TRUSTED® COMPUTING GROUP

GUIDANCE

TPM Active Interposer Mitigation Guidance for Data Centers

Version 1.0
RC 1
May 28, 2026

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

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, DOCUMENT OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this document and to the implementation of this document, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this document or any information herein.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on document licensing through membership agreements.

CHANGE HISTORY

REVISION	DATE	DESCRIPTION
1.00	May 21, 2026	• Initial Release of Version 1.00.

CONTENTS

DISCLAIMERS, NOTICES, AND LICENSE TERMS 1
CHANGE HISTORY 2
1 INTRODUCTION 4
1.1 Prerequisites 4
1.1.1 SPDM 4
1.1.2 DICE 5
1.1.3 DPE 5
1.1.4 Remote Verifier 5
1.2 Architecture 5
2 THREAT MODEL 7
2.1 Scope Exclusions 7
2.2 ATTACKS AND MITIGATIONS 7
2.2.1 Feeding compromised boot firmware to the SoC 8
2.2.2 Suppression of PCR Measurements 8
2.2.3 Injection of PCR Measurements 8
2.2.4 TPM Redirection 8
2.2.5 TPM Second-stage Redirection 9
2.2.6 Spurious TPM Reset 10
3 DESIGN GOALS 12
3.1 Non-Goals 12
4 BOOT FLOW (SPDM) 13
5 BOOT FLOW (non-SPDM) 16
6 PROTECTING HOST MEASUREMENTS 19
7 ATTESTATION 20
8 REFERENCES 21

1 INTRODUCTION

This document describes a technique for mitigating physical interposer attacks on the bus between a System on Chip (SoC) and a discrete Trusted Platform Module chip (TPM) in datacenter hardware environments.

A discrete TPM is a chip that is typically used to hold measurements of an SoC's boot software, such as that of a host SoC or Baseboard Management Controller (BMC), and attest to those measurements. The TPM may also be used to protect secrets for the SoC, and conditionally release them based on the SoC's boot measurements.

In datacenter platforms, hardware may be physically distributed across several modules. The bus between an SoC and its TPM is an attractive target for physical adversaries wishing to attest to false measurements or obtain TPM-bound objects. Boot measurements are typically sent as TPM commands over a serial peripheral interface (SPI) bus. An adversary with physical access may interpose on this bus to suppress true measurements and instead provide their own false measurements. The attacker may also wield TPM-bound objects by injecting their own commands instructing the TPM to wield said objects. Without cryptographically authenticating the caller, the TPM will potentially reveal secrets over the bus, exposing them to the adversary.

Two mitigation techniques are presented in this document. The first technique relies on two technologies: The Security Protocol and Data Model (SPDM) and the Device Identifier Composition Engine (DICE), while the second technique relies only on DICE. The techniques are conceptually similar but the non-SPDM solution is only meant to serve as a stop-gap solution for scenarios that do not support SPDM. While the proposed mitigations may be applicable to other scenarios outside of datacenters, aspects such as boot time were not considered when designing the mitigations and may therefore limit their potential scope.

Both mitigations extend the ones proposed in two similar TCG documents, titled “CPU to TPM Bus Protection Guidance for Active Attacks” (Trusted Computing Group, 2023) and “CPU TPM Bus Protection Guidance Passive Attack Mitigation” (Trusted Computing Group, 2025). The main differences lie in what technologies are used and in what attacks are covered by the mitigations. The previous documents address only attacks that try to expose secrets transmitted between a TPM and its caller by accessing the bus between the TPM and SoC. In this document, attacks that target the software executing on the SoC are considered as well, since an attacker that is capable of interposing a TPM and its SoC may additionally try to compromise the SoC’s firmware.

For example: some protection schemes rely on early boot software running in the SoC to wield a secret, which is used to authenticate to the TPM. The secret may be static for the lifetime of the CPU. The early boot software is assumed to be well-behaved. If early boot software is compromised, it may misuse the secret. In the threat model of this document, a TPM interposer may modify and compromise early boot software by virtue of controlling the SoC’s boot bus. Therefore, a static secret known to early boot software is insufficient to mitigate this threat.

The new mitigations also provide a mechanism for collecting evidence of the existence of the secure channel between the SoC and TPM that can later be presented to a remote verifier.

1.1 Prerequisites

1.1.1 SPDM

The Distributed Management Task Force (DMTF) defines the following specifications: Security Protocol and Data Model (SPDM) (DMTF) and the Secured Messages using SPDM Specification (DSP0277) (DMTF, 2022). SPDM is a protocol for performing message exchanges between devices. SPDM 1.2 and later provides mutually-authenticated session key exchange protocols to enable confidentiality with integrity protected data communication. In DSP0277, the DMTF defines a methodology for communicating application data securely by utilizing SPDM.

This document presents a flow by which TPM commands exchanged between the SoC and the TPM are protected via encapsulation within an SPDM secure session. This flow relies on the specification titled “TPM Communication over SPDM Secure Session” (Trusted Computing Group) published by the TCG.

This document also presents a flow that does not rely on SPDM, and instead leverages EK-salted sessions.

Throughout, the term “TPM identity key” refers to the SPDM session establishment public key (i.e. the public key used to verify SPDM session establishment transcripts), or the EK public key, depending on whether SPDM is used.

1.1.2 DICE

TCG defines the DICE Layering Architecture (Trusted Computing Group), a method of giving a device a unique cryptographic identity bound to its boot measurements. A device that supports DICE contains a Unique Device Secret (UDS) which is used to form the device's identity keys.

This document presents a flow that relies on SoCs being given a cryptographic identity based on DICE, where the SoC wields this identity to establish a remotely-attested secure session with the TPM.

1.1.3 DPE

DICE was designed for relatively low-powered devices, such as those used within the internet of things (IoT). Software layers that use DICE are given a cryptographic identity by way of a private key generated by the preceding boot layer. Software must keep that identity key secret; if the key is compromised, the device's identity can be abused.

DICE was not designed with modern SoC boot layers in mind. It is non-trivial to keep an identity key secret and propagate it across multiple low-level transitions over the course of boot-up.

TCG defines DICE Protection Environment (DPE) (Trusted Computing Group, 2023), a standard that allows one entity to derive and wield DICE identity keys on behalf of a client. In this manner, the identity keys are kept safe against leakage, and the burden of managing DICE keys is alleviated from the client.

