TRUSTED® COMPUTING GROUP

S P E C I F I C A T I O N

TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0

Version 1.0
Revision 0.36
September 3, 2019

Contact: admin@trustedcomputinggroup.org

PUBLISHED

DISCLAIMERS, NOTICES, AND LICENSE TERMS

Acknowledgements

The TCG wishes to thank all those who contributed to this specification. This document builds on considerable work done in the various working groups in the TCG.

Special thanks to the members of the IWG contributing to this document.

Andreas Fuchs	Fraunhofer Institute for Secure Information Technology (SIT)
Ann T. Krieger	United States Government
Bill Sulzen	Cisco Systems
Carolin Baumgartner	Invited Expert
Charles Schmidt	The MITRE Corporation
David Challener	Johns Hopkins University, Applied Physics Lab
Greg Kazmierczak	Bright Plaza, Inc.
Guy Fedorkow	Juniper Networks, Inc.
Henk Birkholz	Fraunhofer Institute for Secure Information Technology (SIT)
Ira McDonald	Invited Expert
Ken Goldman	IBM
Lisa Lorenzin	Invited Expert
Mark Baushke	Juniper Networks, Inc.
Monty Wiseman	General Electric
Ned Smith	Intel Corporation
Suzanne Leicht	United States Government
Tom Laffey	Hewlett Packard Enterprise

CONTENTS

1 Glossary

Attester	A platform or platform component that provides evidence to a Verifier as to its state.
Binding Specification	A specification describing how the information described in this specification is transmitted between Attesters and Verifiers.
Verifier	A system that analyzes evidence from an Attester to determine the Attester’s state.

Table 1: Glossary

2 Introduction

2.1 Scope

Endpoint integrity and corresponding attestation evidence is critical to many use cases. DICE[3][7][9], TPM[1] and platform specification[8] were designed to provide information—evidence—helpful for Verifiers to determine the state of a platform—the Attester. This TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0 specification provides the information elements used by Verifiers. Not all of the information is required by every Verifier.

The Trusted Attestation Protocol (TAP) Information Model is protocol neutral. It may be used in protocols that grant access to networks via VPN, SSH, or other protocols. TAP provides the basic information that is needed to enable an appraisal of endpoint integrity based on TPM PCR values or via DICE signature. It does not define how the information is provided to the Verifier. The information elements provide the flexibility necessary for TAP to be used with TPM 1.2 or 2.0, as well as with two-way or one-way protocols.

Although this document defines an information model, corresponding information elements are illustrated in Type-Length-Value (TLV) format. This is not meant to specify how the elements are actually transferred between an Attester and a Verifier, but is intended to be used as examples. The protocol specifications themselves will specify the actual format (encoding of data in motion) that will be conveyed between an Attester and a Verifier.

2.2 Audience

This TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0 specification is intended to be used in the creation of binding specifications. This Information Model document identifies the information elements that are exchanged and the general functionality that their bindings must support. It also includes other requirements that constrain these bindings to ensure that they all meet the common goals of TAP. While the TCG will provide standards for some TAP bindings, non-TCG TAP binding are possible. In consequence, this document is designed to be extensible.

This specification is also useful to enterprise architects in designing their endpoint integrity assessment capabilities. Because all TAP bindings support a common set of features and information elements, an architect designer can incorporate TAP into their design in a manner that is agnostic to the specific binding employed.

Lastly, the designer of a Verifier will find the TAP specification useful, as it provides the information elements that must be provided to the Verifier (although the format of that data is specified in the binding specification).

2.3 Relation to PTS 1.0

The TAP specification is a redesign of PTS 1.0 and is not backward compatible with PTS 1.0.

TAP adds support for DICE and TPM 2.0, including several new capabilities not available in older TPM versions. As TPM 1.2 remains widely deployed and many environments will contain a mix of TPM 1.2 and TPM 2.0 platforms for the near future, TAP also supports interactions with TPM 1.2. Due to its more limited capabilities, TPM 1.2 platforms will not be able to support all TAP functions.

2.4 Keywords

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119[2]. This specification does not distinguish blocks of informative comments and normative requirements. Therefore, for the sake of clarity, note that lower case instances of must, should, etc. do not indicate normative requirements.

3 Purpose of TAP Information Model

This document describes and specifies the TAP Information Elements, which MAY be conveyed between an Attester and a Verifier. It also describes the format of those information elements when they are being processed by a Verifier. It does not describe the methods or protocols that convey the information. Binding Specifications will specify which Information Elements are Mandatory for a protocol.

The number of information elements is intended to be sufficient to allow protocols to provide assurance to a Verifier. The data needs to have several basic characteristics, namely that it be:

Accurate	The data has to be an accurate representation of the reported information element. In many cases, it must be provable that the data is "fresh", not a replay of data that was previously created and is no longer relevant.
Interpretable	The data has to be meaningful to the Verifier for determining the state of the system.
Attributable	It must be possible to attribute the data to a particular device.

Table 2: Data Characteristics

Every attestation solution comprised of the information elements that are defined in this specification has each of these characteristics. These characteristics are enforced in different ways for different use cases and devices. Since each individual information element may be separately reported, each element is tagged, so that the Verifier can easily analyze the data.

Additionally, it is up to the protocol specifications to provide a means to mitigate against an Asokan attack[2].

4 Information Elements

This section defines the various information elements that may be assembled into an attestation report: evidence conveyed from an Attester to a Verifier. A type-length-value (TLV) notation is the format used to illustrate examples. This document uses the notation: TYPE || Length || (Value) to describe a TLV – each element is delimited by a || character. The Type is 1 byte, the Length is normally 4 bytes, giving the number of bytes of the following Value. The Value in this case may be (and often is) a concatenation of various parameters, and is therefore in parentheses. This TLV representation used in this specification is not required in binding specifications, but rather used to illustrate the contents of information elements required in examples in this document. In the Value description, sometimes the TPM 2.0 Name is used instead of the TPM 2.0 Type, for the sake of readability. If TLV is used to transmit information, it MUST use the format and parameters defined in this Specification.

