint problem" - must be trusted to behave correctly independent of other controls.

Consequently, a "lying endpoint" cannot also be a "trusted system".
Remote Attestation procedures are intended to enable the consumer of information emitted by a Computing Context to assess the validity and integrity of the information transferred. The approach is based, for example, on the assumption that if attestation evidence can be provided in order to prove the integrity of every software instance installed involved in the activity of creating the emitted information in question, the emitted information can be considered valid and integer.

In contrast, such Evidence has to be impossible to create if the software instances used in a Computing Context are compromised. Attestation activities that are intended to create this Evidence therefore also provide guarantees about the validity of the Evidence they can create.

7.1.1. How the RATS Architecture Addresses the Lying Endpoint Problem

RATS imply the involvement of at least two players (roles) who seek to overcome the lying endpoint problem. The Verifier wishes to consume application data supplied by a Computing Context. But before application data is consumed, the Verifier obtains Attestation Evidence about the Computing Context to assess likelihood of poisoned data due to endpoint compromise or failure. Remote Attestation argues that a system’s integrity characteristics should not be believed until rationale for believability is presented to the relying party seeking to interact with the system.

An Interconnect defines an untrusted channel between subject and object wherein the rationale for believability is securely exchanged. The type of interconnect technology could vary widely, ranging from GPIO pins, to a PC peripheral IO bus, to the Internet, to a direct physical connection, to a wireless radio-receiver association, or to a world wide mesh of peers. In other words, virtually every kind communication path could be used as the "Interconnect" in RATS. In fact, a single party could take on all roles at the same time (e.g. Self Encrypting Devices).

Attestation evidence can be thought of as the topics of the exchange that is created the operational primitives of a root of trust for reporting. Evidence may be structured in an interoperable format called claims that may include references to the claimants which are asserting the claims. RATS aims to define "interoperable Remote Attestation" such that evidence can be created and consumed by different ecosystem systems and can be securely exchanged by a broad set of network protocols.
8. RATS Terminology

This document relies on terminology found in [RFC4949]. This document presumes the reader is familiar with the following terms.

- Cryptography
- Entity (System entity)
- Identity
- Object
- Principal
- Proof-of-possession protocol
- Security environment (Environment)
- Security perimeter
- Subject
- Subsystem
- System
- Target-of-Evaluation (TOE)
- Trusted Computing Base (TCB)
- Trusted Platform Module (TPM)
- Trusted (Trustworthy) system
- Verification

Terminology defined by this document is preceded by a dollar sign ($) to distinguish it from terms defined elsewhere and as a way to disambiguate term definition from explanatory text.

Terms defined by this document that are subsequently used by this document are distinguished by capitalizing the first letter of the term (e.g. Term or First_word Second_word).
8.1. Computing Context

This section introduces the term Computing Context in order to specialize the notions of environment and endpoint to terminology that has relevance to trusted computing. Attestation is a discipline of trusted computing.

A Computing Context could refer to a large variety of endpoints. Examples include but are not limited to: the compartmentalization of physical resources, the separation of software instances with different dependencies in dedicated containers, and the nesting of virtual components via hardware-based and software-based solutions. The number of approaches and techniques to construct an endpoint continuously changes with new innovation. Hence, it isn't a goal of this document to define remote attestation for a fixed set of endpoints. Rather, it attempts to define endpoints conceptually and rely on Claims management as a way to clarify the details and specific attributes of conceptual endpoints.

Computing Contexts may be recursive in nature in that it could be composed of a system that is itself a composite of subsystems. In consequence, a system may be composed of other systems that may be further composed of one or more Computing Contexts capable of taking on the RATS roles. The scope and application of these roles can range from:

- Continuous mutual Attestation procedures of every subsystem inside a composite device, to
- Sporadic Remote Attestation of unknown parties via heterogeneous Interconnects.

Analogously, the increasing number of features and functions that constitute components of a device start to blur the lines that are required to categorize each solution and approach precisely. To address this increasingly challenging categorization, the term Computing Context defines the characteristics of the (sub)systems that can take on the role of an Attester and/or the role of a Verifier. This approach is intended to provide a stable basis of definitions for future solutions that continuous to remain viable long-term.

$\text{Computing Context : An umbrella term that combines the scope of the definitions of endpoint [ref NEA], device [ref lar], and thing [ref t2trg], including hardware-based and software-based sub-contexts that constitute independent, isolated and distinguishable slices of a Computing Context created by compartmentalization}$

mechanisms, such as Trusted Execution Environments (TEE), Hardware Security Modules (HSM) or Virtual Network Function (VNF) contexts.


While the semantic relationships highlighted above constitute the fundamental basis to provide a define Computing Context, the following list of object characteristics is intended to improve the application of the term and provide a better understanding of its meaning:


Computing context characteristics provide the following: * An independent environment in regard to executing and running software, * An isolated control plane state (by potentially interacting with other Computing Contexts), * A dedicated management interface by which control plane behavior can be effected, * Unique identification towards reliable disambiguation within a given scope.

Computing context characteristics do not necessarily include a network interface with associated network addresses (as required by the definition of an endpoint) - although it is very likely to have (access to) one.

[Issue: This conclusion could be incorrect] In contrast, a container [ref docker, find a more general term here] context is not a distinguishable isolated slice of an information system and therefore is not an independent Computing Context. [more feedback on this statement is required as the capabilities of docker-like functions evolve continuously]

Examples include: a smart phone, a nested virtual machine, a virtualized firewall function running distributed on a cluster of physical and virtual nodes, or a trust-zone.

8.1.2. Computing Context Semantic Relationships

Computing Contexts may relate to other Computing Contexts that are decomposable in a variety of ways.

- Singleton,
- Tuples (e.g. 2-tuple, n-tuple),
The scope of Computing Context encompasses a broad spectrum of systems including, but not limited to:

- An information system,
- An object,
- A composition of objects,
- A system component,
- A system sub-component,
- A composition of system sub-components,
- A system entity,
- A composition of system entities.

A Computing Context may be realized in a variety of ways including, but not limited to:

- A process, thread or task as defined by an operating system,
- A privileged operating system task, interrupt handler or event handler,
- A virtual machine,
- A virtual machine monitor,
- A processor mode (e.g. system management mode),
- A co-processor,
- A peripheral device,
- A secure element,
- A trusted execution environment,
- A controller, sensor, actuator, switch, router or gateway,
o An FPGA,
o An ASIC,
o A memory resource,
o A storage resource.

Analogously, a computing sub-context is a decomposition of a
Computing Context; a subsystem is a decomposition of a system; a sub-
component is a decomposition of a component; and a peer node is a
decomposition of a node cluster.

A formal semantic relationship is therefore expressed using an
information model that captures interactions, relationships, bindings
and interfaces among systems, subsystems, system components, system
entities or objects.

[Issue: A tangible relationship to an information model is required
here] An information model that richly captures Computing Context
semantics is therefore believed to be relevant if not fundamental to
Remote Attestation.

8.1.3. Computing Context Identity

The identity of a Computing Context implies there is a binding
operation between an identifier and the Computing Context.

$ Computing Context Identity: Computing Context Identity provides
the basis for associating attestation Evidence about a particular
Computing Context to create believable knowledge about attestation
provenance.

Confidence in the identity assurance level [NIST SP-800-63-3] or the
assurance levels for identity authentication [RFC4949] is a property
of the identifier uniqueness properties and binding operation
veracity. Such properties impact the trustworthiness of associated
attestation Evidence.

8.2. Remote Attestation Concepts

Attestation Evidence created by RATS is a form of telemetry about a
computing environment that enables better security risk management
through disclosure of security properties of the environment.
Attestation may be performed locally (within the same computing
environment) or remotely (between different computing environments).
The exchange of attestation evidence can be formalized to include
well-defined protocol, message syntax and semantics.
8.3. Core RATS Terminology

$ Attestation: The creation of evidence by the Attester based on measurements or other claimant output.

A form of telemetry involving the delivery of Claims describing various security properties of a Computing Context by an Attester, such that the Claims can be used as Evidence toward convincing a Verifier regarding trustworthiness of the Computing Context.

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

$ Verification: The appraisal of Evidence by the Verifier who evaluates it against a reference policy. See also RFC4949 [1].

$ Remote Attestation: A procedure involving Attestation, Conveyance and Verification.

8.4. RATS Information Model Terminology

Evidence conveyed to a Verifier by an Attester is structured to facilitate syntactic and semantic interoperability. An information model defines the tag namespaces used to create tag-value pairs containing discrete bits of Evidence.

$ Evidence: A set of Measurements, quality metrics, quality procedures or assurance criteria about an Computing Context’s behavioral, operational and intrinsic characteristics.

$ Claim: Structured Evidence asserted about a Computing Context. It contains metadata that informs regarding the type, class, representation and semantics of Evidence information. A Claim is represented as a name-value pair consisting of a Claim Name and a Claim Value [RFC7519]. In the context of SACM, a Claim is also specialized as an attribute-value pair that is intended to be related to a statement [I-D.ietf-sacm-terminology].

$ Attestable Claim: Structured Evidence including one or more Claims that are asserted by a Claimant (Note: an Attester role doubles as a Claimant role). An Attestable Claim has the following structure:

1. A Claim or Claims.
4. Proof the Claimant intended to make these Claims.

Note: Proofs of Claims assertions may be separated from the Claim itself. For example, a secure transport over which Claims are conveyed where Claimant’s signing key integrity protects the transport payload could be used as proof of Claim assertion. Alternatively, each Claim could be separately signed by a Claimant.

