TRUSTED® COMPUTING GROUP

REFERENCE

TCG Trusted Attestation Protocol (TAP) Use Cases for TPM Families 1.2 and 2.0 and DICE

Version 1.0
Revision 0.33
August 13, 2019

Contact: admin@trustedcomputinggroup.org

PUBLIC REVIEW

Work in Progress

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

DISCLAIMERS, NOTICES, AND LICENSE TERMS

Acknowledgement

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 TCG members contributing to this document:

Name	Affiliation
Monty Wiseman	**General Electric**
David Challener	**The Johns Hopkins University Applied Physics Laboratory**
Greg Kazmierczak	**Bright Plaza, Inc.**
Carolin Baumgartner	**Carolin Baumgartner**
Charles Schmidt	**The MITRE Corporation**
Ann T. Krieger	**United States Government**
Henk Birkholz	**Fraunhofer Institute for Secure Information Technology (SIT)**
Ira McDonald	**High North, Inc.**
Ken Goldman	**IBM**
Lisa Lorenzin	**Lisa Lorenzin**
Guy Fedorkow	**Juniper Networks, Inc.**
Andreas Fuchs	**Fraunhofer Institute for Secure Information Technology (SIT)**
Stan Potter	**United States Government**
Jessica Fitzgerald-McKay	**United States Government**
Tom Laffey	**Hewlett Packard Enterprise**
Sue Leicht	**United States Government**
Andreas Steffen	**HSR University of Applied Sciences Rapperswil**
Michael Eckel	**Huawei Technologies Co., Ltd.**
Ned Smith	**Intel**
Mark D. Baushke	**Juniper Networks, Inc.**

CONTENTS

1 Introduction

1.1 Scope

Endpoint integrity and corresponding attestation evidence is critical to many use cases. DICE [2], TPM [1] and platform specification[5] were designed to provide information -- evidence -- helpful for Verifiers to determine the state of a platform -- the Attester. The Trusted Attestation Protocol (TAP) [4] defines a protocol that enables the collection of information from a TPM or DICE embedded in a platform. The primary intent of TAP is to enable operators to use this information to determine endpoint integrity. However, TAP itself does not specify how these integrity checks are performed; it is concerned only with ensuring that sufficient information can be collected from endpoints and delivered in a manner that preserves its suitability for such checks.

TAP is implementation neutral. It may be used by 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 Platform Configuration Register (PCR) values. It does not define how the information is provided to the Verifier.

1.2 Audience

These TAP Information Use Cases are useful for implementers of a TAP binding specification. These use cases are also useful to enterprise architects in designing their endpoint integrity capabilities. Readers should have a familiarity with TPMs and/or DICE.

1.3 Relation to PTS 1.0

TAP is a complete redesign of Platform Trust Services (PTS) 1.0. It is significantly different from PTS 1.0 and is not backward compatible with PTS1.0.

TAP adds support for DICE and TPM 2.0, including several new capabilities not available in older TPM versions. Because TPM 1.2 remains widely deployed and many environments will contain a mix of TPM 1.2 and TPM 2.0 machines for the near future, TAP also supports interactions with TPM 1.2s. TPM 1.2 machines are not able to support all TAP functions due to their more limited capabilities.

2 Purpose of this document: TAP Use Cases

This document describes and specifies Trusted Attestation Protocol (TAP) Use Cases. It is to be used in conjunction the TAP Information Model Document [4].

This document does not describe in detail how the information described in the TAP Information Model is to be transmitted or interpreted. Instead it defines what information can be requested, and what information is provided by an Attester that desires access to a network.

3 Use cases

In order to understand what information is to be requested and provided from the TAP Information Model, it is useful to describe some use cases. In addition to describing the use cases themselves, these descriptions identify pre- and post-conditions for the case as well as any constraints on the process itself that TAP needs to enforce across all bindings.

3.1 Attester access to a network or compute cluster

In this use case an Attesting platform (called an Attester) seeks access to a network or compute cluster and must attest to its state as a condition of entry.

3.1.1 Preconditions

The Verifier already has the public portion of a Restricted Signing Key (also known as an Attestation Key or AK) for every Attester that could legitimately join the network or compute cluster, or the public key of a CA (or CAs) which has/have been used to create a certificate for those AKs. In the case of TPM 1.2, an Attestation Identity Key (AIK) is used instead of an AK.

Additionally, if a VPN is being established, all Attesters have been pre-provisioned with a key that allows a standard mechanism for sharing an ephemeral key between them (once attestation has been achieved). That VPN key must be bound in some manner to the AK/AIK, so that Bell-LaPadula attacks [3] can be avoided. (E.g. both keys can be associated with same system on a Lightweight Directory Access Protocol (LDAP) database).

In addition, the Attester may be required to store a quote for all prior boot sessions since its last TPM reset. This information is necessary to ensure that PCRs can be verified across hibernation sessions for those systems that use post-boot extension of PCRs (such as IMA in Linux).

> Note: If a system is known to not hibernate, or post-boot PCRs are not used for determination of system state, this will not be necessary.

3.1.2 Process

Three different processes may be used to provide attestation from an Attester to a Verifier. They are distinguished by how freshness of the attestation is proven, though the information sent between the Attester and Verifier is essentially the same.

3.1.2.1 Process one: The verifier provides the nonce

- The process begins with the Attester contacting the network or compute cluster gatekeeper with a request to join the network or compute cluster, including information identifying the Attester.
- The Verifier responds with a request for attestation, including the information types it wishes, and a random nonce (which will be used to prove freshness.)
- If a VPN is to be established, the Attester and Verifier use associated information to establish an ephemeral key to be used for the VPN.
- The Attester responds with the information corresponding to the information types requested. These include PCR values, Attestation of those values, Logs of measurements included in PCRs, etc., which may be found in the TAP (Trusted Attestation Protocol) Information Model [4].
- The Verifier verifies the state of the Attester using the information received.

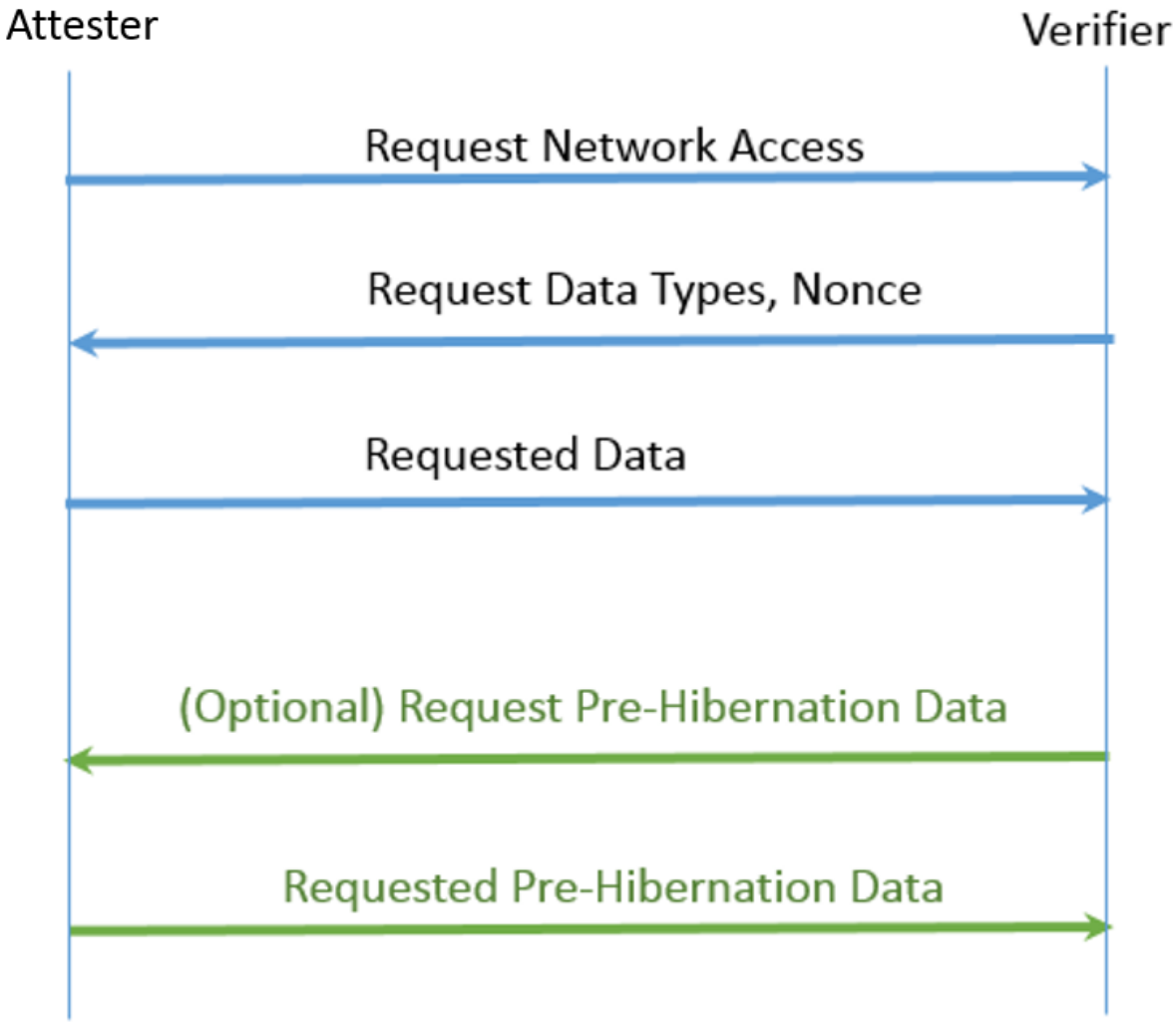

Figure 1 Sequence Diagram for Process One

3.1.2.2 Process two: A third party provides the nonce

In this case the Attester is configured at provisioning time to know the information types that the Verifier will need in order to verify an attestation.

- The process begins with the Attester contacting a third party trusted by the Verifier (such as a time stamping service) to obtain a provably recent random number to be used as a nonce. This third party is (obviously) not behind the VPN.
- The Attester contacts the Verifier with a request for access, including in its request the information corresponding to the information types known to be needed by the Verifier. These include PCR values, Attestation of those values, Logs of measurements included in PCRs, etc., which may be found in the TAP (Trusted Attestation Protocol) Information Model [4].
- This request includes the nonce, and associated information that proves it is fresh.
- The Verifier verifies the state of the Attester using the information received.
- If a VPN is to be established, the Attester and Verifier use associated information to establish an ephemeral key to be used for the VPN.

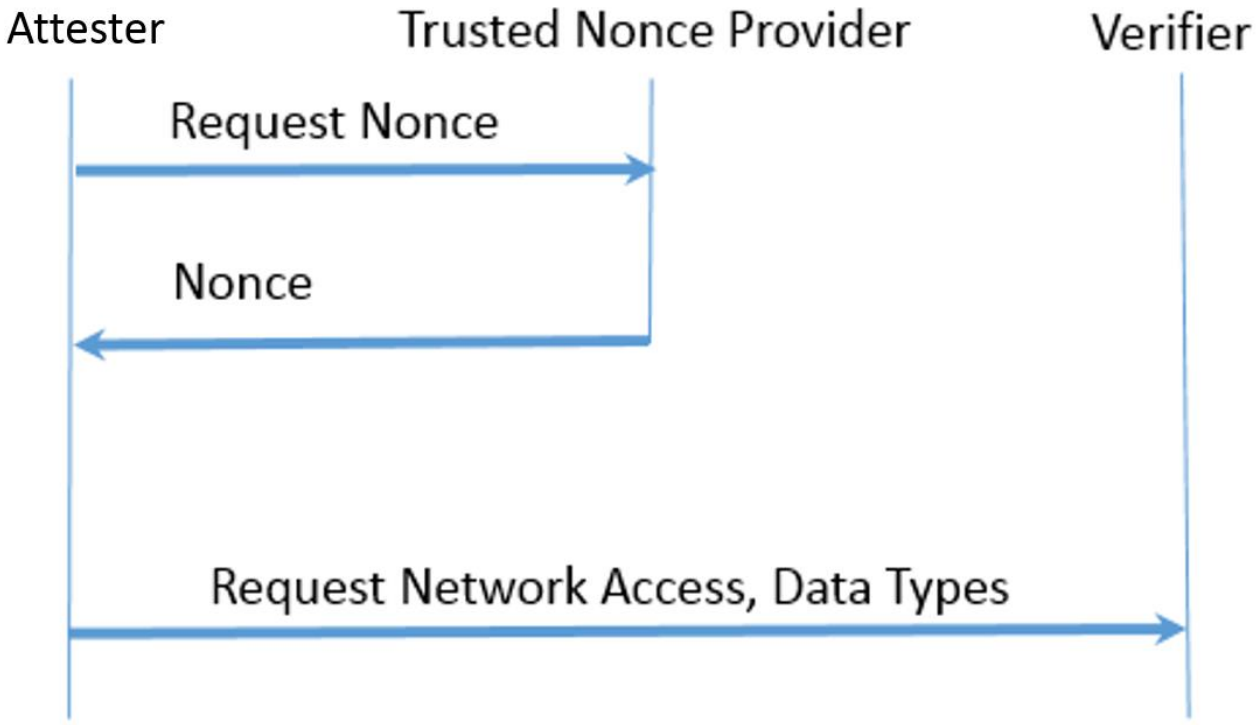

Figure 2 Sequence Diagram for Process Two

3.1.2.3 Process Three: Clock-based attestation (TPM 2.0 only)

In this case, the Attester is configured to know the information types that the Verifier will need in order to verify an attestation.

- The process begins with the Attester contacting a third-party time stamping service and establishing the correctness of the TPM clock. This third-party service is (obviously) not behind the VPN. This TPM clock certificate includes the values of the TPM Reset and Restart counters when the certificate was created.
- The Attester contacts the Verifier with a request for access, including in its request the information corresponding to the information types known to be needed by the Verifier. These include PCR values, Attestation of those values, Logs of measurements included in PCRs, etc., which may be found in the TAP (Trusted Attestation Protocol) Information Model [4].
- This request includes the proof of the correctness of the Attester's TPM clock.
- The Verifier verifies the state of the Attester using the information received.

If a VPN is to be established, the Attester and Verifier use associated information to establish an ephemeral key to be used for the VPN. Attestation will include a signature. TPM 2.0 signatures used in the attestation sign the then current TPM clock value as well as the values of the TPM reset and restart counters. If those values match the values of reset and restart counters in the signed Time Stamp, the value of the TPM clock can be used to determine the time the attestation took place, subject to the accuracy of the original establishment of the TPM clock certificate and subject to the TPM clock drift.

3.1.3 Post conditions

The Verifier has received a report containing the information types has been sent to the Verifier. This comprises sufficient information to allow the Verifier to compare the provided PCRs against the Measurement Log and determine whether the PCRs describe valid system measurements.

At this point, the Verifier may request previous Log reports, if necessary due to a hibernation cycle. Alternatively, at this point, the Attester and the Verifier may exchange an ephemeral key (e.g. for a VPN) using a standard protocol such as TLS, using a key known by the Attester to be associated with the Verifier. (Attestation Keys (AKs) typically cannot be used for TLS key exchange due to the restriction on AKs to only sign things which the TPM has hashed.)

3.2 Attester remeasurement

In this use case, the Attesting platform is already connected to the enterprise network. The Verifier wishes to collect the latest PCR measurements. This is a subset of the process described in Section 3.1.2.1 “Process one: Verifier produced nonce”.

3.2.1 Preconditions

The Verifier has already been through one attestation process with the Attester. The Verifier has retained a record of the previously sent report, and only needs an update.

3.2.2 Process

The Verifier contacts the Attester with a list of PCRs of interest, and an indication of the previously received log value for those PCRs, and a nonce. The Attester responds with the requested PCRs, associated entries from its management log, and its pcrupdatecounter, all signed using the Attester’s key.

By examining the pcrupdatecounter, the Verifier can determine whether there have been any hibernation sessions since the last TPM reset. If that is the case, the Verifier will need to collect quotes from hibernation sessions in order to verify the included PCRs. The Verifier can contact the Attester again, instructing it to send all quotes from these hibernation sessions. The Attester then returns these previously-generated quotes.

As with the previous use cases, while the described process uses a nonce provided by the verifier to prove freshness, freshness can be proven through different techniques, using a Verifier provided nonce, a Trusted Third Party provided nonce, or a certified TPM clock time.

If the Attester has gone through a full power cycle of the system since the last attestation, a full attestation (as described in 3.1.2.1) will have to be returned. (When a full power cycle has happened, all PCRs are reset to their starting value of all zeros.)

3.2.3 Post conditions

The Verifier has received all the information types required.

This comprises sufficient information to allow the Verifier to compare the provided PCRs against the measurement log and determine whether the PCRs describe valid system measurements.

3.3 State of an endpoint when a signature took place

In this use case, the Attester needs to attest that its PCRs were in a given state at a given point in time. This is done despite the endpoint not being connected to the network at that time. For example, the Attester might wish to digitally sign a contract, and would like to assure the contract’s recipient that its PCRs were in a certain state at the time the contract was signed.

3.3.1 Preconditions

The Verifier already has the public portion of an Attestation Key (Restricted Signing Key as defined in the TPM 2.0 Library specification) of the Attesting platform on which a contract will be signed, or a public key corresponding to a CA that signed the AK certificate.

In addition, the Attester is required to store a quote for all prior boot sessions since the last time the Attester’s TPM was reset. This information is necessary to ensure that PCRs can be verified across hibernation sessions.

3.3.2 Process

When the user is ready to sign the contract, the Attester's TPM is used to generate a signing key, SK, whose secret portion can never leave the TPM and whose use is only allowed by a policy bound to the current set of PCR values.

The AK is used to certify the SK, using TPM2_Certify. An audit session is started, and the TPM is used to sign the contract with the signing key and then read the PCR values. The audit session is then signed with the AK.

The Attester then collects the following information:

- The signature over the contract
- The results of Reading the PCRs (twice)
- The signature over the audit session
- All log entries associated with those PCR
- The result of the TPM2_Certify of the signing key
- The elements used to create the signing key's policy
- Quotes associated with all hibernation cycles in the current boot cycle

Later, the Attester can deliver the signed contract and associated information to a Verifier

3.3.3 Post conditions

The recipient of the signed contract and associated required information types will be able to determine:

1) That the contract was signed using the signing key identified in the quote bundle.
2) That the signing key could only have been used when the TPM's PCRs were in a specific state.
3) The set of software measurements leading to the PCR values used when the contract was signed.

Based on examination of this information, the contract recipient will be able to derive the state of the Attester's software at the time the contract was signed.

3.4 Simultaneous Attestation to Multiple Verifiers

In this case, the Attester needs to attest its state to a large number of Verifiers. For example, in a data center, data processing units might wish to determine the state of the control plane manager. Given that there might be hundreds of data processing units all seeking attestation of the manager's state, having the manager attest to each of these units individually could be prohibitive. Instead, it would be useful to support a way of combining attestation requests so that multiple requests can be served by a single quote.

3.4.1 Preconditions

Each Verifier already has the public portion of the same AK for the Attester, or a public key corresponding to a CA that signed the AK certificate.

In addition, the Attester may be required to store a quote for all prior boot sessions since its last TPM reset. This information may be necessary to ensure that PCRs can be verified across hibernation sessions.

3.4.2 Process

3.4.2.1 Process 1

The process is identical to Section 3.1.2.1 with the following exception: Instead of immediately responding to a Verifier's request, the Attester might wait a brief time to see if other requests arrive. Each request will arrive with its own nonce value.

3.4.2.2 Process 2

Process 2 is to use a nonce from a Third Party as in Section 3.1.2.2, trusted by all the Verifiers.

3.4.2.3 Process 3

Process 3 uses a Third Party, trusted by all the Verifiers, to establish trust in the Attester's TPM's clock as in Section 3.1.2.3. The Attester's TPM's internal clock is then used to provide proof of the freshness of the Attester's attestation.

3.4.3 Post conditions

Each Verifier will independently be able to validate the state of the Attester's PCRs and associated software and will also be able to determine that the received response was fresh.

3.5 Attestation in Unidirectional Point-to-Point Scenarios

In this use case, the Attester resides in a high-security network zone (such as a voting machine), whilst the Verifier resides in a low-security zone. Network security functions restrict communication between these zones to one-way from the high-security zone to the low-security zone (e.g. via a data diode).

Note: The Attester decides on the attestation type. In contrast to Section 3.6, no attestations requests are sent from the Verifier to the Attester.

3.5.1 Preconditions

The Verifier already has the public portion of an AK for the Attester signing the contract, or a public key corresponding to a CA that signed the AK certificate.

In addition, the Attester may be required to store a quote for all prior boot sessions since its last TPM reset. This information may be necessary to ensure that PCRs can be verified across hibernation sessions.

3.5.2 Process

An approach is to perform an attestation as in Section 3.1.2.3.

3.5.3 Post conditions

A report containing the information types has been sent to the Verifier. It will therefore have sufficient information to determine the state of the attesting party.

3.6 Attestation via protocols that do not support sessions

In this use case, the Attester and the Verifier communicate via a technique that does not support sessions. This means that challenge/response-based approaches are not supported natively. Instead, for each protocol transaction a single information element is either transferred from A to B, or from B to A, but no association between each transaction is maintained (no application association state is created by control plane functions that enable the intended protocol behavior).

Note: It is possible for the Verifier to request a specific attestation type from the Attester (e.g. mime-type or IOD) without being able to parameterize the request with a nonce.

3.6.1 Preconditions

Each Verifier already has the public portion of the same Restricted Signing Key for the attesting Attester, or a public key corresponding to a CA that signed the AK certificate.

In addition, the Attester may be required to store a quote for all prior boot sessions since its last TPM reset. This information may be necessary to ensure that PCRs can be verified across hibernation sessions.

3.6.2 Process

3.6.2.1.1 Process 1

Process 1 is to use a nonce from a Third Party as in Section 3.1.2.2, trusted by all the Verifiers.

3.6.2.1.2 Process 2

Process 2 uses a Third Party, trusted by all the Verifiers, to establish trust in the Attester's internal clock as in Section 3.1.2.3. The Attester's TPM's internal clock is then used to provide proof of the freshness of the Attester's attestation.

3.6.3 Post conditions

Each Verifier will independently be able to validate the state of the Attester's PCRs and associated software and will also be able to determine that the received response was generated recently.

3.7 Attestation of Composite Devices

3.7.1 Preconditions

The Verifier already has the public portions of the Restricted Signing Keys for the attestation of all components, including sub-components, of a composite device. Furthermore, the Verifier needs to know the identity of all components of a composite device and thus detect unknown or missing components. These devices should have the ability to attest (which implies they have a TPM or DICE).

3.7.2 Process

The Verifier contacts the composite device and requests an attestation, including all sub-components. Typically, a composite device is reachable from the outside via a single IP address, usually that of the master or main control plane. A nonce is transmitted from the Verifier to the composite device to prevent replay attacks. The main control plane in turn triggers subcomponent TPM quotes for all sub-components, wraps them in a data structure, hashes this data structure plus the nonce received from the Verifier (concatenation), and passes the hash as the nonce in a TPM2_Quote command to the control plane's TPM. The TPM quote of the main control plane and the data structure containing all sub-component attestations are sent to the Verifier.

3.7.3 Post conditions

The Verifier has sufficient information to determine the state of the attesting party and to know the information it received was recent.

3.8 Bell-LaPadula Considerations for Runtime Attestation

A platform handling high assurance data may require access to lower assurance components such as network routers and switches. These high assurance platforms typically comply with the Bell-LaPadula[3] model by encrypting data that is communicated via lower classification components. However, networks still may need to verify a platform's posture (even identity) before passing or processing encrypted traffic.

Providing attestation of runtime applications (i.e., event logs of applications) may provide an attacker with a side-channel attack on the high assurance platform's assets. Therefore, one cannot allow general access to the runtime Event Logs. Thus, the Event Log requires access control.

3.8.1 Preconditions

The Attester's network stack is configured to provide TAP services only to authorized Verifiers and optionally with confidentiality protection. The Attester's network stack does this using a TAP binding protocol that provides a capability such as TLS.

3.8.2 Process

The Attester requests network services. Network equipment attempts to open a TAP session but cannot connect to the Attester's TAP services because the network equipment doesn't have credentials to open the connection. Network equipment forwards the Attester's request to a Verifier with the necessary Attester's credentials. Verifier opens the TAP binding protocol using its credentials. Depending on the session's policy, the contents may be confidentiality and integrity protected. Upon approving the Attester's posture, the Verifier notifies the Network equipment.

NOTE: The referring and approval protocols between the Network equipment and the Verifier are out of scope for TAP.

3.8.3 Post conditions

The network equipment must allow the Attester's traffic.

3.9 IoT Device with DICE reports the status of its firmware when requested

For this use case, it is assumed that the reader is familiar with the reference document Implicit Based Device Attestation [2].

3.9.1 Preconditions

An IoT DICE device, and a certificate for the attestation key of that DICE subsystem. This certificate specifies the state of the DICE subsystem (especially its firmware).

3.9.2 Process

The process is detailed in the reference document: Implicit Based Device Attestation [2]. When a verifier requests the device to give evidence of the state of the (potentially mutable) firmware of the device, the verifier provides a nonce to the IoT device, and the IoT device uses its unique key to sign that nonce and returns it along with the signing key's certificate. The service then verifies the certificate, verifies that the certificate purports that the key implicitly attests that the IoT device's firmware is in the correct state.

3.9.3 Post conditions

The verifier is able to use the information sent as evidence of what version of firmware is being run on the IoT device, and will grant access to services as appropriate.

4 References

[1] Trusted Computing Group, TPM 2.0 Library Specification:

https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0.zip

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

[3] Bell-Lapadula:

http://www.cs.unc.edu/~dewan/242/f96/notes/prot/node13.html

[4] Trusted Computing Group, TAP (Trusted Attestation Protocol) Information Model:

https://trustedcomputinggroup.org/resource/tcg-trusted-attestation-protocol-tap-information-model-for-tpm-families-1-2-and-2-0-and-dice-family-1-0/

[5] Trusted Computing Group, PC Client Specific Platform Firmware Profile Specification:

http://trustedcomputinggroup.org/resource/pc-client-specific-platform-firmware-profile-specification/