4.1 Information Element Identifier (1 BYTE)

The Information Elements are:

Information Element Type Identifier	**Description**
0x00	TAP Information Model Specification Version
0x01	AK certificate
0x02	Attestation of a TPM 2.0 signing key for implicit attestation
0x03	PCRs and their values for TPM 1.2
0x04	PCRs and their values for TPM 2.0
0x05	PCR log values
0x06	Freshness attestation element
0x07	Nonce qualification information
0x08	TPM 2.0 Clock Time Certification
0x09	PCR attestation element
0x0A	Signature using Signing Key
0x0B	Previous Hibernation Report
0x0C	Supplementary Log Report
0x0D	Attestation of a DICE signing key for implicit attestation

Table 3: Information Element Identifier

4.2 Specification Version

The specification version is 2 BYTES.

4.2.1 Description

The Specification Version identifies the version of the TAP Information Model specification. The first byte indicates the Major Number of the Specification Version, the second byte, the Minor Number.

4.2.2 TLV Format

0x00 || 0x00000002 || (Value)

Where Value MUST consist of the Major and Minor numbers of the TAP Information Model Specification.

4.2.3 Example

In this example, the current Major Number is 2, and the current Minor Number is 0. The TLV is a total of 7 bytes:

0x00 || 0x00000002 || (0x0200)

4.3 X509 Certificate Chain of Attestation Key (AK)

4.3.1 Description

The AK certificate chain contains the certificate of the AK used to attest to the PCR values and any other certificates necessary for the Verifier to validate the AK. It is a sequence of bytes of DER encoded X.509 Certificates.

The Value MUST consist of the number of certificates to follow, followed by a list of tuples comprising the length of each certificate and the corresponding certificate

Note: Generally, this list starts with the AK cert, and is ordered back to the common trust anchor.

4.3.2 TLV Format

0x01 || length || number of certificates to follow (2 bytes) ||

(DER encoded first certificate ||

DER encoded second certificate ||

…)

4.3.3 Example

In this example there are 3 certificates, with lengths 0x00000100, 0x00000101, and 0x00000102. The three certificates have values:

cert1 = 0xABCD…cert2: 0x012...cert3: 0xA1B2...

The resultant TLV in this example is:

0x01 || length || 0x00000003 || (0x00000100 || 0xABC...|| 0x00000101 || 0x012...|| 0x00000102 || 0xA1B2...)

4.4 Attestation of TPM 2.0 Signing Key used for Implicit Attestation

4.4.1 Description

Note: This information element should be used only if either the only PCRs to which the key is locked are ones remeasured after a hibernation cycle, or the platform being attested is known to not go through hibernation cycles.

An attestation signing key used for implicit attestation is an unrestricted or restricted signing key that has a policy that only allows it to sign when the system is in a particular state. This information element contains the data returned from the TPM when using the AK to certify a Signing Key with a TPM2_Certify command. This information element MUST consist of the public data of the Signing Key and the result of a TPM2_Certify over that key signed by the AK. The result of TPM2_Certify (per the TPM 2.0 specification Part 3, Section 18.2) is

Type	Name	Description
TPM2B_ATTEST	certifyInfo	the structure that was signed

TPMT_SIGNATURE	Signature	the asymmetric signature over certifyInfo using the key referenced by signHandle

The certifyInfo parameter contains the policy of the key (TPM 2.0 Part 1: Session-Based Authorizations) This policy is derived exclusively from execution of a TPM2_PolicyPCR command, which restricts the usage of the key to when the PCRs are in a particular state. In order for the Verifier to determine to which PCRs and values the usage of the key is locked, this information element will contain the PCRs (and their values) to which the key is locked by policy. In order to make the policy easy to interpret by the Verifier, the key's policy will only contain this TPM_PolicyPCR value, which is calculated by:

$\text{policyDigest}_{\text{new}} := H_{\text{policyAlg}}(\text{policyDigest}_{\text{old}} \mid\mid \text{TPM\_CC\_PolicyPCR} \mid\mid \text{pcrs} \mid\mid \text{digestTPM})$

where $\text{policyDigest}_{\text{old}}$ is the initial state of all zeros. From the TPM 2.0 specification Part 2, Section 10.12.1 TPMS_QUOTE_INFO:

Type	**Name**	**Description**
TPML_PCR_SELECTION	pcrSelectionOut	the PCR in the returned list
TPML_DIGEST	pcrValues	the contents of the PCR indicated in pcrSelect as tagged digests

4.4.2 TLV Format

0x02 || length || (pcrUpdateCounter || TPML_PCR_SELECTION || TPML_DIGEST || TPM2B_ATTEST || TPMT_SIGNATURE)

4.5 TPM 1.2 PCR Values

4.5.1 Description

This information element MUST contain a list of requested PCRs and their current values. Each PCR value will be 20 bytes long because SHA1 is the only hash algorithm used by TPM 1.2. The Value in the TLV consists of a BYTE which contains the number of PCRs reported followed by a list of tuples comprising the number of each PCR and its corresponding value. (If this information element is sent by a Verifier to an Attester, to identify which PCRs are requested, the PCR Values MAY be all zeros.)

4.5.2 TLV Format

```
0x03 || length (4 bytes) || (
    number of PCRs (1 byte) ||
    first PCR (1 byte) || value of first PCR ||
    second PCR (1 byte) || value of second PCR ||
    ....
)
```

4.5.3 Example: Policy Digest when three PCRs are chosen

In this example, the values are:

PCR0 = 0x0000000000000000000000000000000000000000
PCR2 = 0x2222222222222222222222222222222222222222

PCR17 = 0x1717171717171717171717171717171717171717

The resultant information element in TLV format is:

(0x03 || 0x00000040 || 0x03 ||
0x00 || 0x0000000000000000000000000000000000000000 ||
0x02 || 0x2222222222222222222222222222222222222222 ||

0x17 || 0x1717171717171717171717171717171717171717)

4.6 TPM 2.0 PCR Values

4.6.1 Description

This information element MUST list requested PCRs and their values. It need not contain all PCR values. However, if the Verifier is concerned about hibernation, it will need all PCRs and values that have been extended. This element provides the data derived from executing a TPM2_PCR_Read command, which according to the TPM 2.0 specification, Part 3 response table contains:

Type	Name	Description
UINT32	pcrUpdateCounter	the current value of the PCR update counter
TPML_PCR_SELECTION	pcrSelectionOut	the PCR in the returned list
TPML_DIGEST	pcrValues	the contents of the PCR indicated in pcrSelect as tagged digests

(If this information element is sent by a Verifier to an Attester, to identify which PCRs are requested, the PCR Values MAY be all zeros.)

4.6.2 TLV Format

(0x04) || length || (pcrUpdateCounter || TPML_PCR_SELECTION || TPML_DIGEST)

4.6.3 Example

In this example the values to be reported are:

PCR0 = 000...0000; PCR2 = 222....222

pcrUpdateCounter = 0x00001234

TPML_PCR_SELECTION	0x00000002 0x000B 03 0b00000000 00000000 00000101
TPML_DIGEST	0x00000002 0x0040

0x00000000000000000000000000000000000000000000000000000000000000000000000000000000

0x22222222222222222222222222222222222222222222222222222222222222222222222222222222

The resultant information element in TLV format is:

0x04 || 0x0000004C || (0x00001234 || 0x00000002 || 0x000B || 03 || 0b00000000 || 00000000 || 00000101 ||

0x00000002 || 0x00000040 ||

0x00000000000000000000000000000000000000000000000000000000000000000000000000000000 ||

0x22222222222222222222222222222222222222222222222222222222222222222222222222222222)

4.7 PCR Log Values

4.7.1 Description

The PCR Log Values Information Element MUST contain a log presumed to contain all values extended into PCRs reported in section 4.5 or 4.6.

Note: log values do not usually contain values for PCRs that are not requested.

The PCR log values are not necessarily in canonical format, as different platform specifications may differ in how they report those log values.

4.7.2 TLV Format

0x05 || length (8 bytes) || (log)

Note: PCR Log values may exceed a 4 BYTE length, so for this TLV value, the length is given in 8 BYTEs.

4.8 Freshness of Attestation

4.8.1 Description

This specification provides three different means of proving the freshness of an attestation.

One of the following freshness description indicators MUST be used:

Freshness Indicator	**Description**
0x0000	A nonce provided by the Verifier is included in the attestation. This freshness element indicator is followed by 2 BYTES indicating the nonce size, followed by the nonce.
0x0001	A nonce provided by a trusted third party is included in the attestation. This freshness indicator is followed by 2 BYTES indicating the nonce size, followed by the nonce. If this freshness indicator is used, then proof of provenance of the nonce MUST be provided as in section 4.9.
0x0002	Proof of freshness is derived from the TPM clock value which is part of the signed data If this freshness element indicator is used, then proof that TPM Clock can be relied upon MUST be provided as in section 4.10.

Table 4: Freshness Indicator

4.8.2 TLV Format

0x06 || 0x00000002 || (freshness element indicator)

4.8.3 Example: Nonce provided by Verifier

0x06 || 0x00000002 || (0x0000)

4.9 Nonce Qualification Information

4.9.1 Description

Nonce qualification information is metadata provided with a nonce that is used to prove the nonce is "recent", if it was not provided by the Verifier. It may be a Time Stamp, if the nonce is a hash of the time stamp. It may be a reference to a Verifier and time, if the Verifier is known to provide nonces periodically associated with time.

This specification defines two qualification numbers, one of which MUST be used:

Qualification Number	Description
0x0000	The nonce is the hash of a time stamp. The qualification number is followed by the time stamp [5][6].
0x0001	The qualification information refers to a URL and time from which the nonce was obtained. The qualification number is followed by a representation of the URL and a representation of a time. The URL is represented as 2 bytes for the size of the URL, followed by the URL. The time is represented as 64 bytes representing the number of seconds since 0000:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970.

Table 5: Qualification Number

4.9.2 TLV Format

0x07 || length || (Qualification Number || Data)

4.9.3 Example 1: Nonce is Hash of Time Stamp

The nonce is the hash of a 512-byte time stamp:

0x07 || 0x00000212 || (0x0000 || 512-byte time stamp[5][6])

4.9.4 Example 2: Nonce is obtained from Server

The nonce was obtained from a known server at URL nonceServer.mycompany.com, 0x59B17B16 seconds since Jan 1, 1970 (midnight UTC/GMT) (8 bytes):

0x07 || length || 0x0001 || (0x19 || "nonceServer.mycompany.com" || 00000059B17B16)

Note: It is up to the Verifier to verify that the server at this IP address is a trusted server during verification, and that it provided the nonce in question recently.

4.10 TPM 2.0 Clock Time Certification

4.10.1 Description

This Information Element is not available with TPM 1.2.

A Clock Time certification MAY be used when it is impossible for the Attester to obtain a nonce from a Verifier. Clock Time certification is provided that can be used (together with the TPM's Reset Counter), to prove that the Clock time (always present in TPM 2.0 attestation) is accurate. Given t_1 and t_3 are provided by Trusted Time Stamper and t_2 is provided by TPM clock, $t_1<t_2<t_3$, $t_2 = ½ (t_3 + t_1) +/- ½ (t_3 - t_1)$.

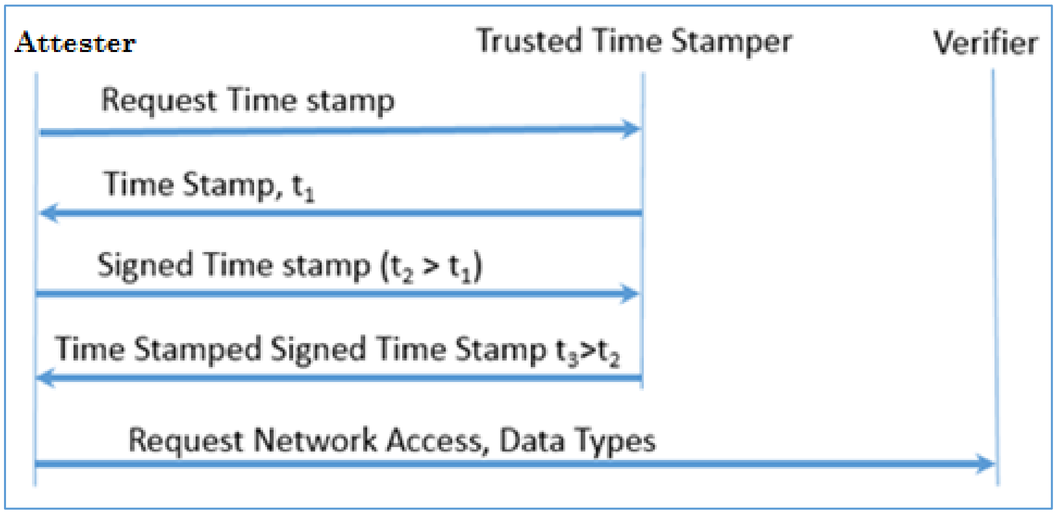
Attester
Trusted Time Stamper
Verifier
Request Time stamp
Time Stamp, t_1
Signed Time stamp ($t_2 > t_1$)
Time Stamped Signed Time Stamp $t_3>t_2$
Request Network Access, Data Types

Figure 1: TPM 2.0 Clock Time Certification

The composition work-flow of time-based uni-directional attestation (TUDA) information elements [4] where TSA is a Time Stamping Authority, is:

- Attester to TSA: Send application/timestamp-query request ([5], Section 3.4) with a payload of TimeStampReq ([5], Section 2.4.1). Nonce is set to zero-length.
- TSA to Attester: Send application/timestamp-response response ([5], Section 3.4) with a payload of TimeStampResp ([5], Section 2.4.2). Call this TimeStampLeft.
- Call TPM2_GetTime (pre TPM 2.0 revision 138, TPM2_Quote without any PCR being selected) with qualifyingData := hash(TimeStampLeft).
- TPM-Response: is timeInfo and signature. Call this TPM-TickStamp := timeInfo || signature.
- Attester to TSA: Send application/timestamp-query request ([5], Section 3.4) with a payload of TimeStampReq ([5], Section 2.4.1) Nonce is set to hash(TPM-TickStamp).
- TSA to Attester: Send application/timestamp-response response ([5], Section 3.4) with a payload of TimeStampResp ([5], Section 2.4.2). Call this TimeStampRight.

4.10.2 TLV Format

0x08 || length (4bytes) || (TimeStampLeft || TickStamp || TimeStampRight)

4.11 Explicit Attestation

4.11.1 Description

There are multiple attestation subtypes of Explicit Attestation Elements to support TPM 1.2 and TPM 2.0 Family.

There are two Explicit Attestation subtypes for TPM 1.2:

1. Quote using TPM_Quote
2. Quote using TPM_Quote2

There are three Explicit Attestation subtypes for TPM 2.0:

1. Quote of Audit Session using Nonce
2. Quote of Audit Session using Clock
3. Quote using TPM2_Quote

4.11.1.1.1 Caveats for use of TPM2_Quote for Explicit Attestation

The attestation variety using TPM2 Family TPM2_Quote is compatible with the two TPM 1.2 Family attestations and may be simpler to provision in a heterogeneous environment with both TPM 1.2 and TPM 2.0. However, the TPM2_Quote command does not return the quoted PCR values or the pcrCounter value within an atomic operation. Absent the PCR values within an atomic attestation operation, it is possible for an extend operation to occur between the retrieval of the PCR values and the TPM2_Quote, resulting in a verification failure (which is likely a false failure – but a failure nonetheless). Additionally, having the pcrCounter in an atomic attestation operation is necessary for detection of certain platform operations such as platform hibernation. TPM2_Quote also does not support attestation of NV PCRs.

4.11.1.1.2 Explicit Attestation using an Audit Session

An attestation using either of the Explicit Attestations using audit sessions MUST include the following information:

- pcrCounter value
  - useful if a hibernation session is being used.
  - useful for matching the log values to the PCR values.

- Actual PCR values
  - useful if more than one PCR has been extended between the quote and the reading of the log.
  - useful in analysis/debugging if the PCR values and the log values do not match.

The audit session MAY be used to quote NV indexes used as PCRs.

A design using an audit session for attestation is more extensible than an Explicit Attestation using TPM2_Quote, which allows future proofing, should the design later require more information to be requested and attested (e.g. TPM capability values).

4.11.1.1.3 Prerequisites for Audit Session based Attestations

In order to use audit session style attestation, the TPM2_GetSessionAuditDigest [1] command is required at the end of an audit session. The TPM2_GetSessionAuditDigest command requires Endorsement Hierarchy authorization, either through the EH password or through satisfying the EH policy. The following policy component, if put in the EH policy, allows anyone to use the TPM2_GetSessionAuditDigest command. The command TPM2_SetPrimaryPolicy sets the EK policy. Any of the policy components from the table below will enable the use of TPM2_GetSessionAuditDigest with EH authorization. The choice of policy component from Table 6 depends on the desired hash algorithm.

Policy	**SHA1**	**SHA256**	**SHA384**	**SHA512**

PolicyCommandCode set to TPM_GetSessionAuditDigest.
 	
Start with all zeros, of the length of the hash algorithm, and extend with 0x0000016c 0x0000014D.
	`f5a92c8b`
	`0192eac3`
	`493f3083`
	`6d711af1`
	`400e9c4a`
		`3684cf78`
		`5bfe9a2b`
		`0be276ef`
		`8d7523ef`
		`5ec75181`
		`1244ab1b`
		`e296a7d0`
		`89778c09`
			`8f5e5e87`
			`a7cc0982`
			`c3cc4ccd`
			`adde6e79`
			`b3a9927c`
			`b51a814f`
			`5b79835c`
			`333db7c9`
			`72caceb5`
			`f0208575`
			`4db25572`
			`575bb6a3`
				`54fe4149`
				`327454b3`
				`899a6408`
				`11b1688f`
				`07e27107`
				`12704c4e`
				`bec14477`
				`61c9c6f7`
				`a7c19cb8`
				`7da4b334`
				`af08ab74`
				`e770b259`
				`65cda5f6`
				`c4bdbcb2`
				`8c214a9a`
				`91b80c23`

Table 6: Policy Component for Hash Algorithm

4.11.2 TLV Format

The subtype of attestation used MUST be identified by the first octet within the TLV's Value field.

0x09 || length (4 bytes) || (subtype (1 BYTE) || attestation)

The attestation subtype MUST be one of the following:

TPM Family	Subtype	Value
1.2	Explicit Attestation using TPM_Quote	0x00
1.2	Explicit Attestation using TPM_Quote2	0x01
2.0	Explicit Attestation using Audit Session and Nonce	0x02
2.0	Explicit Attestation using Audit Session and Clock	0x03
2.0	Explicit Attestation using TPM2_Quote	0x04

Table 7: Subtype for Explicit Attestation

4.11.3 TPM 1.2 Explicit Attestation using TPM_Quote

4.11.3.1 Description

The information in attestation subtype 0x00 MUST be produced by a TPM v1.2 TPM_Quote (Section 16.4 in 1.2 TPM Specification) command.

4.11.3.2 TLV Format

Explicit Attestation subtype 0x00 is followed by the sequence of bytes that represents a TPM_Quote output as follows:

0x09 || length (4 bytes) || (0x00 || pcrData || sigSize || sig)

The table below illustrates the data returned by TPM_Quote

Type	Name	Description
TPM_PCR_COMPOSITE	pcrData	A structure containing the same indices as targetPCR, plus the corresponding current PCR values.
UINT32	sigSize	The used size of the output area for the signature
BYTE[ ]	Sig	The signed data blob.

4.11.4 TPM 1.2 Explicit Attestation using TPM_Quote2

4.11.4.1 Description

The information in attestation subtype 0x01 MUST be produced by a TPM 1.2 TPM_Quote2 command (Section 16.5 in 1.2 TPM Specification) command.

Explicit Attestation subtype 0x01 is followed by the sequence of bytes which represents the TPM_Quote2 outputs as follows:

Type	Name	Description
TPM_PCR_INFO_SHORT	pcrData	The value created and signed for the quote
UINT32	versionInfoSize	Size of the version info
TPM_CAP_VERSION_INFO	versionInfo	The version info
UINT32	sigSize	The used size of the output area for the signature
BYTE[ ]	Sig	The signed data blob.

4.11.4.2 TLV Format

0x09 || length (4 bytes) || (0x01 || pcrData || versionInfoSize || versionInfo || sigSize || sig)

4.11.5 TPM 2.0 Explicit Attestation using Audit Session and Nonce

4.11.5.1 Description

The information in Explicit Attestation subtype 0x02 MUST be produced using a TPM 2.0 audit session with a nonce to attest to PCR values. The nonce is extended into a PCR which is then reported in a TPM2_PCR_Read command before (or together with) any other TPM2_PCR_Read commands are executed in the session.

The Explicit Attestation subtype 0x02 is followed by a list of elements, each element comprising a size, command Code, input, and output of each command sent to the TPM 2.0. The list is followed with the audit information that was signed, followed by the signature itself.

For reference, a TPM 2 Family general-purpose audit session includes:

- Number of commands
- FirstCommandData
    - Number inputs
    - Inputs
    - cpHash
    - Number outputs
    - Outputs

    - rpHash
- SecondCommandData
    - Number inputs
    - Inputs
    - cpHash
    - Number outputs
    - Outputs
    - rpHash
    - ...
- auditInfo
- Signature

The auditInfo is the extended hash formed by extending all the cpHash and rpHash values of the commands executed in a session in the sequence in which they were executed. The auditInfo is signed by a TPM2_GetSessionAuditDigest command. The cpHash and rpHash are hashes formed from the inputs and outputs of the executed commands. These inputs and outputs, (per the TPM 2.0 Library Specification, Part 3, Section 22.4) from executing TPM2_PCR_Read are as follows:

Inputs:

Type	Name	Description
TPML_PCR_SELECTION	pcrSelectionIn	The selection of PCR to read

Outputs:

Type	Name	Description
UINT32	pcrUpdateCounter	the current value of the PCR update counter
TPML_PCR_SELECTION	pcrSelectionOut	the PCR in the returned list
TPML_DIGEST	pcrValues	the contents of the PCR indicated in pcrSelect as tagged digests

The result of executing TPM2_GetSessionAuditDigest (from TPM2.0 Library Specification, Part 3, Section 18.5) is:

Type	Name	Description
TPM2B_ATTEST	auditInfo	the audit information that was signed
TPMT_SIGNATURE	Signature	the signature over auditInfo

In verifying an audit quote, the audit digest is regenerated by extending the cpHash and rpHash values of each command into an initial value which is composed of all zeros, and is the length of session hash.

The cpHash (defined in Part 1 of the TPM 2.0 Library Specification, Section 18.7) is calculated as

cpHash :=$H_{sessionAlg}$ (commandCode {|| Name1 {|| Name2 {|| Name3 }}} {|| parameters }) (a)

where:

$H_{sessionAlg}$	hash function using the algorithm selected for the session when it was initialized
commandCode	command code for the command
Name1	unique identity of the entity associated with the first handle
Name2	unique identity of the entity associated with the second handle
Name3	unique identity of the entity associated with the third handle
parameters	remaining command parameters

For TPM2_Read_PCR, the commandCode is 0x0000017E, and the parameters are given by the table found in Part 3 of the TPM 2.0 Library Specification, Part 3, in section 22.4.2 as:

Type	Name	Description
TPML_PCR_SELECTION	pcrSelectionOut	the PCR in the returned list

Calculating rpHash is given in Part 1 of the TPM 2.0 specification in section 18.8 as:

rpHash := $H_{sessionAlg}$ (responseCode || commandCode {|| parameters})

where the responseCode will always be 0x00000000 and the parameters will be as defined by the table below.

Type	Name	Description
UINT32	pcrUpdateCounter	the current value of the PCR update counter
TPML_PCR_SELECTION	pcrSelectionOut	the PCR in the returned list
TPML_DIGEST	pcrValues	the contents of the PCR indicated in pcrSelect as tagged digests

4.11.5.2 **Example**

When requesting the PCR values for PCRs 0-7, 10, 16, with algID = SHA-256, and resulting values of

PCR0 = 0000000000000000000000000000000000000000000000000000000000000000

PCR1 = 0101010101010101010101010101010101010101010101010101010101010101

...

PCR10 = 0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A

PCR16 = 123456789ABCDEF123456789ABCDEF123456789ABCDEF123456789ABCDEF

pcrUpdateCounter = 0x0000111

Then

cpHash = SHA-256(0x0000017E || count=10=0x0000000A || algID=0x000B || sizeOfSelect=0x03 || 0b00000001 00000100 11111111)

and

rpHash = SHA-256(0x00000000 || 0x0000017E || count=10=0x0000000A || algID=0x000B || sizeOfSelect=0x03 || 0b00000001 00000100 11111111 || count=10=0x0000000A ||

0000000000000000000000000000000000000000000000000000000000000000 ||

0101010101010101010101010101010101010101010101010101010101010101 ||

...

0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A ||

123456789ASCDEF123456789ASCDEF123456789ASCDEF123456789ASCDEF)

auditInfo should be:

SHA-256(

SHA-256(
0x000000000000000000000000000000000x00000000000000000000000000000000 || cpHash1 || rpHash1
) || cpHash2 || rpHash2)

Note: during most protocol exchanges, cpHash1 and cpHash2 will not change because once the PCR values being queried are determined they will not change. But rpHash1 and rpHash2 will change as they return the PCR values and include the nonce.

auditInfo is then used in forming the TLV, along with the signature which is returned by the TPM2_GetSessionAuditDigest command.

4.11.5.3 **TLV Format**

0x09 || length (4 bytes) || (0x02 || length of command || command code (e.g. 0x0000017E) || pcrSelectionIn || pcrUpdateCounter || pcrSelectionOut || pcrValues || auditInfo || signature)

4.11.6 TPM 2.0 Explicit Attestation using Audit Session and Clock

4.11.6.1 Description

The information in attestation subtype 0x03 MUST be produced using a TPM 2.0 quote of an audit session with a clock value to attest to PCR values. The quote is returned from a TPM2_GetSessionAuditDigest command.

Explicit Attestation subtype 0x03 is followed by a list of quadruples consisting of size, commandCode, input, output. This list is then followed with the audit information, followed by the signature itself.

The first command in the audit session associates the clock with the timer value, using the TPM2_ReadClock command. The next command (or commands) in the audit session is the TPM2_PCR_Read command.

Note: Depending on the TPM's buffer size, and the number of PCRs being read, it is possible that multiple TPM2_PCR_Read commands may need to be executed

4.11.6.2 TLV Format

The TLV is:

0x09 || length (4 bytes) || (0x03|| length of command || command code (i.e. 0x00000181) || currentTime || length of command || command code (i.e. 0x0000017E) || input pcrSelectionIn || output pcrUpdateCounter || pcrSelectionOut || pcrValues || … || auditInfo || signature)

The various parameters are taken from the TPM 2.0 Part 2 specification and are defined as having the following structures:

Type	Name	Description
TPMS_TIME_INFO	currentTime	—

Next, TPM2_PCR_Read is executed to read out any PCR values of interest, and all inputs and outputs recorded. This command may have to be repeated multiple times if the TPM buffer is too small to return all the data at one time. The result of TPM2_PCR_READ comes from the TPM 2.0 Specification Part 3.

Type	Name	Description
TPML_PCR_SELECTION	pcrSelectionIn	The selection of PCR to read

UINT32	pcrUpdateCounter	the current value of the PCR update counter
TPML_PCR_SELECTION	pcrSelectionOut	the PCR in the returned list
TPML_DIGEST	pcrValues	the contents of the PCR indicated in pcrSelect as tagged digests

Lastly, a TPM2_GetSessionAuditDigest command is executed for the session using the AK. All outputs of the TPM2_GetSessionAuditDigest necessary to evaluate the resultant signature are recorded, as in TPM 2.0 Library Specification Part 3, namely:

Type	Name	Description
TPM2B_ATTEST	auditInfo	the audit information that was signed
TPMT_SIGNATURE	Signature	the signature over auditInfo

The cpHash for TPM2_ReadClock is always

cpHash1 = SHA-256(0x00000181)

and the rpHash is

rpHash1 = SHA-256(0x00000000 || 0x00000181 || TPMS_TIME_INFO)

4.11.6.3 Example

Assume PCRs 0-10 are read with result:

PCR0 = 0x0000000000000000000000000000000000000000000000000000000000000000
PCR1 = 0x0101010101010101010101010101010101010101010101010101010101010101
…
PCR10 = 0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A
pcrUpdateCounter = 0x0000111

Then

cpHash2 = SHA-256(0x0000017E || count=11=0x0000000B|| algID=0x000B || sizeOfSelect=0x03 ||
0b00000000 00000100 11111111)

and

rpHash2=SHA-256(0x00000000 || 0x0000017E || count=9=0x00000009 || aldID=0x000B || sizeOfSelect=0x03 ||
0b00000000 00000100 11111111 || count=9=0x00000009 ||
0x0000000000000000000000000000000000000000000000000000000000000000 ||
0x0101010101010101010101010101010101010101010101010101010101010101 ||
…
0x0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A)

Now auditInfo is:

SHA-256(

SHA-256(

0x00000000000000000000000000000000x00000000000000000000000000000000 || cpHash1 || rpHash1

) || cpHash2 || rpHash2

)

4.11.7 TPM 2.0 Explicit Attestation using TPM2_Quote

4.11.7.1 Description

Attestation subtype 0x04 uses the TPM2_Quote command to attest to PCR values.

TPM2_Quote signs a TPM2B_ATTEST structure, whose buffer is filled with a TPMS_ATTEST structure (consult the TPM 2.0 specification Section 10.12.9 for a normative definition):

Definition of TPM2B_ATTEST Structure <OUT>

Parameter	Type	Description
Size	UINT16	size of the attestationData structure
attestationData[size]{:sizeof(TPMS_ATTEST)}	BYTE	the signed structure

The last parameter, attestationData is a TPMS_ATTEST Structure as defined in the TPM 2.0 Library, Part 2:

Definition of TPMS_ATTEST Structure <OUT>

Type	Name	Description
Magic	TPM_GENERATED	the indication that this structure was created by a TPM (always TPM_GENERATED_VALUE)
Type	TPMI_ST_ATTEST	type of the attestation structure. In this case TPM_ST_ATTEST_QUOTE
qualifiedSigner	TPM2B_NAME	Qualified Name of the signing key
extraData	TPM2B_DATA	external information supplied by caller NOTE A TPM2B_DATA structure provides room for a digest and a method indicator to indicate the components of the digest. The definition of this method indicator is outside the scope of this specification.
clockInfo	TPMS_CLOCK_INFO	Clock, resetCount, restartCount, and Safe
firmwareVersion	UINT64	TPM-vendor-specific value identifying the version number of the firmware
[type]attested	TPMU_ATTEST	the type-specific attestation information

The TPMS_QUOTE_INFO structure is defined in the TPM 2.0 Library Part 2:

Definition of TPMS_QUOTE_INFO Structure <OUT>

Parameter	Type	Description
pcrSelect	TPML_PCR_SELECTION	information on algID, PCR selected and digest
pcrDigest	TPM2B_DIGEST	digest of the selected PCR using the hash of the signing key

TPML_PCR_SELECTION structure (consult the TPM 2.0 Specification, Part 2)

Definition of TPML_PCR_SELECTION Structure

Parameter	**Type**	**Description**
Count	UINT32	number of selection structures A value of zero is allowed.
pcrSelections[count]{:HASH_COUNT}	TPMS_PCR_SELECTION	list of selections
#TPM_RC_SIZE		response code when count is greater than the possible number of banks

pcrSelection is a TPMS_PCR_SELECTION Structure as defined in the TPM 2.0 Library, Part 2:

Definition of TPMS_PCR_SELECTION Structure

Parameter	**Type**	**Description**
Hash	TPMI_ALG_HASH	the hash algorithm associated with the selection
sizeofSelect {PCR_SELECT_MIN:}	UINT8	the size in octets of the pcrSelect array
pcrSelect [sizeofSelect] {:PCR_SELECT_MAX}	BYTE	the bit map of selected PCR
#TPM_RC_VALUE		

4.11.7.2 **TLV Format**

The TLV format is:

0x09 || length || 0x04 || TPM2B_ATTEST || TPMT_SIGNATURE

4.12 Implicit Attestation

Implicit attestation is a technique used when a system is expected to be in a static condition for a long period of time. Given this expectation, a simplified technique can be used to prove the system is in that state.

4.12.1 Implicit Attestation using constrained Signing Key

4.12.1.1 Description

Implicit attestation is created when a signature is signed with a signing key (described in section 4.4), but here the additional requirement is that the signing key can provably only be used to sign when the Attester in a particular state. Thus, the very act of signing using that key proves the Attester is in that known state. If the data signed includes a nonce, then this proves the Attester was in that state after the nonce was provided.

This information element contains a signature using the key. It starts with the element: 0x0A, followed by the length (4 bytes) of the signature, followed by the signature itself. This signature is of type TPMT_SIGNATURE (cf. TPM 2.0 specification[1], Part 2, Section 11.3.4):

Type	Name	Description
TPMT_SIGNATURE	Signature	the signature

4.12.1.2 TLV Format

The TLV is:

0x0A || length || signature

4.12.2 Implicit Attestation using DICE with Asymmetric Keys

4.12.2.1 Description

The element identifier is 0x0D.

A DICE key is implicitly defined such that it can only be generated when the system first mutable firmware is in a known state. A certificate for that key contains information about the state to which it is locked. This certificate structure can be found in the TCG Specification Implicit Identity based Device Attestation *[3]*.

4.12.2.2 TLV Format

0x0D || length of DICE signature [3] || DICE signature [3])

4.13 Previous Hibernation Reports

If a system has previously gone through a hibernation event, then the PCR values may not reflect the log values that are stored by the OS, as the computer memory has been restored to its previous state. Since the TPM was powered off, the PCRs are reset to their respective initial states[1] and no longer reflect the event records in the PCR log. In this event, previous hibernation records are needed (along with an audit session which records the number of events that were extended into the TPM since the last full reboot of the Attester). This information element is used to return a report of a previous hibernation record in the same boot cycle.

4.13.1 Description

Hibernation reports start with 0x0B. They contain PCR values prior to a hibernation cycle, TPM Reset counter and log values prior to the hibernation, TPM Reset counter and log values prior to the hibernation and attests those values were valid when the value of the TPM reset counter is constant during the same TPM reset counter value. No proof

[1] Depending on the PCR number, the initial state for a PCR is all zeroes or all ones.

that the attestation was “recent” is necessary. However, the pcrUpdateCounter does need to be part of the report. Every hibernation report needs to contain both the previous log and an attestation over an audit log that contains (using TPM2_PCRRead) the pcrUpdateCounter.

4.13.2 TLV Format

The TLV is:

0x0B || length || previous hibernation report

4.14 Supplementary Log Report

4.14.1 Description

When a Verifier wants an update from an Attester, it does not need to have log values repeated if they have already been reported. A Supplementary Log Report will only include new log values that have occurred since the last report from the Attester to the Verifier.

A supplementary log reports start with 0x0C. It includes only the latest Log Reports from the Attester for requested PCRs, after the number of the log entry specified by the Verifier. This way older log values that have already been analyzed by the Verifier do not need to be repeated.

4.14.2 TLV Format

The TLV is:

0x0C || length of supplementary log report || supplementary log report

5 References

[1] Trusted Computing Group, TPM 2.0 Library Specification:
https://trustedcomputinggroup.org/resource/tpm-library-specification/

[2] Asokan Attack:
https://trustedcomputinggroup.org/tnc-if-t-protocol-bindings-tunneled-eap-methods-specification/, https://www.rfc-editor.org/rfc/rfc6813.txt

[3] Trusted Computing Group, DICE Implicit Identity based Device Attestation:
https://trustedcomputinggroup.org/resource/implicit-identity-based-device-attestation/

[4] Time-based uni-directional attestation (TUDA) information elements:
https://datatracker.ietf.org/doc/draft-birkholz-i2nsf-tuda/,

[5] RFC 3161 IETF:
https://www.ietf.org/rfc/rfc3161.txt

[6] RFC-5816 IETF:
https://www.ietf.org/rfc/rfc5816.txt

[7] Hardware Requirements for a Device Identifier Composition Engine:
https://trustedcomputinggroup.org/resource/hardware-requirements-for-a-device-identifier-composition-engine/

[8] Trusted Computing Group, PC Client Specific Platform Firmware Profile Specification:
http://trustedcomputinggroup.org/resource/pc-client-specific-platform-firmware-profile-specification/

[9] Foundational Trust for IoT and Resource Constrained Devices:
https://trustedcomputinggroup.org/resource/foundational-trust-iot-resource-constrained-devices/

6 Example Designs using the Information elements

The following tables are a matrix of many use cases in detail and which information elements are sent by an Attester and Verifier. Note that there may be multiple exchanges which are not represented in this matrix.

Note: The absence of Verifier information elements from a use case is because that use case employs a one-way protocol.

1. Attester requests service from a Verifier. Verifier produces a nonce, and a list of information elements it needs the Attester to attest (no log information or hibernation data is requested). Attester provides the listed information elements.
2. Attester requests service from a Verifier and simultaneously supplies the Verifier with the request and his attestation, which uses either a 3rd party nonce or clock/tick based freshness proof and a list (no log information is requested) of information elements needed by the Verifier.
3. Attester requests service from a Verifier. Verifier produces a nonce, and a list of information elements it needs the Attester to attest (log information is requested but no hibernation information is requested.) Attester provides the listed information elements.
4. Attester requests service from a Verifier. Verifier produces a nonce, and a list (including log and hibernation information) of information elements it needs the Attester to attest. Attester provides the listed information elements and previous hibernation information if available.
5. Verifier asks Attester for an update on its state.
6. Verifier asks Attester for an update on its state, including any hibernation events.
7. Attester requests service from a Verifier and simultaneously supplies the Verifier with the request and his attestation, which uses either a 3rd party nonce or clock/tick based freshness proof and a list (including log information) of information elements needed by the Verifier.
8. Attester requests service from a Verifier and simultaneously supplies the Verifier with the request and his attestation, which uses either a 3rd party nonce or clock/tick based freshness proof and a list (including log information) of information elements needed by the Verifier. Verifier asks Attester for an update on its state. Previous hibernation information is included.
9. Attester creates a signature with information that proves the state of the system PCRs at the time the signature took place (no log file or hibernation file), to later be sent to a Verifier. (This is implicit attestation.)
10. Attester creates a signature with information that proves when the signature took place, to later be sent to a Verifier (no log file or hibernation file)). (This is implicit attestation.)
11. Attester creates a signature with information that proves the state of the system PCRs at the time the signature took place (with log file, but no hibernation file), to later be sent to a Verifier. (This is implicit attestation.)
12. Attester creates a signature with information that proves the state of the system PCRs at the time the signature took place (with log file and hibernation file), to later be sent to a Verifier. This includes hibernation data if it is relevant. (This is implicit attestation.)
13. Attester periodically reports on its state (no log files or hibernation files).
14. Attester periodically reports on its state (log files, no hibernation files).
15. Attester periodically reports on its state (log files and hibernation files).

Uses cases by number follow below in Table 8: Use Cases 1-15

Information Element \\ Use case number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Attester															
AK certificate Section 4.3	🟩	🟩	🟩	🟩			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
PCR list and values Sections 4.5, 4.6	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Freshness type and qualification data Sections 4.8 4.9	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩		🟩	🟩	🟩	🟩
Attestation subtype and Attestation data Section 4.11.3	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Signature using Signing Key Section 4.12.1							🟩	🟩	🟩	🟩	🟩	🟩			
Previous Hibernation report Section 4.13				🟩		🟩		🟩				🟩			🟩
Log Report Sections 4.7 4.14			🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩		🟩	🟩
Verifier															
Requested information types (list) Section 4.1	🟩		🟩	🟩	🟩	🟩									
PCR list Sections 4.5, 4.6	🟩		🟩	🟩	🟩	🟩									
Nonce Sections 4.8.3	🟩		🟩	🟩	🟩	🟩									

Table 8: Use Cases 1-15