This document introduces the term iRoT (integrated root of trust) to refer to any hardware root of trust located within the SoC’s package.

This document uses the term “early-boot firmware” to refer to all firmware that runs in an SoC prior to the SoC being able to talk to the external TPM.

The flows defined in this document expect that the SoC has an iRoT that implements DPE. The flows further expect the SoC to capture early-boot firmware measurements into the iRoT, such that the captured measurements accurately reflect the code that runs within the SoC. A remote verifier may retrieve these measurements directly from the iRoT. This document additionally assumes that the SoC stores some objects or measurements in the discrete TPM, which the SoC wishes to bind to the iRoT, along with the early-boot firmware measurements captured by the iRoT. This enables secure usage of advanced TPM features that are not assumed to be available in the iRoT.

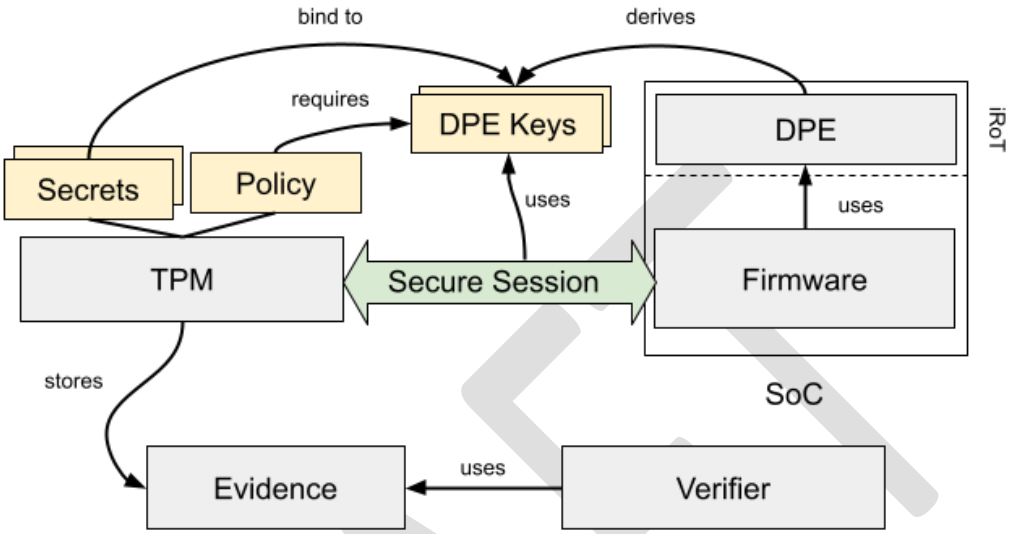
1.1.4 Remote Verifier

The solution that follows assumes the existence of a trustworthy remote verifier that can examine attested evidence emitted from the SoC iRoT and TPM, to determine (a) whether the SoC-TPM pairing is correct, and (b) whether the software that booted on the SoC is legitimate.

1.2 Architecture

In the proposed mitigation, the firmware running on the SoC uses its DICE identity to establish a secure session with the TPM. Using this secure session, the firmware can provision secrets to the TPM. Host firmware then uses policies to bind these secrets to its DICE identity. A remote verifier can consume evidence from the TPM and verify

that a particular secret was bound to a particular DPE key. This way, the verifier can verify assumptions about the platform's hardware and firmware since both aspects influence the DPE keys. At the same time, potential interposer attacks that aim to compromise the secrets or measurements performed within the secure session are prevented or detected using the secure session.



The proposed mitigation makes it possible to control the time that secrets are allowed to be used: In some use cases, it is desirable to prevent later boot stages (such as an operating system) from using secrets provisioned by early-boot firmware. Early-boot firmware may still choose to pass on control to later boot stages if that's required. The proposed schemes are examples and it is possible to create variations of them that involve several, independent secrets that are available at different boot stages.

2 THREAT MODEL

We model our adversary as being able to perform a number of attacks:

- Passively view traffic that flows over the bus between the SoC and its TPM.
- Suppress, modify, or redirect individual TPM requests or responses that are exchanged between the SoC and its TPM.
- Inject their own commands to the TPM. This may either be performed in situ or via theft of the TPM.
- Assume control of the SoC by providing their own boot code (e.g. platform firmware, kernel).

The latter capability is equivalent to attackers leveraging a known vulnerability in old platform firmware via rollback, or modifying the platform firmware itself. This is mitigated by capturing early-boot firmware measurements into the iRoT.

It is worth highlighting that even with first-instruction integrity features such as Intel Boot Guard or AMD Secure Boot enabled, there can be implementation bugs in authentic firmware signed by the OEM or platform owner. On some systems, components such as glue-logic Complex Programmable Logic Devices (CPLDs) or Field Programmable Gate Arrays (FPGAs) exist that already interpose the connection between the SoC and the discrete TPM. When an adversary gains remote access to these components, it might be possible to perform an active interposer attack without physical access. Attackers who have physical access to a device may therefore be able to exploit such vulnerable components and control the SoC during boot. Since supply chains can be complex, it is reasonable to assume that motivated adversaries will have physical access to a device at some point. Finally, enabling first-instruction integrity features such as Intel Boot Guard or AMD Secure Boot requires blowing public keys into fuses, limiting the ability to repurpose hardware in the future.

Note that platform firmware may be protected by an external platform root of trust, which would impede a remote attacker's ability to provide compromised platform firmware to the SoC. However, we model a physically-present adversary as being able to bypass platform root of trust protections.

2.1 Scope Exclusions

The following is a non-exhaustive list of attacks that are out of scope for the present document:

- Leverage zero-day bugs in existing/intended code to gain code execution (i.e. buffer overflows).
- Interposing between the individual SoCs of a multi-socket machine.
- Monitoring or interfering in intra-SoC or intra-TPM traffic.
- DMA or cold-boot attacks to retrieve secrets held in SoC memory.
- Attacks that steal primary seeds held by the TPM, or the DICE UDS held by the SoC iRoT.
- Denial-of-service attacks.
- Attacks that involve opening the SoC's or TPM's chip package.
- Compromising the integrity of locality indicators.
- Compromising platform authorization.

2.2 ATTACKS AND MITIGATIONS

This section enumerates a number of attacks on the proposed schemes and mitigations to each.

2.2.1 Feeding compromised boot firmware to the SoC

Assume the existence of a critical bug in low-level SoC boot firmware. This bug could be of the same severity as one that would affect the integrity of traditional S-RTM measurements sent during TPM initialization. An attacker may wish to take advantage of such a bug by forcing the SoC to boot through this compromised firmware.

Mitigation: The SoC's iRoT ensures that the measurements of this low-level boot firmware are captured and factored into the DICE alias key used to establish the session between the SoC and TPM. The measurements are used both in the derivation of the alias private key, and are added to the alias key's certificate. Thus, when a verifier evaluates the DICE certificate chain it can verify that the expected boot firmware is running on the SoC. The verifier can also evaluate a TPM object's policy digest to verify that any object bound to that DICE key via TPM2_PolicyTransportSPDM will be unavailable to an attacker that forces the SoC to run different boot firmware in the future.

2.2.2 Suppression of PCR Measurements

An attacker may wish to run, for example, a compromised kernel on a SoC, and mask that fact by dropping the PCR extensions containing the kernel's measurements, substituting their own false extensions.

Mitigation: The SoC relies on a secure session between the TPM and its firmware for detecting any dropped TPM commands. When tampering is detected, firmware will instruct the iRoT to invalidate its current DPE context. Even when the TPM's PCRs indicate that a legitimate kernel is running, the compromised kernel will be unable to modify the iRoT measurements or wield any TPM objects bound to the SoC's DICE alias key. For more information on how measurements can be protected, please refer to the Protecting Host Measurements chapter.

2.2.3 Injection of PCR Measurements

An attacker may wish to inject their own measurements into the TPM's PCRs, to falsely indicate that some intended software is running.

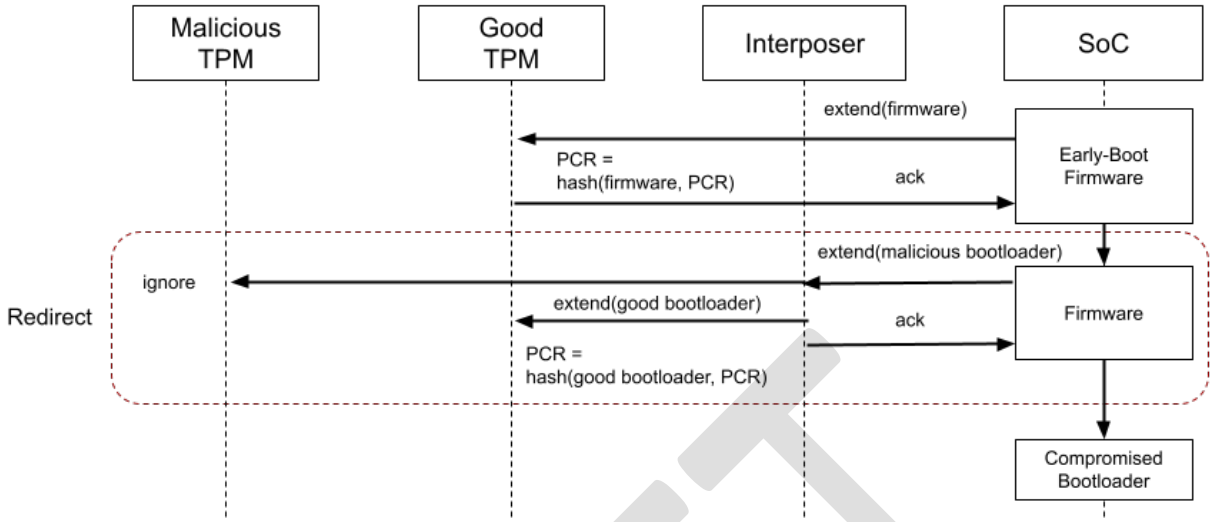
Mitigation: The attacker's ability to inject false PCR extensions will merely serve as a denial-of-service attack; the attacker-supplied measurements will permute the PCRs past any intended or expected value.

2.2.4 TPM Redirection

Suppose a SoC boots through early-boot firmware, a bootloader, and a kernel; and suppose each of these software layers establishes their own secure session with the TPM.

An attacker with access to the TPM bus may wish to run a compromised bootloader on a SoC, and mask that fact by preventing the actual bootloader measurements from being extended into the legitimate TPM.

The attacker may do this by directing the platform firmware's actual (compromised) bootloader measurements to a separate (possibly-emulated) TPM, while in the meantime the attacker feeds falsified (expected) bootloader measurements to the legitimate TPM. The attacker can then swap the legitimate TPM back when the bootloader runs. The SoC never detected a dropped command, and so the compromised bootloader has access to the keys it needs to communicate securely with the legitimate TPM.

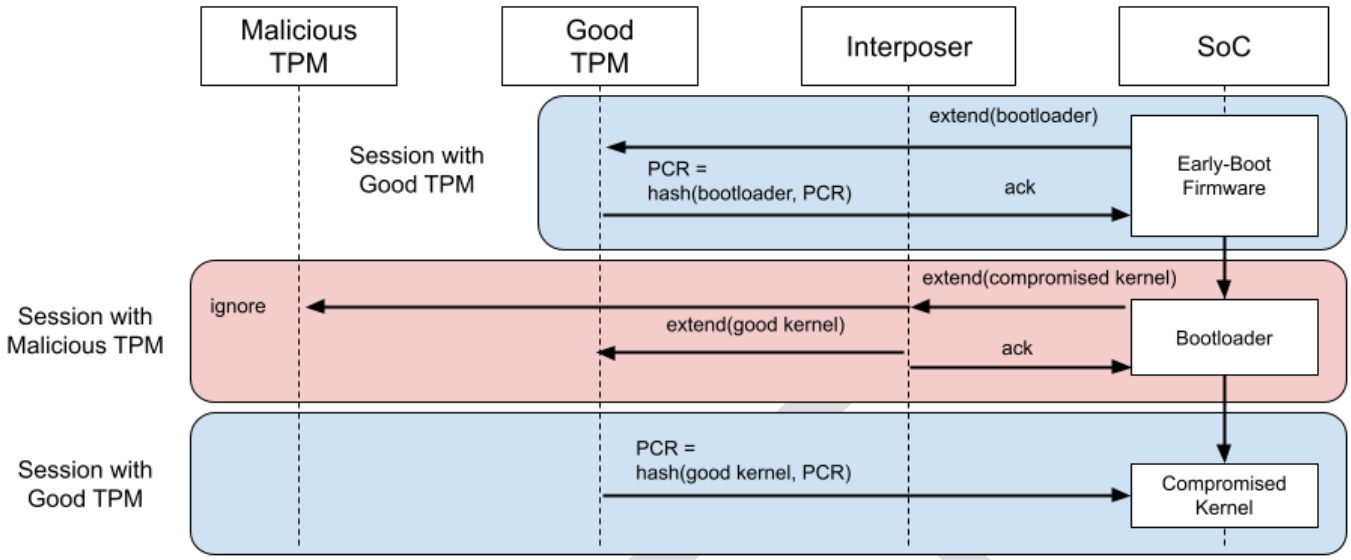


Mitigation: In addition to measuring the firmware into the discrete TPM, the SoC captures firmware measurements into the iRoT, which prevents the attacker from redirecting those measurements. Additionally, the platform firmware executing on the SoC will deposit these measurements over a secure SPDM channel, allowing it to cryptographically verify the TPM identity key wielded by the TPM. The platform firmware will then measure the public portion of this key into its iRoT's DPE during boot. Evidence of that key will be present in the SoC's DPE alias key certificate and will be factored into the alias key's derivation. Thus, a remote verifier will be able to detect the identity of the TPM which received the actual bootloader measurements, and therefore detect any attempt to redirect those measurements to an illegitimate TPM. In addition, any pre-existing TPM objects bound to the correct TPM identity will be unavailable to an attacker that attempts this redirection attack, as the SoC will be unable to wield the correct DPE key.

2.2.5 TPM Second-stage Redirection

Suppose again that platform firmware, the bootloader, and the kernel each establish their own secure session with the TPM.

An attacker may wish to run a compromised kernel on the SoC, and mask that fact by redirecting the SoC's actual (honest) kernel measurements to a separate TPM. The attacker will allow early-boot firmware to send bootloader measurements to the legitimate TPM – it must allow this to happen in order to get the legitimate TPM's identity extended into the SoC's DPE alias key. Before the bootloader runs, the attacker will swap in their own (possibly-emulated) TPM, to capture the kernel's actual (honest) measurements as reported by the bootloader. The attacker will send their own falsified (expected) kernel measurements into the legitimate TPM. Finally, when the compromised kernel runs, the attacker will swap the TPMs back, allowing the compromised kernel to issue commands to the legitimate TPM. The SoC never detected a dropped command, and so the compromised kernel has access to the keys it needs to communicate securely with the legitimate TPM.



Mitigation: Platform firmware captures the TPM's identity key and stores it in memory, propagating it forward to subsequent boot layers. Each subsequent boot layer that establishes a secure session with the TPM will sense the TPM's identity key and compare it to the key propagated by the prior layer, ensuring they are equal. Since all TPM commands are transmitted through the secure session, the bootloader will detect the initial swap, and will react by instructing the iRoT's DPE to revoke the SoC's own ability to securely communicate with the TPM until the next platform reset. The attack is further mitigated by using the iRoT for all firmware measurements.

2.2.6 Spurious TPM Reset

An attacker may wish to run a compromised kernel on a SoC, and mask that fact by resetting the TPM's PCRs after measurements of that kernel have been made into the TPM. The attacker would then be free to extend falsified (expected) kernel measurements into the TPM.

An attacker can do so by abusing the TPM's reset monitoring line to simulate a SoC reboot, which will cause the TPM to reset its PCRs to their default state. The attacker can then make arbitrary measurements into the PCRs. As the SoC never observed a dropped or modified TPM command, the SoC has not revoked its DPE alias key, meaning the SoC can wield the alias key to satisfy TPM object policies. The honest kernel measurements made by the SoC have been erased and replaced with attacker-controlled measurements.

Mitigation: In an ideal case, the verifier only trusts measurements from the SoC's iRoT and those measurements would be unaffected by a TPM reset. Since it is assumed that the system is not perfect and some measurements have to be sent to the TPM, a separate mitigation is implemented: Early-boot firmware creates an NV index with the TPMA_NV_EXTEND attribute set. This NV index is policy-bound to an SPDM key (or DICE alias key) that only early-boot firmware had access to. Such a DICE key may be established by extending a DICE layer upon completion of the early-boot phase.

The early-boot firmware may extend measurements into this NV index through the secure session, to prove that they were created when the firmware had access to the corresponding SPDM or DICE alias key. The NV index has the TPMA_NV_CLEAR_STCLEAR attribute set, meaning that if the TPM senses a host reset, it clears the NV index. The only way to re-create this NV index is to reset the host SoC, as only early-boot firmware has access to the proper DICE alias key needed to create it.

To support remote verification flows, the SPDM or DICE key to which the NV index is bound should be endorsed by the SoC's iRoT, and that endorsement should be made available to later boot stages so that it can be presented to remote verifiers.

3 DESIGN GOALS

The flow presented in this document has the following properties:

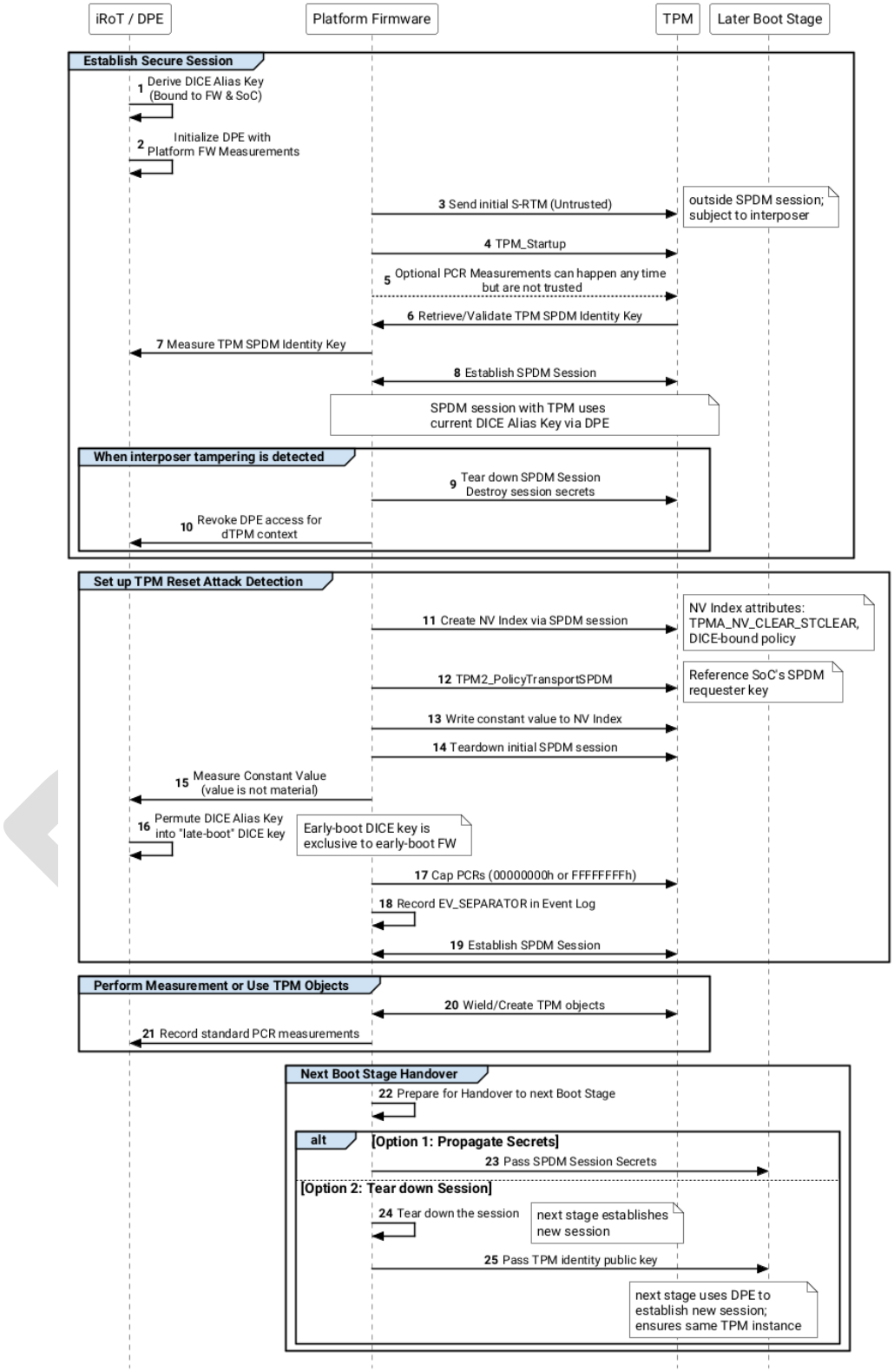
- The SoC and TPM can establish a remotely-attested, integrity-protected and encrypted session before exchanging measurements or secrets, or wielding objects.
- The TPM can bind a given object to the signing key wielded by the SoC when establishing an SPDM session. Going forward, the TPM can enforce that only an SoC with access to that same key can wield the object.
- The SoC's SPDM session establishment key, managed by DPE, is deterministic based on the SoC instance and SoC boot firmware. Thus, any TPM objects bound to that key will remain usable provided the SoC boot firmware has not changed since the TPM object was created.
- The SoC and TPM can provide evidence to a verifier that a given TPM object is bound to a combination of TPM instance, SoC instance, and low-level SoC boot code. The verifier is modeled to be a remote physically separate system.
- The boot software executed by the SoC can revoke its own access to its DICE alias key if it detects that any TPM commands have been dropped or interfered with. Command modification can be detected by way of SPDM secured message integrity tags.
- The TPM and SoC need not undergo any explicit pairing operation. The TPM may bind objects to any SoC, and it is up to a verifier to decide whether the given TPM-SoC pairing is legitimate.
- The remote verifier will be responsible for parsing DICE and EK certificate chains. The TPM will verify the DICE leaf key as part of a policy check.
- The iRoT need not directly participate in SoC → TPM communications. It need only wield DICE keys on behalf of the host software that manages the communication channel.

3.1 Non-Goals

- This flow does not address TPM object migration across SoC boot firmware updates.

4 BOOT FLOW (SPDM)

This section describes an order of operations during machine boot, in generic terms. The specific elements at play and order of operations may differ based on the platform. The following diagram shows the proposed boot flow:



The following is a textual description of the SPDM-based boot flow:

1. **Establish Secure Session**
    1. Once initialized, the iRoT derives a DICE alias key. This alias key is bound to the iRoT's firmware version and is endorsed with a certificate chain that roots back to the SoC manufacturer; a verifier can validate this certificate chain, and the measurements included within that chain, to assure themselves that the alias key may only be wielded by the given iRoT firmware image running on the given physical SoC.
    2. The iRoT firmware measures the platform firmware that runs on the main compute complex. iRoT firmware initializes its DPE instance with those measurements. The DPE instance is now ready to wield DICE keys on behalf of platform firmware.
    3. Prior to platform firmware running, the SoC may send initial S-RTM[1] measurements to the TPM. These measurements are harmless but untrustworthy, as they were sent outside of a secure SPDM session and could have been modified by an interposer.
    4. Platform firmware starts the TPM if it has not already been started.
    5. Extend TPM Session Establishment Key into DPE: Platform firmware measures the TPM's session establishment key (as defined in the document titled "TPM Communication over SPDM Secure Session" (Trusted Computing Group)) and sends that measurement to DPE. DPE will permute the SoC's alias key and capture that measurement in the SoC alias key's certificate. The SoC will then instruct DPE to wield that alias key to establish the SPDM session.
        1. This is necessary because not all host measurements will be made into DPE; some will be sent as PCR extensions to the TPM, via the SPDM session. To mitigate TPM swap attacks, the host must attest to the identity of the TPM that received those measurements. It does so by including the TPM's identity in its DPE alias key certificate.
        2. Optionally, the firmware may verify the trustworthiness of the TPM at this point. The mechanism that's used to do this is out of scope for this document. This step is not material to the security of this scheme, however it can provide an early signal to the firmware that the bus is being tampered or the system is not yet configured correctly.
        3. If the SoC re-establishes a session with the TPM, for whatever reason, it must either (a) verify the TPM's SPDM session establishment key against the one that was previously measured into DPE, or (b) measure the key into DPE again.
    6. Platform firmware establishes an SPDM session with the TPM.
    7. At any point during communication with the TPM over SPDM, if host software detects that a TPM request was not received by the TPM as-transmitted over the SPDM session, this may be evidence that an attacker is dropping or interfering with TPM messages. In response, host software tears down its SPDM session, destroys the associated session secrets, and invalidates the current DPE context until next platform reset.
2. **Set up TPM Reset Attack Protection**
    1. Platform firmware creates an NV Index in the TPM, using the ST_CLEAR attribute and a PolicySigned that can only be fulfilled if firmware wields the current DICE Alias Key. This NV Index serves as a "Reset Checkpoint" that the interposer cannot re-create if it resets the TPM and compromises the firmware running on the SoC later, since the DICE alias key is invalidated immediately after this step.
    2. Platform firmware tears down the SPDM session, permutes the SoC's DICE alias key using a constant value, and then establishes a new session using the new DICE alias key. This new key might be called the "late-boot" DICE key because it may be used for anything that happens past boot. The pre-boot DICE key shall be exclusive to early-boot firmware to protect against TPM reset attacks as explained in 2.2.6.

[1] Features such as Intel Boot Guard or AMD Platform Secure Boot often generate S-RTM measurements

3. To permute the alias key, platform firmware measures a constant into DPE. The value of the constant is not material to the flow; the requirement is that early boot code has access to a DICE alias key that later boot code does not.
4. Platform firmware caps PCRs in all active banks with the digest of 00000000h or FFFFFFFFh and records an EV_SEPARATOR event with the value of 00000000h as event data in the event log for each PCR.

3. Perform Measurements or use TPM Objects

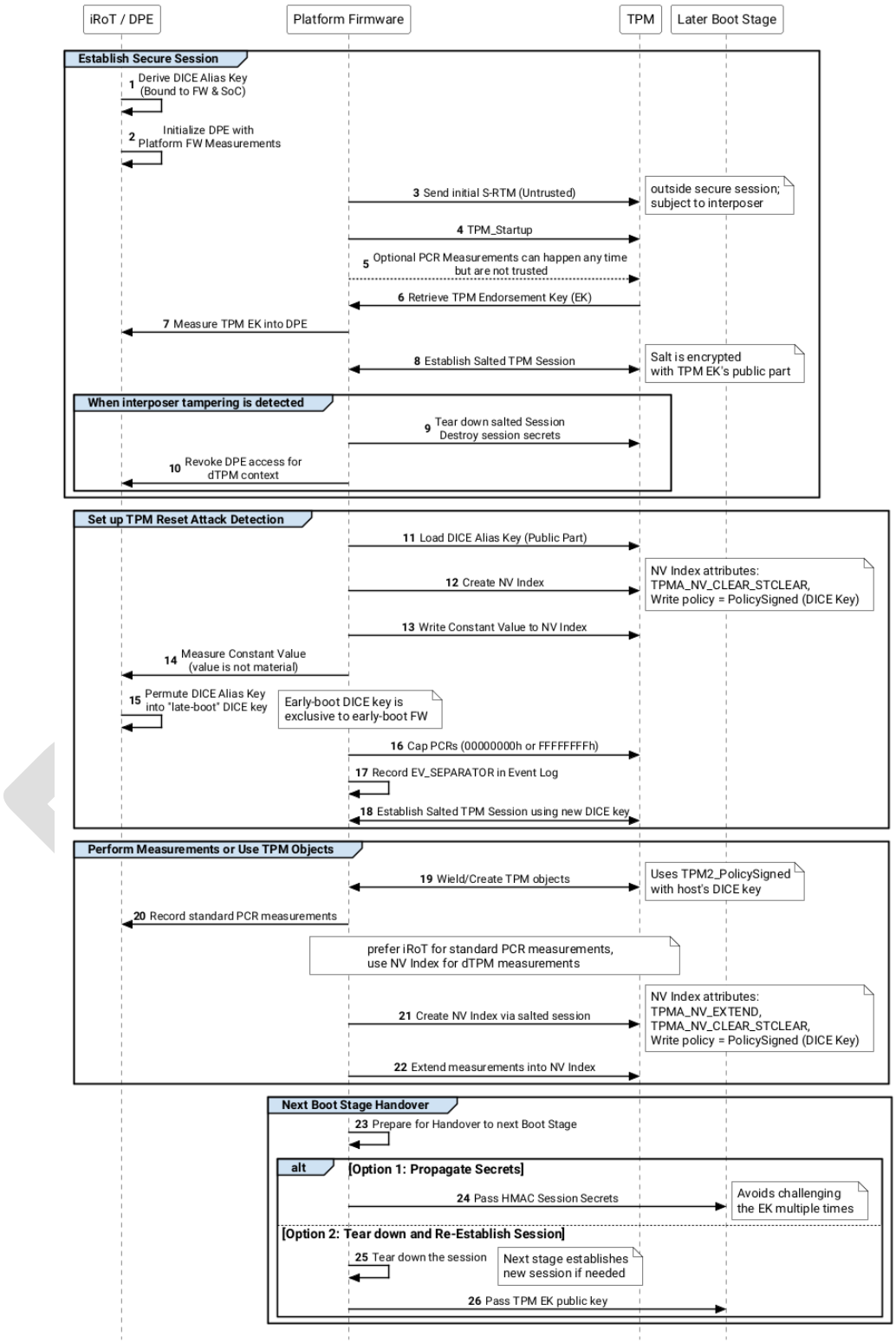
1. Platform firmware records its standard PCR measurements into the iRoT. The firmware may continue to use DPE for these measurements, or use whatever method the iRoT provides for collecting firmware integrity measurements. In general, implementations should follow the recommendations of the "Protecting Host Measurements" Chapter.
2. For example, whenever platform firmware extends measurements to the TPM via SPDM and requires the ability to prove this operation to a remote verifier, it creates an NV index and extends any important measurements into this NV index through the SPDM session. The NV index has the following attributes:
    1. TPMA_NV_EXTEND
    2. TPMA_NV_CLEAR_STCLEAR
    3. A policy that can only be satisfied if the SoC wields its current DICE alias key to establish an SPDM session with the TPM.
3. At any point in the boot flow, the SoC may wish to create or wield TPM objects that are resistant to physical adversaries. The SoC does this by way of TPM2_PolicyTransportSPDM and referencing the SoC's SPDM requester key, as defined in the TPM Library specification (Trusted Computing Group).

4. Next Boot Stage Handover

1. Platform firmware passes control to later boot stages.
    1. At this time, platform firmware may choose to propagate the SPDM session secrets forward, allowing use of the session by the next boot stage.
    2. Alternatively, platform firmware may tear down the session, in which case the next boot stage would again invoke DPE to establish a new SPDM session with the TPM. The SoC must enforce that the TPM wields the same SPDM signing key for any future SPDM sessions as was initially obtained by platform firmware.

5 BOOT FLOW (non-SPDM)

This section describes an order of operations during machine boot when SPDM is not used. Like in the previous section, the specific order of operations may differ per platform. The following diagram shows the entire non-SPDM boot flow:



The following is a textual description of the non-SPDM based boot flow:

1. **Establish Secure Session**
    1. Once initialized, the iRoT derives a DICE alias key. This alias key is bound to the iRoT's firmware version and is endorsed with a certificate chain that roots back to the SoC manufacturer; a verifier can validate this certificate chain, and the measurements included within that chain, to assure themselves that the alias key may only be wielded by the given iRoT firmware image running on the given physical SoC.
    2. The iRoT firmware measures the platform firmware that runs on the main compute complex. iRoT firmware initializes its DPE instance with those measurements. The DPE instance is now ready to wield DICE alias keys on behalf of platform firmware.
    3. Platform firmware starts the TPM if it has not already been started.
    4. Prior to platform firmware running, the SoC may send initial S-RTM measurements to the TPM. These measurements are harmless but untrustworthy, as they were sent outside of a secure TPM session and could have been modified by an interposer.
    5. Platform firmware measures the TPM's endorsement key and sends that measurement to DPE. DPE will permute the SoC's alias key and capture that measurement in the SoC alias key's certificate. The host firmware establishes a salted TPM session, using a salt that is encrypted using the TPM EK's or AK's public part (herein referred to as "TPM Identity").
        1. Using the EK might not be desirable, since there might be restrictions on the EK or the Endorsement Hierarchy. Implementations should use a suitable, asymmetric secret for this, such as the TPM AK. Security is provided by this key, as long as the verifier trusts it.
        2. Similar to the SPDM boot flow described in previous chapters, the host must attest to the identity of the TPM that received those measurements. It does so by including the TPM's identity in its DPE alias key certificate.
        3. Optionally, the firmware may verify the trustworthiness of the TPM at this point. The mechanism that's used to do this is out of scope for this document. This step is not material to the security of this scheme, however it can provide an early signal to the firmware that the bus is being tampered or the system is not yet configured correctly.
    6. Platform firmware loads the DICE alias key's public part into the TPM and creates a PolicySigned that requires host firmware to wield its current DICE alias key when it wants to use resources created in the salted session.
        1. Optionally, platform firmware uses parameter encryption to encrypt the public part of the DICE alias key to provide confidentiality
        2. Platform firmware uses the HMAC response to verify that the TPM actually executed the original command. This way, the interposer can not substitute the public portion of the DICE alias key without corrupting the command and the interposer can't route the command to a different TPM either
2. **Set up TPM Reset Attack Protection**
    1. Platform firmware creates an NV Index in the TPM, using the ST_CLEAR attribute and a PolicySigned that can only be fulfilled if firmware wields the current DICE Alias Key. This NV Index serves as a "Reset Checkpoint" that the interposer cannot re-create if it resets the TPM and compromises the firmware running on the SoC later, since the DICE alias key is invalidated immediately after this step.
    2. Then, platform firmware permutes the SoC's DICE alias key using a constant value. This new key might be called the "late-boot" DICE key because it may be used for anything that happens past boot. The pre-boot DICE key shall be exclusive to early-boot firmware to protect against TPM reset attacks as explained in 2.2.6.

1. To permute the alias key, platform firmware measures a constant into DPE. The value of the constant is not material to the flow; the requirement is that early boot code have access to a DICE alias key that later boot code does not.
2. Platform firmware caps PCRs in all active banks with the digest of 00000000h or FFFFFFFFh and records an EV_SEPARATOR event with the value of 00000000h as event data in the event log for each PCR.

3. Perform Measurements or use TPM Objects

1. Platform firmware records its standard PCR measurements into the iRoT. The firmware may continue to use DPE for these measurements, or use whatever method the iRoT provides for collecting firmware integrity measurements. In general, implementations should follow the recommendations of the "Protecting Host Measurements" Chapter.
2. For example, whenever platform firmware extends measurements to the TPM via the salted session and requires the ability to prove this operation to a remote verifier, it creates an NV index and extends any important measurements into this NV index through the salted session. The NV index has the following attributes:
    1. TPMA_NV_EXTEND
    2. TPMA_NV_CLEAR_STCLEAR
    3. A write policy that can only be satisfied if the SoC wields its current DICE alias key
3. At any point in the boot flow, the SoC may wish to create or wield TPM objects that are resistant to physical adversaries. The SoC does this by way of TPM2_PolicySigned using the host's DICE key.

4. Next Boot Stage Handover

1. Platform firmware passes control to later boot stages.
    1. At this time, platform firmware may choose to propagate the HMAC session secrets forward, allowing use of the session by the next boot stage. This avoids challenging the TPM Identity multiple times.
    2. When establishing new sessions instead, the firmware needs to ensure that the TPM still wields the same Identity (EK or AK) as was used in previous sessions and measure it into DPE before establishing the session.

6 PROTECTING HOST MEASUREMENTS

Because implementations face varying tradeoffs regarding requirements for other hardware components, and the availability of specific firmware interfaces, it is impractical to provide universal recommendations on how to treat host measurements. A significant design choice involves verifier integration: depending on the selected method, a remote verifier may or may not be able to cryptographically prove that measurements were transmitted through a secure session. Consequently, implementers must evaluate these tradeoffs against the following order of preference:

- **Use only DPE for Measurements:** The most preferred method is to capture measurements directly within the integrated Root of Trust (iRoT) using DPE. The verifier already trusts DPE measurements and infers the fact that they cannot be interposed from the iRoT's protected location in the SoC's chip package.
- **Within SPDM session to TPM**: When sending measurements through the secure session, the platform must provide evidence allowing a remote verifier to cryptographically verify that the host measurements were transmitted through the secure session. Implementers may choose from the following methods to establish this proof:
    - **Policy-Bound NV Index:** The platform records measurements into an NV index configured with the TPMA_NV_EXTEND attribute, bypassing PCRs entirely. This index must be bound via policy to the SoC's DPE or session establishment key. The verifier evaluates the NV index state and its associated policy to guarantee that the measurements could only have been recorded by the entity controlling the SPDM session
    - **Audited PCR Extensions:** The platform extends PCRs within a TPM audit session and within the secure session. The remote verifier evaluates the resulting audit digest, which cryptographically proves that the specific TPM2_PCR_Extend commands were executed within the secure session.
- **Within Salted Session:** Environments that do not support SPDM can use EK-salted sessions as described in the non-SPDM flow. Here, the same options exist as in the SPDM session case: the implementation can use policy-bound NV Indexes or an audit session.

Regardless of which option above is used, DPE must contain evidence of at least all firmware and configuration required to communicate with the TPM.

Protecting host measurements also requires mitigating **TPM reset attacks**, which an adversary might use to clear honest measurements as described in the Threat Model Chapter. Platform firmware can address this through the following mechanisms:

- **ResetCheckpoint NV Index:** Early-boot firmware creates an NV index (with the TPMA_NV_CLEAR_STCLEAR attribute) bound by policy to the SoC's specific DPE key. By extending a constant value into this index and by invalidating the DPE key subsequently, the platform can cryptographically prove that the TPM has not been reset since early boot, as only the early-boot firmware possesses the key required to recreate the index.
- **Persistent Host Sessions:** Maintaining a persistent secure session between the host and the TPM ensures continuous communication security across boot stages. This can be implemented in two ways:
    - Platform firmware hands off the secure session secrets directly to subsequent firmware stages during execution transfer.
    - The iRoT stores SPDM session secrets and exposes them by proxy.

7 ATTESTATION

If implemented, the proposed mitigations create a remotely attested session: A verifier examines evidence of which keys were used to establish the session and determines whether to trust the keys and any objects or measurements that have been used or created in the context of the established session.

In particular, host software may wish to provide evidence to a verifier that a given TPM object is bound to a particular SoC instance and SoC boot configuration. The SoC does this by providing the following:

1. The results of TPM2_Certify, providing evidence of the TPM to which the object is bound, along with the object's policy hash. The policy hash includes a TPM2_PolicyTransportSPDM or TPM2_PolicySigned assertion, which provides evidence of the SoC's DICE key that the host must control to wield the object.
2. The SoC's DICE certificate chain as provided by DPE, providing evidence of the SoC instance and SoC boot configuration that controls the DICE key found in the object's policy hash. This certificate chain also includes evidence of the TPM which the SoC believed it was communicating with throughout boot, as this was measured into DPE previously.
3. Host software may also provide any other evidence required to satisfy the verifier, such as the TCG event log in cases where the object is bound via TPM2_PolicyPCR. Note that while objects can always be bound to the TPM's PCR values by policy, the verifier would avoid trusting TPM PCR values for determining the integrity of the firmware running on the SoC. Instead, the verifier would only trust firmware measurements from the iRoT, or values extended into a policy-bound TPM NV index via the secure session.

The verifier would determine whether the keys that were used during session establishment are trustworthy by verifying the TPM2_Certify result. Combined with a platform certificate and one of the TPM's EK certificates, the verifier can validate the following information:

- The TPM is a trustworthy TPM that wields a valid EK and its EK certificate is signed by a trustworthy manufacturer
- The TPM wields a trustworthy EK, since its EK certificate's serial number is referenced in the platform certificate
- The SoC's DICE certificate chain roots in a device certificate that is known to the verifier or referenced by the platform certificate, which means that the verifier can assign an identity to the SoC and verify its DICE certificate chain
- The verifier calculates whether it should trust any objects created within the secure session by verifying the TPM2_Certify response and the associated policies. In particular, the policies reference the SoC's DICE alias keys that were used during boot.

In order for the verifier to ensure that the evidence provided in the above flow is fresh (i.e. that the TPM object's policy describes the state that the host is currently in), the verifier must obtain fresh evidence that the host can wield the TPM object in question. For example, if the TPM object is a signing key, the verifier can receive a nonce signed by that key. Alternatively, if the TPM object is a sealing key, the key can be used to seal a separate credential, where the host wielding that credential provides implicit proof that its current boot state matches the TPM object policy.

8 REFERENCES

DMTF. (2022). *Secured Messages using SPDM Specification - DSP0277.* Retrieved from https://www.dmtf.org/sites/default/files/standards/documents/DSP0277_1.1.0.pdf

DMTF. (n.d.). *Security Protocol and Data Model - DSP0274.* Retrieved from https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.2.0.pdf

Trusted Computing Group. (2023, May). *CPU-TPM Bus Protection Guidance - Active Attack Mitigations.* Retrieved from https://trustedcomputinggroup.org/resource/tcg-cpu-to-tpm-bus-protection-guidance-for-active-attacks/

Trusted Computing Group. (2023). *DICE Protection Environment.* Retrieved from https://trustedcomputinggroup.org/wp-content/uploads/TCG-DICE-Protection-Environment-Specification_14february2023-1.pdf

Trusted Computing Group. (2025). *TCG CPU TPM Bus Protection Guidance Passive Attack Mitigation.* Retrieved from https://trustedcomputinggroup.org/resource/tcg-cpu-tpm-bus-protection-guidance-passive-attack-mitigation/

Trusted Computing Group. (n.d.). *DICE Layering Architecture.* Retrieved from https://trustedcomputinggroup.org/resource/dice-layering-architecture/

Trusted Computing Group. (n.d.). *TPM 2.0 Library Part 3: Commands, Version 185.* Retrieved from https://trustedcomputinggroup.org/resource/tpm-library-specification/

Trusted Computing Group. (n.d.). *TPM Communication over SPDM Secure Session.* Retrieved from https://trustedcomputinggroup.org/resource/tpm-communication-over-spdm-secure-session/