$ Attested (Asserted) Claim: An Attestable Claim where the proof elements are populated.

$ Evidence (Claims) Creation: Instantiation of Attested Claims by a Claimant.

$ Evidence (Claims) Collection: Assembling of Attested Claims by an Attester for the purpose of Conveyance.

$ Verified (Valid) Claim: An Attested Claim where the proof elements have been verified by a Verifier according to a policy that identifies trusted Claimants and/or trusted Evidence values.

8.5. RATS Work-Flow Terminology

This section introduces terms and definitions that are required to illustrate the scope and the granularity of RATS workflows in the domain of security automation. Terms defined in the following sections will be based on this workflow-related definitions.

In general, RATS are composed of iterative activities that can be conducted in intervals. It is neither a generic set of actions nor simply a task, because the actual actions to be conducted by RATS can vary significantly depending on the protocols employed and types of Computing Contexts involved.

$ Activity: A sequence of actions conducted by Computing Contexts that compose a Remote Attestation procedure. The actual composition of actions can vary, depending on the characteristics of the Computing Context they are conducted by/in and the protocols used to utilize an Interconnect. A single Activity provides only a minimal amount of semantic context, e.g. defined by the Activity’s requirements imposed upon the Computing Context, or via the set of actions it is composed of. Example: The Conveyance of cryptographic Evidence or the appraisal of Evidence via imperative guidance.

$ Task: A unit of work to be done or undertaken.
In the scope of RATS, a task is a procedure to be conducted. Example: A Verifier can be tasked with the appraisal of Evidence originating from a specific type of Computing Contexts providing appropriate identities.

$ Action:  The accomplishment of a thing usually over a period of time, in stages, or with the possibility of repetition.

In the scope of RATS, an action is the execution of an operation or function in the scope of an Activity conducted by a Computing Context. A single action provides no semantic context by itself, although it can limit potential semantic contexts of RATS to a specific scope. Example: Signing an existing public key via a specific openssl library, transmitting data, or receiving data are actions.

$ Procedure:  A series of actions that are done in a certain way or order.

In the scope of RATS, a procedure is a composition of activities (sequences of actions) that is intended to create a well specified result with a well established semantic context. Example: The activities of Attestation, Conveyance and Verification compose a Remote Attestation procedure.

8.6. RATS Reference Use Cases

A "lying endpoint" is not trustworthy.

This document provides NNN prominent examples of use cases Attestation procedures are intended to address:

- Verification of the source integrity of a Computing Context via data integrity proofing of installed software instances that are executed, and
- Verification of the identity proofing of a Computing Context.

8.6.1. Use Case A

8.6.2. Use Case B

8.7. RATS Reference Terminology

$ Attestable Computing Context:  A Computing Context where a Claimant is able to create Claims, an Attester is able to Attest those Claims and a Verifier is able to verify the Claims.
$ Attestation Identity: An identity that refers to an Attester.

$ Attestation Identity Credential: A credential used to authenticate an Attestation Identity.

$ Attestation Identity Key (AIK): An Attestation Identity Credential in the form of an asymmetric cryptographic key where the AIK private key is protected by a Computing Context with protection properties that are stronger than the Computing Context about which the AIK attests. A root-of-trust Computing Context normally protects AIK private keys.

$ Claimant Identity: An identity that refers to a Claimant.

$ Claimant Identity Credential: A credential used to authenticate a Claimant Identity.

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

$ Reference Integrity Measurements: Signed Measurements about a Computing Context’s characteristics that are provided by a vendor or manufacturer and are intended to be used as declarative guidance [I-D.ietf-sacm-terminology] (e.g. a signed CoSWID).

$ Root-of-trust: The Computing Context that protects the following where no other Computing Context is expected to provide its Attestation Evidence: + Attestation Evidence. + AIKs. + Code used during the collection and reporting of Attestation Evidence.

$ Root-of-trust-for-measurement (RTM): A trusted Computing Context where a Claimant creates integrity Measurements and other Evidence about a Computing Context where no other Computing Context is expected to provide its Attestation Evidence.

$ Root-of-trust-for-reporting (RTR): A trusted Computing Context where an Attester stages reporting of Claims where no other Computing Context is expected to provide its Attestation Evidence.

$ Root-of-trust-for-storage (RTS): A trusted Computing Context where a Claimant or Attester stores Claims, Evidence, credentials or policies associated with Attestation where no other Computing Context is expected to provide its Attestation Evidence.
Trustworthy Computing Context: A Computing Context that guarantees trustworthy behavior and/or composition (with respect to certain declarative guidance and a scope of confidence). A trustworthy Computing Context is a trustworthy system.

Trustworthy Statement: Evidence conveyed by a Computing Context that is not necessarily trustworthy.

8.8. Interpretations of RFC4949 Terminology for Attestation

Assurance: An attribute of an information system that provides grounds for having confidence that the system operates such that the system’s security policy is enforced [RFC4949] (see Trusted System below).

In common criteria, assurance is the basis for the metric level of assurance, which represents the "confidence that a system’s principal security features are reliably implemented".

The NIST Handbook [get ref from 4949] notes that the levels of assurance defined in Common Criteria represent "a degree of confidence, not a true measure of how secure the system actually is. This distinction is necessary because it is extremely difficult-and in many cases, virtually impossible-to know exactly how secure a system is."

Historically, assurance was well-defined in the Orange Book [http://csrc.nist.gov/publications/history/dod85.pdf] as "guaranteeing or providing confidence that the security policy has been implemented correctly and that the protection-relevant elements of the system do, indeed, accurately mediate and enforce the intent of that policy. By extension, assurance must include a guarantee that the trusted portion of the system works only as intended."

Confidence: The definition of correctness integrity in [RFC4949] notes that "source integrity refers to confidence in data values". Hence, confidence in an Attestation procedure is referring to the degree of trustworthiness of an Attestation Activity that produces Evidence (Attester), of an Conveyance Activity that transfers Evidence (interconnect), and of a Verification Activity that appraises Evidence (Verifier), in respect to correctness integrity.

Correctness: The property of a system that is guaranteed as the result of formal Verification activities.
Correctness integrity: The property that the information represented by data is accurate and consistent.

Data Integrity: (a) The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner. (See: data integrity service. Compare: correctness integrity, source integrity.)

(b) The property that information has not been modified or destroyed in an unauthorized manner.

Entity: A principal, Subject, relying party or stake holder in an Attestation ecosystem.

Identity: The set of attributes that distinguishes a principal.

Identifier: The set of attributes that distinguishes an object.

Identity Proofing: A vetting process that verifies the information used to establish the identity of a system entity.

(Information) System: An organized assembly of computing and communication resources and procedures - i.e., equipment and services, together with their supporting infrastructure, facilities, and personnel - that create, collect, record, process, store, transport, retrieve, display, disseminate, control, or dispose of information to accomplish a specified set of functions.

Object: A system component that contains or receives information.

Source Integrity: The property that data is trustworthy (i.e., worthy of reliance or trust), based on the trustworthiness of its sources and the trustworthiness of any procedures used for handling data in the system.

Subject: A Computing Context acting in accordance with the interests of a principal.

Subsystem: A collection of related system components that together perform a system function or deliver a system service.

System Component: An instance of a system resource that (a) forms a physical or logical part of the system, (b) has specified functions and interfaces, and (c) is extant (e.g., by policies or specifications) outside of other parts of the system. (See: subsystem.)
An identifiable and self-contained part of a $Target-of-Evaluation.

Token: A data structure suitable for containing Claims.

Trusted (Trustworthy) System: A system that operates as expected, according to design and policy, doing what is required – despite environmental disruption, human user and operator errors, and attacks by hostile parties – and not doing other things.

Verification: (a) The process of examining information to establish the truth of a claimed fact or value.
   (b) The process of comparing two levels of system specification for proper correspondence, such as comparing a security model with a top-level specification, a top-level specification with source code, or source code with object code.

8.9. Building Block Vocabulary (Not in RFC4949)

[working title, pulled from various sources, vital]

Attribute: TBD
Characteristic: TBD
Context: TBD
Endpoint: TBD
Environment: TBD
Manifest: TBD
Telemetry: An automated communications process by which data, readings, Measurements and Evidence are collected at remote points and transmitted to receiving equipment for monitoring and analysis. Derived from the Greek roots tele = remote, and metron = measure.

9. IANA considerations

This document will include requests to IANA:

   o first item
   o second item
10. Security Considerations

There are always some.

11. Acknowledgements

Maybe.

12. Change Log

No changes yet.

13. References

13.1. Normative References


13.2. Informative References


[I-D.tschofenig-rats-psa-token]
Tschofenig, H., Frost, S., Brossard, M., and A. Shaw,
"Arm's Platform Security Architecture (PSA) Attestation
Token", draft-tschofenig-rats-psa-token-00 (work in
progress), March 2019.

FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007,

[RFC7519]  Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
(JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015,

13.3.  URIs


Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt  64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Monty Wiseman
GE Global Research
USA

Email: monty.wiseman@ge.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge  CB1 9NJ
UK

Email: hannes.tschofenig@gmx.net
YANG Module for Basic Challenge-Response-based Remote Attestation Procedures
draft-birkholz-rats-basic-yang-module-00

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 13, 2019.
1. Introduction

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

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

2. The YANG Module for Basic Remote Attestation Procedures

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

2.1. Tree format

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

rpcs:
  +--x tpm2-challenge-response-attestation
    |  +--w input
    |     |  +--w tpm2-attestation-challenge
    |     |     |  +--w pcr-list* []
    |     |     |     |  +--w pcr
    |     |     |  +--w pcr-indices* uint8
```

Internet-Draft                    BRAT                        March 2019

+-ro certificate-name?          string
+-ro attestation-certificate?   ietfct:end-entity-cert-cms
+-ro (key-identifier)?
  +--:(public-key)
  |  +--ro pub-key-id?          binary
  +--:(uuid)
  |  +--ro uuid-value?          binary
+-x log-retrieval
  +--w input
    +--w log-selector* [node-name]
    |  +--w node-name               string
    |  +--w node-physical-index?    int32 {ietfhw:entity-mib}?
    |  +--w (index-type)?
    |     +--:(last-entry)
    |     |  +--w last-entry-value?    binary
    |     +--:(index)
    |     |  +--w index-number?        uint64
    |     +--:(timestamp)
    |        +--w timestamp?       yang:date-and-time
    +--w log-type                identityref
    +--w pcr-list* []
      +--w pcr
        +--w pcr-indices*           uint8
        +--w (algo-registry-type)
          +--:(tcg)
          |  +--w tcg-hash-algo-id?    uint16
          +--:(ietf)
          |  +--w ietf-ni-hash-algo-id? uint8
        +--w log-entry-quantity?    uint16
    +--ro output
    +--ro system-event-logs
    +--ro node-data* [node-name]
      +--ro node-name             string
      +--ro node-physical-index?  int32 {ietfhw:entity-mib}?
      +--ro up-time?              uint32
      +--ro tpm-updated* [tpm_name]
        +--ro tpm_name             string
        +--ro tpm-physical-index?  int32 {ietfhw:entity-mib}?
      +--ro log-result
    +--ro (log-type)
      +--:(bios)
        +--ro bios-event-logs
          +--ro bios-event-entry* [event-number]
            +--ro event-number        uint32
            +--ro event-type?         uint32
            +--ro pcr-index?          uint16
            +--ro digest-list* []
              +--ro (algo-registry-type)
2.2. Raw Format

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

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

  organization
    "Fraunhofer SIT";
  contact
    "Henk Birkholz
     Fraunhofer Institute for Secure Information Technology
     Email: henk.birkholz@sit.fraunhofer.de";
  description
    "A YANG module to enable a TPM 2.0 based remote attestation procedure.
     Copyright (C) Fraunhofer SIT (2018).";
  revision "2018-06-15" {

<CODE ENDS>
description
"Initial version";
reference
"draft-birkholz-yang-basic-remote-attestation";
}

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

grouping hash {
  description
  "The hash value including hash-algo identifier";
  list hash-digests {
    description
    "The list of hashes.";
    container hash-digest {
      description
      "A hash value based on a hash algorithm registered by an
SDO.

uses hash-algo;
leaf hash-value {
    type binary;
    description
        "The binary representaion of the hash value."
    }
}

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

grouping pcr-selection {
    description
        "A Verifier can request one or more PCR values uses its
        individually created AC. The corresponding selection filter is
        represented in this grouping. Requesting a PCR value that is not in
        scope of the AC used, detailed exposure via error msg should be
        avoided."
    list pcr-list {
        description
            "For each PCR in this list an individual list of banks (hash-algo)
            can be requested. It depends on the datastore, if every bank in
            this grouping is included per PCR (crude), or if each requested
            bank set is returned for each PCR individually (elegant)."
        container pcr {
            description
                "The composite of a PCR number and corresponding bank numbers."
            leaf-list pcr-indices {
                type uint8;
                description
                    "The number of the PCR. At the moment this is limited
                    32";
            }
        }
    }
}
uses hash-algo;
}
}
}

grouping pcr-selector {
  description
  "A Verifier can request the generation of an attestation certificate (a signed public attestation key (non-migratable, tpm-resident) wrt one or more PCR values. The corresponding creation input is represented in this grouping. Requesting a PCR value that is not supported results in an error, detailed exposure via error msg should be avoided.";
  list pcr-list {
    description
    "For each PCR in this list an individual hash-algo can be requested.";
    container pcr {
      description
      "The composite of a PCR number and corresponding bank numbers.";
      leaf-list pcr-index {
        type uint8;
        description
        "The numbers of the PCRs that are associated with the created key. At the moment the highest number is 32";
      }
      uses hash-algo;
    }
  }
}

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

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

grouping tpm-name {
    description
    "In a system with multiple-TPMs get the data from a specific TPM identified by the name and physical-index."
    leaf tpm_name {
        type string;
        description
        "This selects the TPM to use for attestation...";
    }
}
"Name of the TPM or All";
}
leaf tpm-physical-index {
  if-feature ietfhw:entity-mib;
  type int32 {
    range "1..2147483647";
  }
  config false;
  description "The entPhysicalIndex for the TPM."
  reference "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
}
}
grouping compute-node {
  description "In a distributed system with multiple compute nodes
    this is the node identified by name and physical-index.";
  leaf node-name {
    type string;
    description "Name of the compute node or All";
  }
  leaf node-physical-index {
    if-feature ietfhw:entity-mib;
    type int32 {
      range "1..2147483647";
    }
    config false;
    description "The entPhysicalIndex for the compute node.";
    reference "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
  }
}
grouping node-uptime {
  description "Uptime in seconds of the node.";
  leaf up-time {
    type uint32;
    description "Uptime in seconds of this node reporting its data";
  }
}
identity log-type {
  description "The type of logs available.";
}
identity bios {
    base log-type;
    description
        "Measurement log created by the BIOS/UEFI.";
}

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

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

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

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

" hash(filedata-hash, filename-hint);"
}
leaf pcr-index {
    type uint16;
    description
    "Defines the PCR index that this event extended";
}
leaf signature {
    type binary;
    description
    "The file signature";
}
}

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

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

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

        case bios {
rpc tpm2-challenge-response-attestation {
  description "This RPC accepts the input for TSS commands of the managed device. ComponentIndex from the hardware manager YANG module to refer to dedicated TPM in composite devices, e.g. smart NICs, is still a TODO.";
  input {
    container tpm2-attestation-challenge {
      description "This container includes every information element defined in the reference challenge-response interaction model for remote attestation. Corresponding values are based on TPM 2.0 structure definitions";
      uses pcr-selection;
      uses nonce;
      uses signature-scheme;
      uses attestation-key-identifier;
    }
    uses tpm-name;
  }
  output {
    list tpm2-attestation-response {
      key tpm_name;
      description "The binary output of TPM2b_Quote. An TPMS_ATTEST structure
including a length, encapsulated in a signature
uses tpm-name;
uses node-uptime;
uses compute-node;
container tpms-attest {
leaf pcrdigest {
    type binary;
    description
    "split out value of TPMS_QUOTE_INFO for convenience";
}
leaf tpms-attest-result {
    type binary;
    description
    "The complete TPM generate structure including signature.";
}
leaf tpms-attest-result-length {
