Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

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

October 19, 2015

Time-Based Uni-Directional Attestation
draft-birkholz-tuda-00

Abstract

This memo documents the method and bindings used to conduct time-based unidirectional 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 http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Table of Contents

1. Introduction

In many contexts and scenarios it is not feasible to deploy bi-directional protocols, due to constraints in the underlying communication schemes. Furthermore, many communication schemes do not have a notion of connection, which disallows the usage of connection context related state information. These constraints may make it impossible to deploy challenge-response based schemes to achieve freshness of messages in security protocols. Examples of these constrained environments include broadcast and multicast schemes such as automotive IEEE802.1p as well as communication models that do not maintain connection state over time, such as REST [REST] and SNMP [RFC3411].

The protocols usually employed - such as the Platform Trust Service (PTS) Protocol [PTS] - for Remote Attestations using the Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG) are based upon the TPM_Quote() function. It consists of the sending of a nonce-challenge that is then used within TPM_Quote()'s signature to prove the freshness of the Attestation response. This scheme requires bi-directional communication.

This specification describes a new scheme for Remote Attestations based upon a combination of TPM_CertifyInfo() and TPM_TickStampBlob() to implement a time-based attestation scheme. The approach is based upon the work described in [MTAF] and [SFKE2008]. The freshness properties of a challenge-response based protocol define the time-frame between the transmission of the nonce and the reception of the response as the point in time of attestation. Given the time-based attestation scheme, the point in time of attestation lies within the time-frame given by the accuracy of the time-synchronization and the drift of clocks. If the point in time is within the range of the typical round-trip of a challenge response attestation, the freshness property of TUDA is equivalent to that of classic challenge response attestation. Even if the typical round-trip time is exceeded slightly, the TUDA attestation statements provide sufficiently fresh proofs for most scenarios. In contrast to classical attestations, TUDA attestations can serve as proof of integrity in audit logs with point in time guarantees. Also, it can be used via uni-directional and connection-less communication channels.

Appendix A contains a realization of TUDA using TPM 1.2 primitives. TODO: TPM 2.0 follows next year.

1.1. Terminology

This specification makes use of the terminology defined in [RFC4949].

This specification uses CDDL as defined in [I-D.greevenbosch-appsawg-cbor-cddl]. The specific data structures defined by this specification for use by other specifications are:

```
tuda = [TUDA-Synctoken, TUDA-Verifytoken, TUDA-RestrictionInfo,
        TUDA-Cert, TUDA-Measurement-Log]
```

Common types used in these are:

```
Cert = bytes ; an X.509 certificate

PCR-Hash = Hash
Hash = bytes
```

The roles used in this document are:

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

Verifier: the endpoint that consumes the attestation of another endpoint.

TSA: Time Stamp Authority [RFC3161].

TSA-CA: a Certificate Authority, that provides the certificate for the TSA.

AIK-CA: The Attestation Identity Key (AIK) is a special key type used within TPMs for identity-related operations (such as TPM_Certify or TPM_Quote). Such an AIK can be established in many ways, using either a combination of TPM_MakeIdentity and TPM_ActivateIdentity with a so-called PrivacyCA [AIK-Enrollment] or by means of TPM_CreateWrapKey, readout in a secure environment and regular certification by a custom CA similar to IDevIDs or LDevIDs in [IEEE802.1AR]. AIK-CA is a placeholder for any 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.

2. Time-Based Uni-Directional Attestation

A Time-Based Uni-Directional Attestation (TUDA) consists of the following four elements in order to gain assurance of the Attestee'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.

- 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).

- Restriction Info

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.

- 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.

- Implicit Attestation

  The actual attestation is then based upon a TPM_TickStampBlob operation using the restricted temporary key that was certified in the steps above. The TPM_TickStampBlob 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.

These elements could be sent en bloc, but it is recommended to retrieve them separately to save bandwidth, since each of these elements has different update cycles.

Furthermore, in some scenarios it might be feasible not to store all elements on the Attestee end device, but instead they will be retrieved from another location or pre-deployed to the Verifier. It may even be feasible to only store public keys at the Verifier and skip all certificate provisioning completely in order to save bandwidth and computation time for certificate verification.

When mapped to TPM1.2 (see Appendix A), one additional item is added to these five:

- AIK Certificate ([AIK-Credential], [AIK-Enrollment]; see Appendix A.2.1).

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 identity property.