    type uint32;
    description
    "Length of attest result provided by the TPM structure.";
}
description
    "A composite of value and length and list of selected
    pcrs (original name: [type]attested)";
}
leaf tpmt-signature {
    type binary;
    description
    "Split out value of the signature for convenience. TODO: check for
    length values that compleet binary value data node leafs.";
}
}
}
rpc basic-trust-establishment {
    description
    "This RPC creates a tpm-resident, non-migratable key to be used
    in TPM_Quote commands, an attestation certificate."
    input {
        uses nonce;
        uses signature-scheme;
        uses tpm-name;
        leaf certificate-name {
            type string;
            description
            "An arbitrary name for the identity certificate chain requested.";
        }
    }
    output {

list attestation-certificates {
  key tpm_name;
  description
    "Attestation Certificate data from a TPM identified by the TPM name";
  uses tpm-name;
  uses node-uptime;
  uses compute-node;
  leaf certificate-name {
    type string;
    description
    "An arbitrary name for this identity certificate or certificate chain.";
  }
}
leaf attestation-certificate {
  description
  "The binary signed certificate chain data for this identity certificate."
  type ietfct:end-entity-cert-cms;
} uses attestation-key-identifier;
}
}

crpc log-retrieval {
  description
  "Logs Entries are either identified via indices or via providing
  the last line received. The number of lines returned can be limited.
  The type of log is a choice that can be
  augmented.";
  input {
    list log-selector {
      key node-name;
      description
      "Selection of log entries to be reported.";
      uses compute-node;
      choice index-type {
        description
        "Last log entry received, log index number, or timestamp.";
        case last-entry {
          description
          "The last entry of the log already retrieved.";
          leaf last-entry-value {
            description
            "Content of an log event which matches 1:1 with a
            unique event record contained within the log. Log
            entries subsequent to this will be passed to the
            requestor. Note: if log entry values are not unique,
            this MUST return an error.";
          }
        }
      }
    }
  }
}
type binary;
}
}
case index {
    description
    "Numeric index of the last log entry retrieved, or zero.";
    leaf index-number {
        description
        "The numeric index number of a log entry. Zero means
        to start at the beginning of the log. Entries
        subsequent to this will be passed to the
        requestor.";
        type uint64;
    }
}
case timestamp {
    leaf timestamp {
        type yang:date-and-time;
        description
        "Timestamp from which to start the extraction. The next
        log entry subsequent to this timestamp is to be sent.";
    }
    description
    "Timestamp from which to start the extraction.";
}
}
}
uses log-identifier;
uses pcr-selection;
leaf log-entry-quantity {
    type uint16;
    description
    "The number of log entries to be returned. If omitted, it
    means all of them.";
}
}
output {
    container system-event-logs {
        description
        "The requested data of the measurement event logs";
        list node-data {
            key node-name;
            description
            "Event logs of a node in a distributed system
            identified by the node name";
            uses compute-node;
            uses node-uptime;
            list tpm-updated {
key tpm_name;
  description
  "TPM these events may have recorded data in";
  uses tpm-name;
}
container log-result {
  description
  "The requested entries of the corresponding log.";
  uses event-logs;
}
}
}
}
}
}
}
}
container rats-support-structures {
  leaf-list supported-algos {
    type uint16;
    description
    "Supported TPM_ALG_ID values for the TPM in question.
    Will include ComponentIndex soon.";
  }
  list tpms {
    key tpm_name;
    uses tpm-name;
    description
    "A list of TPMs in this composite device that rats can be conducted with.";
  }
  list compute-nodes {
    key node-name;
    uses compute-node;
    description
    "A list names of hardware components in this composite device that rats can be conducted with.";
  }
  container endorsement-certificates {
    list certificate {
      key tpm_name;
      uses tpm-name;
      description
      "The TPM’s endorsement-certificate.";
      leaf endorsement-certificate {
        type binary;
        mandatory true;
        description
        "The signed public endorsement key (EK) and corresponding claims (EK Certificate). In a TPM 2.0 the EK Certificate resides in a well-defined NVRAM location by the TPM vendor.";
      }
    }
  }
}
3. IANA considerations

This document will include requests to IANA:

To be defined yet.

4. Security Considerations

There are always some.

5. Acknowledgements

Not yet.

6. Change Log

Changes from version 00 to version 01:

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

7. References
7.1. Normative References

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

[I-D.ietf-netconf-crypto-types]


7.2. Informative References

[I-D.birkholz-attestation-terminology]

Authors’ Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt  64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Michael Eckel
Huawei Technologies
Feldbergstrasse 78
Darmstadt  64293
Germany

Email: michael.eckel@huawei.com
Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 13, 2019.

Copyright Notice

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

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

Reference Interaction Model for Challenge-Response-based Remote Attestation
draft-birkholz-rats-reference-interaction-model-00
1. Introduction

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

1.1. Requirements notation

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

2. Disambiguation

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

3. Scope

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

4. Component Roles

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

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

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

5. Prerequisites

Identity: An Attester must have a unique Identity in such a way that a Verifier can uniquely identify an Attester. This Identity MUST be part of the signed claims (attestation evidence) that the Attester conveys to the Verifier.

Secret: A Secret that is present on the Attester and that a Verifier can identify by its Secret ID, e.g. a public key. This Secret MUST be established before a remote attestation procedure can take place. How this Secret is established is out of scope for this reference model.
6. Remote Attestation Interaction Model

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

6.1. Information Elements

Nonce: mandatory

The Nonce (number used once) is a number intended to be unique and intended to be effectively infeasible to guess. In this reference interaction model it MUST be provided by the Verifier and MUST be used as a proof of freshness, with respect to conveyed evidence ensuring that the result of an attestation activity was created recently (i.e. triggered by the challenge emitted by the Verifier). As such, the Nonce MUST be part of the signed evidence sent by the Attester to the Verifier.

Secret ID: mandatory

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

Evidence: mandatory

The signed claims that are required to enable integrity proving of the corresponding characteristics of the Attester. Examples of claims included in attestation evidence are claims about sensor data, policies that are active on the system entity, versions of composite firmware of a platform, running software, routing tables, or information about a local time source. Attestation evidence must be cryptographically bound to the Verifier-provided Nonce, the Identity of the Attester, as well as to the Secret identified by the Secret ID.

Claim Selection: optional

An Attester MAY provide a selection of claims that are relevant to the appraisal conducted by the Verifier in order to prove correctness of the (integrity) claims created by the Attester. Usually, all available claims that are available to the Attester SHOULD be conveyed. This claim selection can be composed as complementary signed claims or can be encapsulated claims in the signed evidence that composes the evidence about integrity. This information element is optional in order to enable a Verifier to narrow down or increase the amount of received evidence. An
Attester MAY decide whether or not to provide the requested claims or not. In either case, the claim selection MUST be cryptographically bound to the signed claim set. An example for a claim selection is that a Verifier can request from an Attester (signed) Reference Integrity Measurements (RIMs), which represent a claim about the intended platform operational state of the Attester.

Identity: mandatory

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

7. Interaction Model

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

```
[Verifier]
requestAttestation(nonce, secretID, claimSelection)

[Attester]

collectClaims(claimSelection)

signEvidence(claims, secretID, nonce, identity)

[Verifier]

evidence, signature, identity ------------------------>

[Verifier]
appraise(evidence, signature, identity, nonce)
appraisalResult <==>
```

The remote attestation procedure is initiated by the Verifier, sending an attestation request to the Attester. The attestation request consists of a Nonce, a Secret ID, and a Claim Selection. The Nonce guarantees attestation freshness. The Secret ID selects the secret the Attester is requested to sign the Evidence with. The Claim Selection narrows down or increases the amount of received Evidence, if required. If the Claim Selection is empty, then by default all claims that are available on the system of the Attester SHOULD be signed and returned as Evidence. For example, the Verifier is only interested in particular information about the Attester, such as whether the device booted up in a known state, and not include information about all currently running software.
The Attester, after receiving the attestation request, collects the corresponding claims to compose the evidence the Verifier requested— or, in case the Verifier did not provide a claim selection, the Attester collects all information that can be used as complementary claims in the scope of the semantics of the remote attestation procedure. After that, the Attester signs the evidence with the secret identified by the secret ID, including the nonce and the identity information. Then the Attester sends the output back to the Verifier. Important at this point is that the nonce as well as the identity information must be cryptographically bound to the signature, i. e. it is not required for them to be present in plain text. For instance, those information can be part of the signature after a one-way function (e. g. a hash function) was applied to them. There is also a possibility to scramble the nonce or identity with other information that is known to both the Verifier and Attester. A prominent example is the IP address of the Attester that usually is known by the Attester as well as the Verifier. This extra information can be used to scramble the Nonce in order to counter certain types of relay attacks. As soon as the Verifier receives the evidence, it appraises it, including the verification of the signature, the identity, the nonce, and the claims included in the evidence. This process is application-specific and can be done by e. g. comparing the claims to known (good), expected reference claims, such as Reference Integrity Measurements (RIMs), or evaluating it in other ways. The final output, the appraisal result (also referred to as attestation result), is a new claim about properties of the Attester, i. e. whether or not it is compliant to policies, or even can be "trusted".

8. Further Context

Depending on the use cases to cover there may be additional requirements.

8.1. Confidentiality

Use confidential communication to exchange attestation information. This requirement usually is present when communication happens over insecure channels, such as the public Internet. Speaking of a suitable communication protocol, TLS is a good candidate. In private networks, such as carrier management networks, it must be evaluated whether or not the transport medium is considered confidential.

8.2. Mutual Authentication

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

8.3. Hardware-Enforcement/Support

In particular use cases hardware support can be desirable. Depending on the requirements those can be secure storage of cryptographic keys, crypto accelerators, or protected or isolated execution environments. Well-known technologies are Hardware Security Modules (HSM), Physical Unclonable Functions (PUFs), Shielded Secrets, Trusted Executions Environments (TEEs), etc.

9. Security Considerations

There are always some.

10. Acknowledgements

Very likely.

11. Change Log

Initial draft -00
Changes from version 00 to version 01:
Added details to the flow diagram

12. References

12.1. Normative References


12.2. Informative References

Authors’ Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt  64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Michael Eckel
Huawei Technologies
Feldbergstrasse 78
Darmstadt  64293
Germany

Email: michael.eckel@huawei.com
Time-Based Uni-Directional Attestation
draft-birkholz-rats-tuda-00

Abstract

This memo documents the method and bindings used to conduct time-based uni-directional attestation between distinguishable endpoints over the network.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 13, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
   1.1. Remote Attestation ..................................... 4
   1.2. Evidence Creation ........................................ 5
   1.3. Evidence Appraisal ........................................ 5
   1.4. Activities and Actions .................................... 5
   1.5. Attestation and Verification ............................... 6
   1.6. Information Elements and Conveyance ....................... 6
   1.7. TUDA Objectives .......................................... 7
   1.8. Hardware Dependencies .................................... 7
   1.9. Requirements Notation ..................................... 7
2. TUDA Core Concept ............................................. 8
3. Terminology ................................................... 9
   3.1. Universal Terms ........................................... 9
   3.2. Roles ................................................... 10
      3.2.1. General Types ......................................... 11
      3.2.2. RoT specific terms ..................................... 11
   3.3. Certificates ............................................. 11
4. Time-Based Uni-Directional Attestation ........................ 11
4.1. TUDA Information Elements Update Cycles .................... 13
5. Sync Base Protocol ............................................ 15
6. IANA Considerations ............................................ 16
7. Security Considerations ........................................ 16
8. Change Log .................................................... 16
9. Contributors ................................................... 17
10. References ................................................... 17
   10.1. Normative References ..................................... 17
   10.2. Informative References ................................... 17
Appendix A. REST Realization ...................................... 21
Appendix B. SNMP Realization ...................................... 21
   B.1. Structure of TUDA MIB ...................................... 22
      B.1.1. Cycle Index .......................................... 22
      B.1.2. Instance Index ....................................... 22
      B.1.3. Fragment Index ....................................... 22
   B.2. Relationship to Host Resources MIB ....................... 23
   B.3. Relationship to Entity MIB ................................ 23
   B.4. Relationship to Other MIBs ................................ 23
   B.5. Definition of TUDA MIB .................................... 23
Appendix C. YANG Realization ...................................... 39
Appendix D. Realization with TPM functions ....................... 54
   D.1. TPM Functions ............................................ 54
      D.1.1. Tick-Session and Tick-Stamp .......................... 54
      D.1.2. Platform Configuration Registers (PCRs) ............ 55
      D.1.3. PCR restricted Keys .................................. 55
1. Introduction

Remote attestation (RA) describes the attempt to determine and appraise properties, such as integrity and trustworthiness, of an endpoint -- the Attestor -- over a network to another endpoint -- the Verifier -- without direct access. Typically, this kind of appraisal is based on integrity measurements of software components right before they are loaded as software instances on the Attestor. In general, attestation procedures are utilizing a hardware root of trust (RoT). The TUDA protocol family uses hash values of all started software components that are stored (extended into) a Trust Anchor (the RoT) implemented as a Hardware Security Module (e.g. a Trusted Platform Module or similar) and are reported via a signature over those measurements.

This draft introduces the concept of including the exchange of evidence -- created via a hardware RoT containing a shielded secret that is inaccessible to the user -- in order to increase the confidence in a communication peer that is supposed to be a Trusted System [RFC4949]. In consequence, this document introduces the term forward authenticity.

Forward Authenticity (FA): A property of secure communication protocols, in which later compromise of the long-term keys of a data origin does not compromise past authentication of data from that origin. FA is achieved by timely recording of assessments of the authenticity from entities (via "audit logs" during "audit sessions") that are authorized for this purpose, in a time frame much shorter than that expected for the compromise of the long-term keys.
Forward Authenticity enables new level of guarantee and can be included in the basically every protocol, such as ssh, router advertisements, link layer neighbor discover, or even ICMP echo.

1.1. Remote Attestation

In essence, remote attestation (RA) is composed of three activities. The following definitions are derived from the definitions presented in [PRIRA] and [TCGGLOSS].

Attestation: The creation of one or more claims about the properties of an Attestor, such that the claims can be used as evidence.

Conveyance: The transfer of evidence from the Attestor to the Verifier via an interconnect.

Verification: The appraisal of evidence by evaluating it against declarative guidance.

With TUDA, the claims that compose the evidence are signatures over trustworthy integrity measurements created by leveraging a hardware RoT. The evidence is appraised via corresponding signatures over reference integrity measurements (RIM, represented, for example via [I-D.ietf-sacm-coswid]).

Protocols that facilitate Trust-Anchor based signatures in order to provide RATS are usually bi-directional challenge/response protocols, such as the Platform Trust Service protocol [PTS] or CAVES [PRIRA], where one entity sends a challenge that is included inside the response to prove the recentness -- the freshness (see fresh in [RFC4949]) -- of the attestation information. The corresponding interaction model tightly couples the three activities of creating, transferring and appraising evidence.

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

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

1.2. Evidence Creation

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

- Platform Configuration Registers (PCR) that store measurements consecutively (corresponding terminology: "to extend a PCR") and represent the chain of measurements as a single measurement value ("PCR value"),

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

- a dedicated source of (relative) time, e.g. a tick counter.

1.3. Evidence Appraisal

To appraise the evidence created by an Attestor, the Verifier requires corresponding Reference Integrity Measurements (RIM). Typically, a set of RIM are bundled in a RIM-Manifest (RIMM). The scope of a manifest encompasses, e.g., a platform, a device, a computing context, or a virtualised function. In order to be comparable, the hashing algorithms used by the Attestor to create the integrity measurements have to match the hashing algorithms used to create the corresponding RIM that are used by the Verifier to appraise the integrity evidence.

1.4. Activities and Actions

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

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

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

1.6. Information Elements and Conveyance

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

- is encoded in the Concise Binary Object Representation (CBOR [RFC7049]) to minimize the volume of data in motion. In this document, the composition of the CBOR data items that represent IE is described using the Concise Data Definition Language, CDDL [I-D.ietf-cbor-cddl]
- that requires a certain freshness is only created/updated when out-dated, which reduces the overall resources required from the Attestor, including the utilization of the hardware root of trust. The IE that have to be created are determined by their age or by specific state changes on the Attestor (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
fresness-window, further reducing the load imposed on the Attestor and its corresponding hardware RoT.

1.7. TUDA Objectives

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

- increase the confidence in authentication and authorization procedures,
- address the requirements of constrained-node networks,
- support interaction models that do not maintain connection-state over time, such as REST architectures [REST],
- be able to leverage existing management interfaces, such as SNMP [RFC3411]. RESTCONF [RFC8040] or CoMI [I-D.ietf-core-comi] -- and corresponding bindings,
- support broadcast and multicast schemes (e.g. [IEEE1609]),
- be able to cope with temporary loss of connectivity, and to
- provide trustworthy audit logs of past endpoint states.

1.8. Hardware Dependencies

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

1.9. Requirements Notation

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

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

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

In contrast, the TUDA protocol uses the combined output of TPM_CertifyInfo and TPM_TickStampBlob. The former provides a proof about the platform’s state by creating evidence that a certain key is bound to that state. The latter provides proof that the platform was in the specified state by using the bound key in a time operation. This combination enables a time-based attestation scheme. The approach is based on the concepts introduced in [SCALE] and [SFKE2008].

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

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

- the time of transmission of the nonce, and
- the reception of the corresponding response.

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

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

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

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

- the drift of clocks included in the hardware RoT.

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

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

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

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

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

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

3. Terminology

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

3.1. Universal Terms

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

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

In the context of SACM, a claim is also specialized as an attribute/value pair that is intended to be related to a statement [I-D.ietf-sacm-terminology].
Endpoint Attestation: the creation of evidence on the Attestor that provides proof of a set of the endpoints’ integrity measurements. This is done by digitally signing a set of PCRs using an AIK shielded by the hardware RoT.

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

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

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

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