2.1. Attestation Element 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 includes associated information), that an endpoint and its TPM can be in, is important to the attestation process.

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

- Some states are more volatile and change at the beginning of each boot cycle. This includes the TPM-internal Tick-Session which provides the basis for the synchronization token and implicit attestation.

- Some states are even more volatile and change during an uptime cycle (the period of time an endpoint is powered on, starting with its boot). This includes the content of PCR registers of a TPM and thereby also the PCR-restricted keys used during attestation.

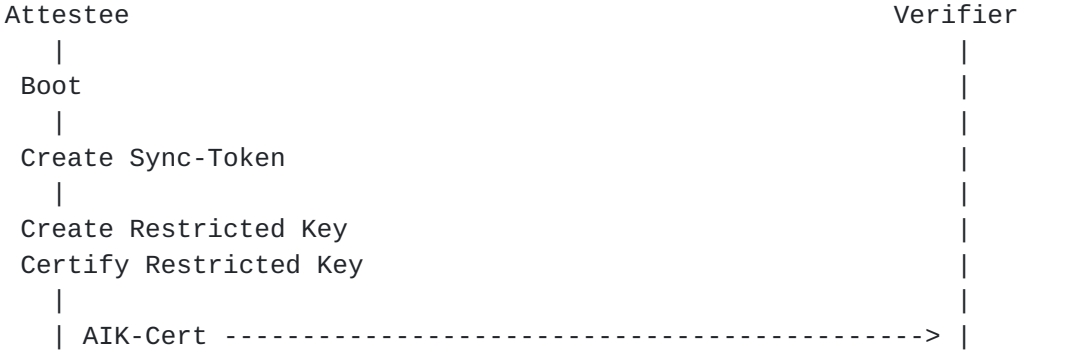
Depending on this lifetime of state, data has to be transported over the wire, or not. E.g. information that does not change due to a reboot typically has to be transported only once between the Attestee and the Verifier.

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

- The Attestee completes a boot-cycle

- A relevant PCR changes

- Too much time has passed since the last attestation statement



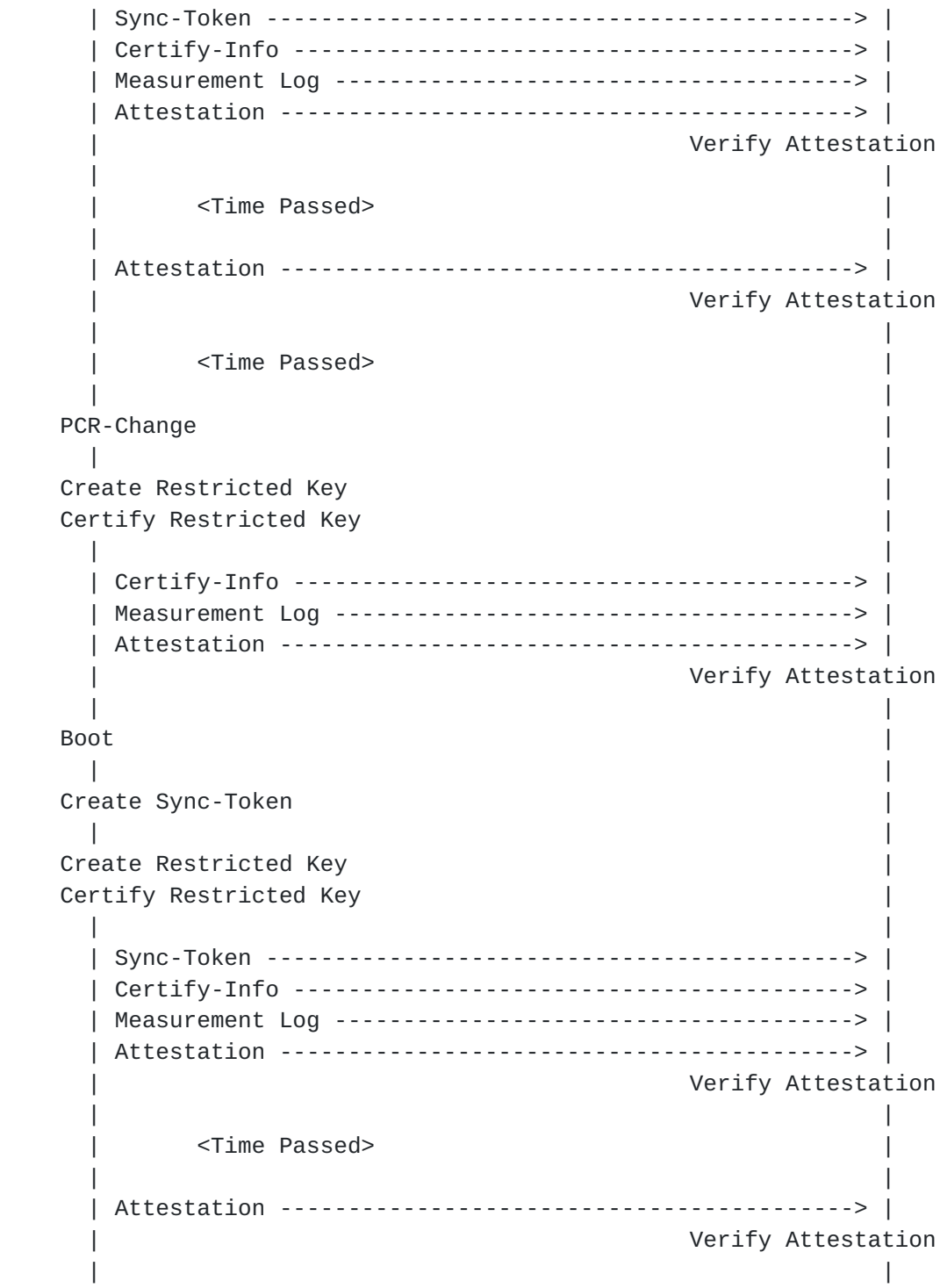


Figure 1: Example sequence of events

3. Realisation Approaches

3.1. SNMP

An SNMP MIB should be defined that encodes each of the five TUDA data items as a table with each row containing a single read-only columnar SNMP object of datatype OCTET-STRING. The values of the set of rows in each table could be concatenated to reconstitute each CBOR encoded data item. The Verifier could retrieve the values for each of these CBOR data items by using SNMP GetNext requests to "walk" each table. The Verifier could then decode each of the CBOR encoded data items according to their CDDL definitions.

Design Ideas:

(1) Over time, attestation values will age and become outside the time window (i.e., no longer fresh attestations). Using a primary table index of a cycle counter object could disambiguate the transition from one attestation cycle to the next.

(2) Over time, the measurement log information (for example) may grow quite large. To allow for more efficient data access using SNMP Get or GetBulk requests, two helper objects could be defined to point at the first and last active row in each table.

(3) Notifications could be used to indicate to a Verifier that a new cycle has occurred (i.e., the synchronization data, measurement log, etc. have been updated by deleting old table rows and adding new rows). The notification should include the cycle counter object.

A partial sketch of the proposed SNMP MIB follows:

```
TUDA-V1-ATTESTATION-MIB DEFINITIONS ::= BEGIN

IMPORTS
       MODULE-IDENTITY, OBJECT-TYPE, Integer32, Counter32, enterprises
           FROM SNMPv2-SMI;                                  -- RFC 2578
    -- SNMP macros, datatypes, and the "enterprises" root OID

tudaV1MIB MODULE-IDENTITY
       LAST-UPDATED    "201510180000Z"  -- October 18, 2015
       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) Fraunhofer Institute for
                      Secure Information Technology (2015)."

       REVISION "201510180000Z" -- October 18, 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 }

tudaV1General           OBJECT IDENTIFIER ::= { tudaV1MIBObjects 1 }

tudaV1GeneralCycles     OBJECT-TYPE
    SYNTAX              Counter32
    MAX-ACCESS          read-only
    STATUS              current
    DESCRIPTION
        "Count of TUDA attestation cycles that have occurred.

        DEFVAL intentionally omitted - counter object."
    ::= { tudaV1GeneralCycles }

tudaV1SyncToken         OBJECT IDENTIFIER ::= { tudaV1MIBObjects 2 }

tudaV1SyncTokenFirst    OBJECT-TYPE
    SYNTAX              Integer32 (0..2147483647)
    MAX-ACCESS          read-only
    STATUS              current
```

```
    DESCRIPTION
        "Low-order index of first active row of TUDA sync token data."
    DEFVAL      { 0 }
    ::= { tudaV1SyncToken 2 }

tudaV1SyncTokenLast     OBJECT-TYPE
    SYNTAX              Integer32 (0..2147483647)
    MAX-ACCESS          read-only
    STATUS              current
    DESCRIPTION
        "Low-order index of last active row of TUDA sync token data."
    DEFVAL      { 0 }
    ::= { tudaV1SyncToken 3 }

tudaV1SyncTokenTable    OBJECT-TYPE
    SYNTAX              SEQUENCE OF TudaV1SyncTokenEntry
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION
        "A table for the TUDA synchronization token data."
    ::= { tudaV1SyncToken 1 }

tudaV1SyncTokenEntry    OBJECT-TYPE
    SYNTAX              TudaV1SyncTokenEntry
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION
        "An entry for one chunk of TUDA synchronization token data."
    INDEX               { tudaV1GeneralCycles,
                          tudaV1SyncTokenIndex }
    ::= { tudaV1SyncTokenTable 1 }

TudaV1SyncTokenEntry ::= SEQUENCE {
        tudaV1SyncTokenIndex            Integer32,
        tudaV1SyncTokenData             OCTET STRING
    }

tudaV1SyncTokenIndex    OBJECT-TYPE
    SYNTAX              Integer32 (1..2147483647)
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION
        "Low-order index of this synchronization token entry.

        DEFVAL intentionally omitted - index object."
    ::= { tudaV1SyncTokenEntry 1 }

tudaV1SyncTokenData     OBJECT-TYPE
```

```
    SYNTAX              OCTET STRING (SIZE(0..1024))
    MAX-ACCESS          read-only
    STATUS              current
    DESCRIPTION
        "A CBOR encoded chunk of the synchronization token data."
    DEFVAL      { "" }
    ::= { tudaV1SyncTokenEntry 2 }

tudaV1AIKCert           OBJECT IDENTIFIER ::= { tudaV1MIBObjects 3 }

tudaV1AIKCertTable      OBJECT-TYPE
    SYNTAX              SEQUENCE OF TudaV1AIKCertEntry
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION
        "A table for the TUDA AIK certificate data."
    ::= { tudaV1AIKCert 1 }

-- etc. for remaining CBOR data items for TUDA
END
```

3.2. REST

Each of the five data items is defined as a media type (Section 4). 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 [I-D.ietf-netconf-restconf].

4. IANA Considerations

This memo includes a request to IANA. TBD

5. Security Considerations

There are Security Considerations. TBD

6. Acknowledgements

7. Change Log

(This section to be removed by the RFC editor.)

Changes from version 00 to version 01:

TBD with 01

8. Contributors

TBD

9. Informative References

[AIK-Credential]
"TCG Credential Profile", 2007,
<http://www.trustedcomputinggroup.org/files/temp/642686EC-1D09-3519-AD58BB4C50BD5028/IWG%20Credential_Profiles_V1_R1_14.pdf>.

[AIK-Enrollment]
TCG Infrastructure Working Group, "A CMC Profile for AIK Certificate Enrollment", 2011,
<https://www.trustedcomputinggroup.org/files/resource_files/738DF0BB-1A4B-B294-D0AF6AF9CC023163/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf>.

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-07 (work in progress), October 2015.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-08 (work in progress), October 2015.

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

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

[PTS] "TCG Attestation PTS Protocol Binding to TNC IF-M", 2011,
<http://www.trustedcomputinggroup.org/files/resource_files/508E7E89-1A4B-B294-D06395D5FD7EC4E7/IFM_PTS_v1_0_r28.pdf>.

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

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

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

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

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

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

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

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <http://www.rfc-editor.org/info/rfc7320>.

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

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

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

Appendix A. Realization with TPM 1.2 functions

A.1. TPM Functions

The following TPM structures, resources and functions are used within this approach. They are based upon the TPM 1.2 specification [TPM12].

A.1.1. Tick-Session and Tick-Stamp

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

It further includes an internal timer that is being initialize to Zero on each reboot. From this point on, the TPM increments this timer continuously based upon its internal secure clocking information until the device is powered down or set to sleep. By its hardware design, the TPM will detect attacks on any of those properties.

The TPM offers the function TPM_TickStampBlob, which allows the TPM to create a signature over the current tick-session and two externally provided input values. These input values are designed to serve as a nonce and as payload data to be included in a TickStampBlob: TickstampBlob := sig(TPM-key, currentTicks || nonce || externalData).

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

and within this document is to use twenty bytes of zero h'0000000000000000000000000000000000000000' as well-known value.

A.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).

A.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.

A.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.

A.2. Protocol and Procedure

A.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

A.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:

TPM_TickStampBlob: explain various inputs and applications

```
| dummyDigest = h'0000000000000000000000000000000000000000'
| dummyNonce = dummyDigest
|
| create_sync_token(AIKHandle, TSA) = {
|   ts_left = TPM_TickStampBlob(
|       keyHandle = AIK_Handle,      /*TPM_KEY_HANDLE*/
|       antiReplay = dummyNonce,     /*TPM_NONCE*/
|       digestToStamp = dummyDigest  /*TPM_DIGEST*/)
|
|   ts = TSA_Timestamp(TSA, nonce = hash(ts_left))
|
|   ts_right = TPM_TickStampBlob(
|       keyHandle = AIK_Handle,      /*TPM_KEY_HANDLE*/
|       antiReplay = dummyNonce,     /*TPM_NONCE*/
|       digestToStamp = hash(ts))    /*TPM_DIGEST*/
|
|   TUDA-SyncToken = [[ts_left.ticks, ts_left.sig], ts,
|                     [ts_right.ticks.currentTicks, ts_right.sig]]
|   /* Note: leave out the nonce and tickRate field for ts_right.ticks */
| }
```

Figure 5: Creating the Sync-Token element

A.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,
                        TPM-CERTIFY-INFO]
```

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

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

Pubkey = bytes ; do we need to expose structure here?

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

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

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

TPM-RSA-KEY-PARMS = [
  ; "size of the RSA key in bits":
  keyLength: uint
  ; "number of prime factors used by this RSA key":
  numPrimes: uint
  ; "This SHALL be the size of the exponent":
  exponentSize: null / uint / biguint
  ; "If the key is using the default exponent then the exponentSize
  ; MUST be 0" -> we represent this case as null
]
```

Figure 6: TUDA-Key element in CDDL

Required TPM functions:

```
| dummyDigest = h'0000000000000000000000000000000000000000'
| dummyNonce = dummyDigest
|
| create_Composite
|
| create_restrictedKey_Pub(pcrsel) = {
|   PCRInfo = {pcrSelection = pcrsel,
|              digestAtRelease = hash(currentValues(pcrSelection))
|              digestAtCreation = dummyDigest}
|   / * PCRInfo is a TPM_PCR_INFO and thus also a TPM_KEY */
|
|   wk = TPM_CreateWrapKey(keyInfo = PCRInfo)
|   wk.keyInfo.pubKey
| }
|
| create_TPM-Certify-Info = {
|   CertifyInfo = TPM_CertifyKey(
|       certHandle = AIK,          /* TPM_KEY_HANDLE */
|       keyHandle = wk,            /* TPM_KEY_HANDLE */
|       antiReply = dummyNonce)    /* TPM_NONCE */
|
|   CertifyInfo.strip()
|   /* Remove those values that are not needed */
| }
```

Figure 7: Creating the pubkey

A.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]
```

A.2.5. Implicit Attestation

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

This element consists only of the TPM_TickStampBlock with no nonce.

```
TUDA-Verifytoken = TickStampBlob-Output
```

Figure 8: TUDA-Verify element in CDDL

Required TPM functions:

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

Figure 9: Creating the Verify Token

A.2.6. Attestation Verification Approach

The five TUDA 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 2.1), only the corresponding code listings need to be re-executed.

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

Figure 10: Verification of Certificates

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

Figure 11: Verification of Synchronization Token

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

Figure 12: Verification of Restriction Info

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

Figure 13: Verification of Measurement Log

```
 | ts = Verifytoken
 |
 | /* Reconstruct ts's omitted values; Alternatively assert == */
 | ts.currentTicks.tickRate = ts_left.currentTicks.tickRate
 | ts.currentTicks.tickNonce = ts_left.currentTicks.tickNonce
 |
 | verifySig(restrictedKey_pub, dummyNonce || dummyDigest || ts)
 |
 | ticks = ts.currentTicks
 |
 | time_left = delta_left + ticks.currentTicks * ticks.tickRate / 1000
 | time_right = delta_right + ticks.currentTicks * ticks.tickRate / 1000
 |
 | [time_left, time_right]
```

Figure 14: Verification of Attestation Token

Authors' Addresses

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

Email: andreas.fuchs@sit.fraunhofer.de

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

Email: henk.birkholz@sit.fraunhofer.de

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

Email: blueroofmusic@gmail.com

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

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