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

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

Trustworthy Endpoint: an endpoint that guarantees trustworthy behavior and/or composition (with respect to certain declarative guidance and a scope of confidence).

Trustworthy Statement: evidence that is trustworthy conveyed by an endpoint that is not necessarily trustworthy.

3.2. Roles

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

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

TSA: a Time Stamp Authority [RFC3161]
3.2.1. General Types

Byte: the now customary synonym for octet
Cert: an X.509 certificate represented as a byte-string

3.2.2. RoT specific terms

PCR: a Platform Configuration Register that is part of a hardware root of trust and is used to securely store and report measurements about security posture
PCR-Hash: a hash value of the security posture measurements stored in a TPM PCR (e.g. regarding running software instances) represented as a byte-string

3.3. Certificates

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

4. Time-Based Uni-Directional Attestation

A Time-Based Uni-Directional Attestation (TUDA) consists of the following seven information elements. They are used to gain assurance of the Attestor’s platform configuration at a certain point in time:

TSA Certificate: The certificate of the Time Stamp Authority that is used in a subsequent synchronization protocol token. This certificate is signed by the TSA-CA.
AIK Certificate: A certificate about the Attestation Identity Key (AIK) used. This may or may not also be an [IEEE802.1AR] IDevID or LDevID, depending on their setting of the corresponding identity property. ([AIK-Credential], [AIK-Enrollment]; see Appendix D.2.1.)
Synchronization Token: The reference for attestations are the relative timestamps provided by the hardware RoT. In order to put
attestations into relation with a Real Time Clock (RTC), it is necessary to provide a cryptographic synchronization between these trusted relative timestamps and the regular RTC that is a hardware component of the Attestor. To do so, a synchronization protocol is run with a Time Stamp Authority (TSA).

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

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

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

Implicit Attestation: The actual attestation is then based upon a signed timestamp provided by the hardware RoT using the restricted temporary key that was certified in the steps above. The signed timestamp provides evidence that at this point in time (with respect to the relative time of the hardware RoT) a certain configuration existed (namely the PCR values associated with the restricted key). Together with the synchronization token this timestamp represented in relative time can then be related to the real-time clock.

Concise SWID tags: As an option to better assess the trustworthiness of an Attestor, a Verifier can request the reference hashes (RIM, which are often referred to as golden measurements) of all started software components to compare them with the entries in the measurement log. References hashes regarding installed (and therefore running) software can be provided by the manufacturer via SWID tags. SWID tags are provided by the Attestor using the Concise SWID representation [I-D.ietf-sacm-coswid] and bundled into a CBOR array (a RIM Manifest). Ideally, the reference hashes include a signature created by the manufacturer of the software to prove their integrity.
These information elements could be sent en bloc, but it is recommended to retrieve them separately to save bandwidth, since these elements have different update cycles. In most cases, retransmitting all seven information elements would result in unnecessary redundancy.

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

4.1. TUDA Information Elements Update Cycles

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

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

- volatile to a degree, because they change at the beginning of each boot cycle. This includes the capability of a hardware RoT to provide relative time which provides the basis for the synchronization token and implicit attestation--and which can reset after an endpoint is powered off.

- very volatile, because they change during an uptime cycle (the period of time an endpoint is powered on, starting with its boot). This includes the content of PCRs of a hardware RoT and thereby also the PCR-restricted signing keys used for attestation.

Depending on this "lifetime of state", data has to be transported over the wire, or not. E.g. information that does not change due to a reboot typically has to be transported only once between the Attestor and the Verifier.

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

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

Sync-Token ------------------------------------------->
Certify-Info ----------------------------------------->
Measurement Log -------------------------------------->
Attestation ------------------------------------------>

<Time Passed>

Attestation ------------------------------------------>

Figure 1: Example sequence of events

5. Sync Base Protocol

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

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

- The corresponding response includes a data-structure incorporating the trusted timestamp token and its signature created by the TSA.

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

The three time-related values -- the relative timestamps provided by the hardware RoT ("left" and "right") and the TSA timestamp -- and their corresponding signatures are aggregated in order to create a corresponding Sync Token to be used as a TUDA Information Element that can be conveyed as evidence to a Verifier.
The drift of a clock incorporated in the hardware RoT that drives the increments of the tick counter constitutes one of the triggers that can initiate a TUDA Information Element Update Cycle in respect to the freshness of the available Sync Token.

content TBD

6. IANA Considerations

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

TBD

7. Security Considerations

There are Security Considerations. TBD

8. Change Log

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

Changes from version 03 to version 04:

  o Refactoring of Introduction, intend, scope and audience

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

  o Added YANG module

  o Added missing changelog entry

Changes from version 02 to version 03:

  o Moved base concept out of Introduction

  o First refactoring of Introduction and Concept

  o First restructuring of Appendices and improved references

Changes from version 01 to version 02:

  o Restructuring of Introduction, highlighting conceptual prerequisites
Restructuring of Concept to better illustrate differences to handshake based attestation and deciding factors regarding freshness properties

Subsection structure added to Terminology

Clarification of descriptions of approach (these were the FIXMEs)

Correction of RestrictionInfo structure: Added missing signature member

Changes from version 00 to version 01:

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

9. Contributors

TBD

10. References

10.1. Normative References


10.2. Informative References

[AIK-Credential]

[AIK-Enrollment]

[I-D.ietf-cbor-cddl]
[I-D.ietf-core-comi]

[I-D.ietf-sacm-coswid]

[I-D.ietf-sacm-terminology]

[IEEE1609]

[IEEE802.1AR]

[PRIRA]

[PTS]
TCG TNC Working Group, "TCG Attestation PTS Protocol Binding to TNC IF-M", 2011,

[REST]

[RFC1213]

Fuchs, et al. Expires September 13, 2019 [Page 18]


Appendix A. REST Realization

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

Appendix B. SNMP Realization

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

Design Principles:

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

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

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

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

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

B.1.1. Cycle Index

A tudaV1<Group>CycleIndex is the:

1. first index of a row (element instance or element fragment) in the tudaV1<Group>Table;

2. identifier of an update cycle on the table, when rows were added and/or deleted from the table (bounded by tudaV1<Group>Cycles);

3. binding in the tudaV1TrapV2Cycles notification for directed polling.

B.1.2. Instance Index

A tudaV1<Group>InstanceIndex is the:

1. second index of a row (element instance or element fragment) in the tudaV1<Group>Table; except for

2. a row in the tudaV1SyncTokenTable (that has only one instance per cycle).

B.1.3. Fragment Index

A tudaV1<Group>FragmentIndex is the:

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

B.2. Relationship to Host Resources MIB

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

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

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

B.3. Relationship to Entity MIB

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

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

B.4. Relationship to Other MIBs

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

B.5. Definition of TUDA MIB

<CODE BEGINS>
TUDA-V1-ATTESTATION-MIB DEFINITIONS ::= BEGIN
IMPORTS
   MODULE-IDENTITY, OBJECT-TYPE, Integer32, Counter32,
   enterprises, NOTIFICATION-TYPE
   FROM SNMPv2-SMI     -- RFC 2578
   MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
   FROM SNMPv2-CONF    -- RFC 2580
   SnmpAdminString
   FROM SNMP-FRAMEWORK-MIB; -- RFC 3411

   tudaV1MIB MODULE-IDENTITY

Fuchs, et al. Expires September 13, 2019 [Page 23]
LAST-UPDATED "201903120000Z" -- 12 March 2019

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 (2019)."

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."
 ::= { tudaV1TSACertTable 1 }

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 }
 ::= { tudaV1TSACertEntry 1 }

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

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

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

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

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

tudaV1SWIDTagCycles OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Count of SWID Tag update cycles that have occurred.
DEFVAL intentionally omitted - counter object."
::= { tudaV1SWIDTag 1 }

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

  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 TSI
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
      "TBD";
  }

  leaf tudaV1AIKCertVersionInfo {
    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.";
  }
}
"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";
}
leaf tudaV1AIKCertData {
  type binary {
    length "0..1024";
    config false;
    description "A fragment of CBOR encoded AIK Certificate data.";
  }
}

container tudaV1TSACert {
  description "TBD";
  leaf tudaV1TSACertCycles {
    type yang:counter32;
    config false;
    description "Count of TSA Certificate chain update cycles that have occurred.
    DEFVAL intentionally omitted - counter object.";
  }

  /* XXX table comments here XXX */
  list tudaV1TSACertEntry {
    key "tudaV1TSACertCycleIndex tudaV1TSACertInstanceIndex tudaV1TSACertFragmentIndex";
    config false;
    description "An entry for one fragment of TSA Certificate data.";

    leaf tudaV1TSACertCycleIndex {
      type int32 {
        range "1..2147483647";
      }
    }
  }
}
leaf tudaV1TSACertInstanceIndex {
    type int32 {
        range "1..2147483647";
    } config false;
    description
    "Middle index of this TSA Certificate fragment. Ordinal of this TSA Certificate in this chain, where the TSA Certificate itself has an ordinal of ‘1’ and higher ordinals go *up* the certificate chain to the Root CA.
    DEFVAL intentionally omitted - index object.";
}

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

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

container tudaV1SyncToken {
    description

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

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

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

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

  leaf tudaV1SyncTokenInstanceIndex {
    type int32 {
      range "1..2147483647";
    }
  }
}
leaf tudaV1SyncTokenFragmentIndex {
  type int32 {
    range "1..2147483647";
  }
  config false;
  description
  "Low-order index of this Sync Token fragment."
  "DEFVAL intentionally omitted - index object.";
}

leaf tudaV1SyncTokenIndex {
  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 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;
  }
}
Internet-Draft                    tuda                        March 2019

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

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

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

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

Fuchs, et al. Expires September 13, 2019 [Page 52]
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.
     
    "TPD"
  }
}

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

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).
     
    "TPD"
  }
}

container tudaV1TrapV2Cycles-tudaV1VerifyTokenCycles {
  description
  "TPD"
  leaf tudaV1VerifyTokenCycles {
    type yang:counter32;
    description
  }
}
Appendix D. Realization with TPM functions

D.1. TPM Functions

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

D.1.1. Tick-Session and Tick-Stamp

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

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

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

D.1.2. Platform Configuration Registers (PCRs)

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

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

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

D.1.3. PCR restricted Keys

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

D.1.4. CertifyInfo

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

D.2. IE Generation Procedures for TPM 1.2

D.2.1. AIK and AIK Certificate

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

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

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

Figure 2: TUDA-Cert element in CDDL

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

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

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

Required TPM functions:
create_AIK_Cert(...) = {
    AIK = TPM_MakeIdentity()
    IdReq = CollateIdentityRequest(AIK, EK)
    IdRes = Call(AIK-CA, IdReq)
    AIK-Cert = TPM_ActivateIdentity(AIK, IdRes)
}

/* Alternative */
create_AIK_Cert(...) = {
    AIK = TPM_CreateWrapKey(Identity)
    AIK-Cert = Call(AIK-CA, AIK.pubkey)
}

Figure 3: Creating the TUDA-Cert element

D.2.2. Synchronization Token

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

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

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

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

TimeStampToken = bytes ; RFC 3161

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

TPM-CURRENT-TICKS = [
    currentTicks: uint
    ? {
        tickRate: uint
        tickNonce: TPM-NONCE
    }
]

; Note that TickStampBlob-Output "right" can omit the values for
;   tickRate and tickNonce since they are the same as in "left"

TPM-NONCE = bytes .size 20

Figure 4: TUDA-Sync element in CDDL

Required TPM functions:
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 / bigint
; "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:

```c
    imp_att = TPM_TickStampBlob(
        keyHandle = restrictedKey_Handle, /*TPM_KEY_HANDLE*/
        antiReplay = dummyNonce,          /*TPM_NONCE*/
        digestToStamp = dummyDigest)      /*TPM_DIGEST*/
    VerifyToken = imp_att
```

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

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

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

Figure 10: Verification of Certificates

```
| ts_left = Synctoken.left
| ts_right = Synctoken.right

/* Reconstruct ts_right’s omitted values; Alternatively assert == */
| ts_right.currentTicks.tickRate = ts_left.currentTicks.tickRate
| ts_right.currentTicks.tickNonce = ts_left.currentTicks.tickNonce

| ticks_left = ts_left.currentTicks
| ticks_right = ts_right.currentTicks

/* Verify Signatures */
| verifySig(AIK_pub, dummyNonce || dummyDigest || ticks_left)
| verifySig(TSA_pub, hash(ts_left) || timestamp.time)
| verifySig(AIK_pub, dummyNonce || hash(timestamp) || ticks_right)

| delta_left = timestamp.time -
|     ticks_left.currentTicks * ticks_left.tickRate / 1000

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

Figure 11: Verification of Synchronization Token
compositeHash = hash_init()
for value in Composite.values:
    hash_update(compositeHash, value)
compositeHash = hash_finish(compositeHash)

certInfo = reconstruct_static(TPM-CERTIFY-INFO)

assert(Composite.bitmask == ExpectedPCRBitmask)
assert(certInfo.pcrinfo.PCRSelection == Composite.bitmask)
assert(certInfo.pcrinfo.digestAtRelease == compositeHash)
assert(certInfo.pubkeyDigest == hash(restrictedKey_Pub))

verifySig(AIK_pub, dummyNonce || certInfo)

Figure 12: Verification of Restriction Info

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

/* Reconstruct ts’s omitted values; Alternatively assert == */
\[ \text{ts.currentTicks.tickRate = ts_left.currentTicks.tickRate} \]
\[ \text{ts.currentTicks.tickNonce = ts_left.currentTicks.tickNonce} \]

\text{verifySig(restrictedKey_pub, dummyNonce || dummyDigest || ts)}

\text{ticks = ts.currentTicks}

\text{time_left = delta_right + ticks.currentTicks * ticks.tickRate / 1000}
\text{time_right = delta_left + ticks.currentTicks * ticks.tickRate / 1000}
\[ \text{[time_left, time_right]} \]

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.

\[
\text{TUDA-AttestationToken} = \text{TUDA-AttestationToken\_quote} / \text{TUDA-AttestationToken\_audit}
\]

\[
\text{TUDA-AttestationToken\_quote} = \text{sig(AIK-Key, TimeInfo = [ time, resetCount, restartCount ]), PCR-Selection = [ * PCR], PCR-Digest := PCRDigest}
\]

\[
\text{TUDA-AttestationToken\_audit} = \text{sig(AIK-key, TimeInfo = [ time, resetCount, restartCount ]), Session-Digest := PCRDigest}
\]

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.

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

Figure 19: TUDA-Proof element for TPM 2.0
Acknowledgements

Authors’ Addresses

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

Email: andreas.fuchs@sit.fraunhofer.de

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

Email: henk.birkholz@sit.fraunhofer.de

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

Email: blueroofmusic@gmail.com

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

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

Abstract

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

Contributing

TBD

Status of This Memo

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

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

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

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

Copyright Notice

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

Table of Contents

1. Introduction .............................................. 3
   1.1. Entity Overview ........................................ 4
   1.2. Use of CBOR and COSE ................................. 5
   1.3. EAT Operating Models ................................ 5
   1.4. What is Not Standardized .............................. 6
       1.4.1. Transmission Protocol ............................. 6
       1.4.2. Signing Scheme .................................. 7
   2. Terminology ............................................. 7
   3. The Claims ............................................... 8
       3.1. Universal Entity ID (UEID) Claim .................... 8
       3.2. Origination (origination) Claims .................... 11
       3.3. OEM identification by IEEE OUI ..................... 11
       3.4. Security Level (seclevel) Claim ................... 12
       3.5. Nonce (nonce) Claim ................................ 13
       3.6. Secure Boot and Debug Enable State Claims ....... 13
           3.6.1. Secure Boot Enabled (secbootenabled) Claim .. 13
           3.6.2. Debug Disabled (debugdisabled) Claim .......... 13
           3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot)
                  Claim ........................................... 13
           3.6.4. Debug Permanent Disable (debugpermanentdisable) 
                  Claim ........................................... 13
           3.6.5. Debug Full Permanent Disable 
                  (debugfullpermanentdisable) Claim ............... 14
       3.7. Location (loc) Claim ................................ 14
           3.7.1. lat (latitude) claim ............................. 14
           3.7.2. long (longitude) claim ........................... 14
           3.7.3. alt (altitude) claim ............................. 14
           3.7.4. acc (accuracy) claim ............................. 14
           3.7.5. altacc (altitude accuracy) claim ............... 15
           3.7.6. heading claim .................................. 15
           3.7.7. speed claim ................................... 15
       3.8. ts (timestamp) claim ................................ 15
       3.9. age claim ............................................ 15
       3.10. uptime claim ........................................ 15
       3.11. The submods Claim .................................. 16
           3.11.1. The submod_name Claim .......................... 16
           3.11.2. Nested EATs, the eat Claim .................... 16
Remote device attestation is fundamental service that allows a remote device such as a mobile phone, an Internet-of-Things (IoT) device, or other endpoint to prove itself to a relying party, a server or a service. This allows the relying party to know some characteristics about the device and decide whether it trusts the device.

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

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

- Proof of the make and model of the device hardware (HW)
Proof of the make and model of the device processor, particularly for security oriented chips

Measurement of the software (SW) running on the device

Configuration and state of the device

Environmental characteristics of the device such as its GPS location

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

1.1. Entity Overview

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

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

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

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

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

1.3. EAT Operating Models

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

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

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

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

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

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

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

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

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

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

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

1.4. What is Not Standardized

1.4.1. Transmission Protocol

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

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

1.4.2. Signing Scheme

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

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

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

2. Terminology

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

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

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

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

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

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

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

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

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

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

3. The Claims

3.1. Universal Entity ID (UEID) Claim

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

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

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

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

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

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

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

When the entity constructs the UEID, the first byte is a type and the following bytes the ID for that type. Several types are allowed to accommodate different industries and different manufacturing processes and to give options to avoid paying fees for certain types of manufacturer registrations.
<table>
<thead>
<tr>
<th>Type Byte</th>
<th>Type Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>GUID</td>
<td>This is a 128 to 256 bit random number generated once and stored in the device. The GUID may be constructed from various identifiers on the device using a hash function or it may be just the raw random number.</td>
</tr>
<tr>
<td>0x02</td>
<td>IEEE EUI</td>
<td>This makes use of the IEEE company identification registry. An EUI is made up of an OUI and OUI-36 or a CID, different registered company identifiers, and some unique per-device identifier. EUIs are often the same as or similar to MAC addresses. (Note that while devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) TODO: normative references to IEEE.</td>
</tr>
<tr>
<td>0x03</td>
<td>IMEI</td>
<td>This is a 14-digit identifier consisting of an 8 digit Type Allocation Code and a six digit serial number allocated by the manufacturer, which SHALL be encoded as a binary integer over 48 bits. The IMEI value encoded SHALL NOT include Luhn checksum or SVN information.</td>
</tr>
<tr>
<td>0x04</td>
<td>EUI-48</td>
<td>This is a 48-bit identifier formed by concatenating the 24-bit OUI with a 24-bit identifier assigned by the organisation that purchased the OUI.</td>
</tr>
<tr>
<td>0x05</td>
<td>EUI-60</td>
<td>This is a 60-bit identifier formed by concatenating the 24-bit OUI with a 36-bit identifier assigned by the organisation that purchased the OUI.</td>
</tr>
<tr>
<td>0x06</td>
<td>EUI-64</td>
<td>This is a 64-bit identifier formed by concatenating the 24-bit OUI with a 40-bit identifier assigned by the organisation that purchased the OUI.</td>
</tr>
</tbody>
</table>

Table 1: UEID Composition Types

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

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

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

3.2. Origination (origination) Claims

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

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

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

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

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

3.3. OEM identification by IEEE OUI

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

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

Note that the OUI is in common use as a part of MAC Address. This claim is only the first bits of the MAC address that identify the
3.4. Security Level (seclevel) Claim

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

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

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

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

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

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

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

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

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

3.6. Secure Boot and Debug Enable State Claims

3.6.1. Secure Boot Enabled (secbootenabled) Claim

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

3.6.2. Debug Disabled (debugdisabled) Claim

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

3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim

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

3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim

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

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

3.7. Location (loc) Claim

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

3.7.1. lat (latitude) claim

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

3.7.2. long (longitude) claim

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

3.7.3. alt (altitude) claim

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

3.7.4. acc (accuracy) claim

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

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

3.7.6. heading claim

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

3.7.7. speed claim

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

3.8. ts (timestamp) claim

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

3.9. age claim

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

3.10. uptime claim

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

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

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

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

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

3.11.1. The submod_name Claim

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

3.11.2. Nested EATs, the eat Claim

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

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

4. CBOR Interoperability

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

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

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

4.2. String Encoding (major type 2 and 3)

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

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

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

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

4.4. Date and Time

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

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

4.5. URIs

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

4.6. Floating Point

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

4.7. Other types

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

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

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

5.1.1. Claims Registered by This Document

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

TODO: add the rest of the claims in here

5.2. EAT CBOR Tag Registration

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

5.2.1. Tag Registered by This Document

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

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

6.1. UEID Privacy Considerations

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

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

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

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

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

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

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

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Examples

A.1. Very Simple EAT

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

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

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

{  
  / nonce /   11:h’948f8860d13a463e8e’,  
  / UEID /   8:h’0198f50a4ff6c05861c8860d13a638ea4fe2f’,  
  / secbootenabled / 13: true,  
  / debugpermanentdisable / 15: true,  
  / ts /   21:1526542894,  
  / seclevel / 10: 3, / secure restricted OS /  
  / submods / 30: [  
    / 1st submod, an Android Application / {  
      / submod_name / 30: 'Android App "Foo"',  
      / seclevel / 10: 1, / unrestricted /  
      / app data / -70000: 'text string'  
    },  
    / 2nd submod, A nested EAT from a secure element / {  
      / submod_name / 30: 'Secure Element EAT',  
      / eat / 31: 71( 18(  
        / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]  
      )  
    }  
  }  
  / 3rd submod, information about Linux Android / {  
    / submod_name / 30: 'Linux Android',  
    / seclevel / 10: 1, / unrestricted /  
    / custom - release / -80000: '8.0.0',  
    / custom - version / -80001: '4.9.51+'  
  }  
}
Authors’ Addresses

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

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

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

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

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

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

Proof-of-possession of a private key is useful to a relying party, but the associated signature in the Token Binding message does not provide an indication as to how the private key is stored and in what kind of environment the associated cryptographic operation takes place. This information may be required by a relying party in order to

Table of Contents

1. Introduction ................................................. 2
2. Attestation Enhancement to TLS Token Binding Message ...... 3
   2.1. KeyStore Attestation ................................. 4
       2.1.1. Verification Procedures ....................... 4
   2.2. TPMv2 Attestation .................................. 5
       2.2.1. Verification Procedures ....................... 6
3. Extension Support Negotiation ............................... 6
   3.1. Negotiating Token Binding Protocol Extensions ...... 7
4. Example - Platform Attestation for Anomaly Detection ...... 7
5. IANA Considerations ........................................ 8
   5.1. TLS Extensions Registry ............................. 8
   5.2. Token Binding Extensions for Attestation .......... 8
6. Security and Privacy Considerations ........................ 9
   6.1. Attestation Privacy Considerations ................ 9
7. Acknowledgments ............................................. 9
8. References .................................................. 9
   8.1. Normative References ................................ 9
   8.2. Informative References .............................. 10
Authors’ Addresses ............................................. 10
to satisfy requirements regarding client platform integrity. Therefore, attestations are sometimes required by relying parties in order for them to accept signatures from clients. As per the definition in [I-D.birkholz-tuda], "remote attestation describes the attempt to determine the integrity and trustworthiness of an endpoint -- the attestee -- over a network to another endpoint -- the verifier -- without direct access." Attestation statements are therefore widely used in any server verification operation that leverages client cryptography.

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

2. Attestation Enhancement to TLS Token Binding Message

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

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

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

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

2.1. KeyStore Attestation

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

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

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

\[
\text{attestation_data} = \{
\text{alg: int}, \\
\text{sig: bytes}, \\
\text{x5c: [credCert: bytes, *(caCert: bytes)]}
\}
\]

2.1.1. Verification Procedures

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

o Verify that attestation_data is in the expected CBOR format.

o Parse the first certificate listed in x5c and extract the public key and challenge. If the challenge does not match the SHA-256 hash of the token binding public key then the attestation is invalid.

o If the challenge matches the expected hash of the token binding public key, verify the sig with respect to the extracted public key and algorithm from the previous step.

o Verify the rest of the certificate chain up to the root. The root certificate must match the expected root for the device.

2.2. TPMv2 Attestation

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

\[
\text{attestation\_data} = ( \\
\text{alg: int,} \\
\text{tpmt\_sig: bytes,} \\
\text{tpms\_attest: bytes,} \\
\text{x5c: [credCert: bytes, *(caCert: bytes)]})
\]

The tpmt\_sig is generated over a tpms\_attest structure signed with respect to the certificate chain provided in the x5c array, and the algorithmic identifier corresponding to the COSE algorithm registry ([cose\_iana]). It is derived from the TPMT\_SIGNATURE data structure defined in Section 11.3.4 of [TPMv2]. tpms\_attest is derived from the TPMS\_ATTEST data structure in Section 10.2.8 of [TPMv2], specifically with the extraData field being set to a SHA-256 hash of the token binding public key.

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

The steps for verifying a Token Binding TPMv2 Attestation are:

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

3. Extension Support Negotiation

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

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

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

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

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

3.1. Negotiating Token Binding Protocol Extensions

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

4. Example - Platform Attestation for Anomaly Detection

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

5. IANA Considerations

This memo includes the following requests to IANA.

5.1. TLS Extensions Registry

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

Value: TBD

Extension name: token_binding_with_extensions

Reference: this document

Recommended: Yes

5.2. Token Binding Extensions for Attestation

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

Android Keystore Attestation:

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

TPM v2 Attestation:

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

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

6.1. Attestation Privacy Considerations

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

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

7. Acknowledgments

Thanks to Andrei Popov for his detailed review and recommendations.

8. References

8.1. Normative References

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

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

[Keystore]
8.2. Informative References

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

Authors' Addresses

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

Laurence Lundblade
Security Theory LLC

Email: lgl@island-resort.com

Mandyam, et al. Expires July 28, 2019
Use cases for Remote Attestation common encodings

draft-richardson-rats-usecases-02

Abstract

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on December 21, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
1. Introduction

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

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

This document will probably not deal with use cases from an end-user point of view, but rather on the technology verticals that wish to use RATS concepts (such as EAT) in their deployments.

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

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

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

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

2.1. Static attestations

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

2.2. Session attestations

A session attestation says something about how the shared session key was created.

2.3. Statements

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

3. Requirements Language

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

4. Overview of Sources of Use Cases

The following specifications have been covered in this document:

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

This document will be expanded to include summaries from:
5. Use case summaries

5.1. Trusted Computing Group (TCG)

The TCG is trying to solve the problem of knowing if a networking
device should be part of a network, if it belongs to the operator,
and if it is running appropriate software.

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

- "non-privacy-preserving applications (i.e., networking, Industrial
  IoT )",
- that the firmware is provided by the device manufacturer
- that there is a manufacturer installed hardware root of trust
  (such as a TPM and boot room)

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

- identify a remote box in a way that’s hard to spoof
- report the inventory of software was launched on the box in a way
  that can not be spoofed

The use case described is to be able to monitor the authenticity of
software versions and configurations running on each device. This
allows owners and auditors to detect deviation from approved software
and firmware versions and configurations, potentially identifying
infected devices.

Attestation may be performed by network management systems.
Networking Equipment is often highly interconnected, so it’s also
possible that attestation could be performed by neighboring devices.
Specifically listed to be out of scope includes: Linux processes, assemblies of hardware/software created by end-customers, and equipment that is sleepy (check term).

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

The TCG uses the following terminology:

- Device Manufacturer
- Attester ("device under attestation")
- Verifier (Network Management Station)
- "Explicit Attestation" is the TCG term for a static (platform) statement.
- "Implicit Attestation" is the TCG term for a session statement.
- Reference Integrity Measurements (RIM), which are signed by device manufacturer and integrated into firmware.
- Quotes: measured values (having been signed), and RIMs
- Reference Integrity Values (RIV)

- devices have a Initial Attestation Key (IAK), which is provisioned at the same time as the IDevID.
- PCR - Platform Configuration Registry (deals with hash chains)

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

5.2. Android Keystore system

[keystore] describes a system used in smart phones that run the Android operation system. The system is primarily a software container to contain and control access to cryptographic keys, and therefore provides many of the same functions that a hardware Trusted Platform Module might provide.
On hardware which is supported, the Android Keystore will make use of whatever trusted hardware is available, including use of Trusted Execution Environment (TEE) or Secure Element (SE)). The Keystore therefore abstracts the hardware, and guarantees to applications that the same APIs can be used on both more and less capable devices.

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

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

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

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

QUOTE: Finally, the Verified Boot state is included in key attestation certificates (provided by Keymaster/Strongbox) in the deviceLocked and verifiedBootState fields, which can be verified by apps as well as passed onto backend services to remotely verify boot integrity [**21]

5.3. Fast IDentity Online (FIDO) Alliance

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

FIDO specifications extend to various hardware second factor authentication devices.

Terminology includes:

- "relying party" validates a claim
- "relying party application" makes FIDO Authn calls
- "browser" provides Web Authentication JS API
- "platform" is the base system
o "internal authenticator" is some credential built-in to the device

o "external authenticator" may be connected by USB, bluetooth, wifi, and may be an stand-alone device, USB connected key, phone or watch.

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

A FIDO use case involves a relying party that having a attestation on the biometric system that identifies a human. It is the state of the biometric system that is being attested to, not the identity of the human.

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

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

6. Privacy Considerations.

   TBD

7. Security Considerations

   TBD.

8. IANA Considerations

   TBD.

9. Acknowledgements

10. References
10.1. Normative References


10.2. Informative References

[android_security]

[fido]

[fido_w3c]

[fidoattestation]

[fidosignature]

[fidotechnote]

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

[keystore]
Appendix A. Changes

- added comments from Guy, Jessica, Henk and Ned on TCG description.

Author’s Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca