PCI Express® Device Security Enhancements

September 2018
Version 0.71

Abstract

PCI Express® (PCIe) Devices may be composed of hardware (immutable) and firmware (immutable and mutable) components. Presently, Vendor ID/Device ID/Revision ID registers convey the hardware identify of a PCIe Device and there is no defined mechanism to convey the firmware identity of a PCIe Device. In addition to the Device identity, the PCIe specification defines various types of Capability structures to convey PCIe Device feature capabilities. Both the Device identity and capability can be spoofed and used maliciously by an adversary.

This specification introduces the notion of *PCIe Device Measurement*, a method of exposing the identity of Device firmware. The *Device Measurement* mechanism used in isolation, however, is subject to supply chain attacks such as counterfeiting and can also be spoofed by an advanced adversary. Additionally, this specification introduces the notion of *PCIe Device Authentication*, which uses public key cryptography to defend against such attacks and to provide higher assurance about the hardware and firmware identities and capabilities. *PCIe Device Authentication* adapts the existing USB Type-C Authentication mechanisms to PCIe---the new elements are the specific PCIe register interface and the associated mechanisms, plus some details that are necessarily specific to PCIe.

PCIe Device Measurement and Device Authentication results can be used in various scenarios, such as: 1) a data center administrator can ensure all PCIe Devices are running appropriate firmware versions, 2) system software can ensure a trusted Device is plugged in before enabling the PCIe Address Translation Services (ATS) for the Device. *PCIe Device Authentication* provides platforms a way to make trust decisions about specific Devices. This in turn provides value to Device vendors because the Authentication feature is itself a valuable Device feature, and supports the detection of counterfeit and potentially malicious Devices by platform verification software.

This specification details the requirements, interface and protocol for *PCIe Device Measurement* and *PCIe Device Authentication*. It also provides general guidelines for implementing these technologies in practice.

1 Introduction

This specification describes the architecture of *PCIe Device Measurement* and *PCIe Device Authentication*. This specification is intended to be used by the vendors of PCIe Devices to implement support for these features in their Device, as well as by the Host software vendors to implement the matching functionality in their software components.

A key first step in improving platform security is to check the identity of a Device and the identity of the firmware components running on the Device. The PCIe specification provides a mechanism for Host software to identify a Device, through the Vendor ID and Device ID in the PCI Configuration Space Header, but this approach is not inherently secure. Additionally, many PCIe Devices include one or more execution logics that operate independently of the Host, but there is no defined mechanism for the Device to expose the firmware identity running on the execution logics. *PCIe Device Measurement*, introduced in this specification, defines a method by which the firmware images running on these execution logics are included in measurements that are exposed to the Host. Today, enterprise client computing systems as well as datacenter installations highly value the ability to validate that a given machine is running expected software versions and has not been compromised. Various mechanisms exist to attest to the validity of platform ingredients, such as the UEFI/BIOS, as well as other active/measureable components built into modern motherboards. *PCIe Device Measurement* allows the firmware identity of PCIe Devices to be included as part of the platform ingredients for enhanced platform assurance.

Industry desire to cryptographically check the *identities* and *capabilities* of a particular Device have grown considerably with the consolidation of computing resources in the cloud and data center environment, where tens of thousands of computing Hosts along with several PCIe-based Devices plugged into each of these Hosts are deployed. Oftentimes these PCIe Devices are procured through a trusted supply chain to ensure genuine Devices are used on the Hosts. However, there have been cases of counterfeit PCIe Devices slipping through the supply chain. These counterfeit PCIe Devices could have their hardware components or firmware components replaced with cheaper or less reliable parts, causing interruptions to the cloud or data center operations. *PCIe Device Authentication*, introduced in this specification, can help mitigate against such attacks. *PCIe Device Authentication* can be applied wherever higher assurance about the PCIe Device behavior is needed. For example:

I. Remote system administrators that manage a large collection of systems, each containing one or more Devices, can dynamically generate a manifest of cryptographic identities of all Devices without physical examination of the systems. The manifest can also be stored for auditing purposes.

II. Low-level system software, e.g., Virtual Machine Monitor (VMM), can establish the identities of the Devices on the system before assigning the Devices to Virtual Machines.

III. When a Device implements the Address Translation Cache (ATC) to achieve the performance benefits given by the Address Translation Services (ATS), the Device is given the privilege to cache the address translation results on the Device. In order for the Host to make the trust decision to grant the privilege to the Device, the Device can authenticate to the Host its unique cryptographic identity.

IV. Runtime/hot-swap verification of PCIe Devices, without requiring host reboot, where the identity of the PCIe Device is verified by OS drivers before assigning resources to the Device.

In summary, *PCIe Device Measurement and Device Authentication* frameworks set up the foundation where the Device identify and capability can be verified, with future expansions for when 1) the identity of the Host can be verified by the Device or, 2) the identity of the Host and the Device can be verified by each other or, 3) the Host and the Device can exchange secrets to set up a secure channel between each other, as illustrated in Figure 1.

1.1 Reference Documents

Document	Location
USB Type-C™ Authentication Specification Rev. 1.0	http://www.usb.org/developers/docs/
MCTP Base Specification	https://www.dmtf.org/standards/published_documents
MCTP PCIe VDM Transport Binding Specification	https://www.dmtf.org/standards/published_documents
MCTP SMBus/I2C Transport Binding Specification	https://www.dmtf.org/standards/published_documents
TCG DICE Architectures	https://trustedcomputinggroup.org/work-groups/dice-architectures/
TCG Glossary	https://trustedcomputinggroup.org/wp-content/uploads/TCG-Glossary-V1.1-Rev-1.0.pdf

1.2 Terminology

Selected terms and acronyms used in this specification are defined below.

DICE	Device Identifier Composition Engine
FW	Firmware
GPA	Guest Physical Address
HPA	Host Physical Address
IOMMU	Input / Output Memory Management Unit
MCTP	Management Component Transport Protocol
Measurement	Cryptographic hash (for example, SHA256)
PF	Physical Function
PCIe	PCI Express
ROM	Read-only Memory
RoT	Root of Trust
RTM	Root of Trust for Measurement
RTR	Root of Trust for Reporting
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TPM	Trusted Platform Module
USB	Universal Serial Bus
VF	Virtual Function

The terms “*shall*” and “*must*” are used to describe mandatory requirements. The terms “*should*” and “*may*” are used to identify optional requirements.

1.3 Changelog

Version	**Date**	**Description**
Version 0.5	March ‘18	First draft release in intel.com, describing device measurements and device authentication architectures.
Version 0.7	June ‘18	Draft version with following major additions: • Single digest register for exposing multiple firmware measurements (Section 3.2.2). • Authentication over MCTP (Section 4.6).
Version 0.71	September ‘18	Draft version with following major additions: • Message definitions modified to be compatible with the out-of-band mechanisms defined by the Cerberus specification (Section 4.10.2). • Added device ownership transfer (Section 5.4.2).

2 Background

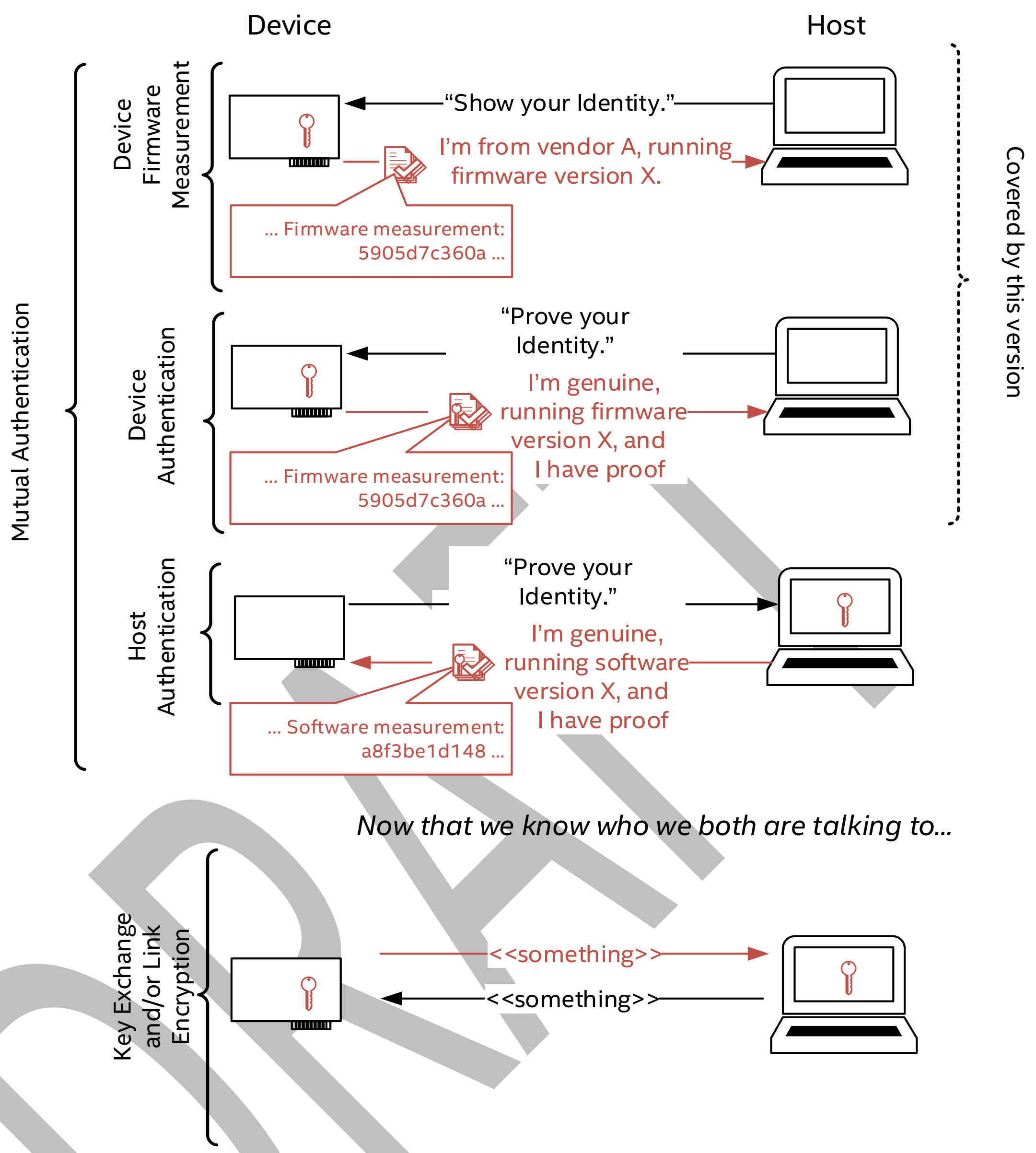
This section provides the necessary background information for *PCIe Device Measurement* and *Device Authentication*, including the overall trust relationships for PCIe Devices, as well as the *PCIe Device Authentication*'s relationship to the USB Type-C Authentication Specification.

2.1 Trust Relationships for PCIe Devices

The motivating examples described in Section 1 can be summarized as a trust relationship between the Host and the PCIe Devices in a system. The Host needs a certain mechanism to determine the *identity* and *capability* of the PCIe Device to make a trust decision, to simply verify that the Device is running approved firmware versions, to grant the Device certain privileges, or to share a secret with the Device. There are established industry paradigms for identity and capability mechanisms, and this specification focuses on the adaptation to PCIe of existing and well-understood approaches, and avoids inventing new elements wherever possible.

Figure 1 depicts various Host-Device relationships for authentication purposes, where:

- The Host can query the Device's firmware version that is *not* tied to the Device private key, termed Device Measurement in this specification. This is covered under this specification as part of the Device Measurement (Section 3).
- The Host can query a Device's hardware and firmware *identities* tied to a Device private key, termed Device Authentication in this specification. This is covered under this specification as part of the Device Authentication (Section 4).
- The Device can challenge the Host (in-band) for proof of identity that is tied to the Host private key, termed Host Authentication in this specification. The device could then provide additional capabilities based on trust decisions coming from obtaining the proof of the Host's identity. The device should not restrict the basic functionality required by the Host to complete the boot process. Such a mechanism may be covered by future revisions of this specification.
- The Device and Host can exchange secrets *after* verifying the identity (and capability, if needed), for link encryption or other security purposes, termed Key Exchange in this specification. Such mechanisms may be covered by future revisions of this specification.



Device
Host
Mutual Authentication
Device Firmware Measurement
"Show your Identity."
I'm from vendor A, running firmware version X.
... Firmware measurement: 5905d7c360a ...
Device Authentication
"Prove your Identity."
I'm genuine, running firmware version X, and I have proof
... Firmware measurement: 5905d7c360a ...
Covered by this version
Host Authentication
"Prove your Identity."
I'm genuine, running software version X, and I have proof
... Software measurement: a8f3be1d148 ...
Now that we know who we both are talking to...
Key Exchange and/or Link Encryption
<<something>>
<<something>>

Figure 1: Host-Device trust relationships.

Per Figure 1, an essential first step is to collect the measurements of the Device firmware. This enables the Host to verify that the Device is executing the correct firmware. For verification that the measurements have not been tampered with, Device Authentication is needed in order for *any* trust decision to be made, so Device Measurement is a critical enabling component of Device Authentication. This specification defines Device Measurement and Device Authentication mechanisms for PCIe Devices, and defines the architecture to allow future expansion to Host and/or mutual authentication as well as key exchange and/or link encryption.

2.2 Relationship to USB Type-C Authentication

The PCIe Device Authentication follows the flow of the USB Type-C Authentication Specification (see Section 1.1 for a link) for the Authentication Architecture, Authentication Protocol and Authentication Messages defined by the USB Type-C Authentication Specification. Specifically, the following sections of the USB Type-C Authentication Specification are used *as-is*, *unless otherwise noted in this specification in Section 4*.

- ✓ Section 3 Authentication Architecture of USB Type-C Authentication Specification.
- ✓ Section 4 Authentication Protocol of USB Type-C Authentication Specification.
- ✓ Section 5 Authentication Messages of USB Type-C Authentication Specification.

The following sections of the USB Type-C Authentication Specification are USB-specific and do not apply to PCIe Device Authentication.

- ✗ Section 6 Authentication of PD Products of USB Type-C Authentication Specification.
- ✗ Section 7 Authentication of USB Products of USB Type-C Authentication Specification.

The majority of the PCIe Device Authentication architecture outlined in Section 4 can be seen as an equivalence to the above two USB-specific sections, to describe how the Authentication Architecture, Authentication Protocol and Authentication Messages are mapped to the PCIe architecture. This document assumes that the reader is familiar with the Authentication Architecture, Authentication Protocol, and Authentication Messages defined in the USB Type-C Authentication Specification.

There are multiple benefits of leveraging the existing USB Type-C Authentication definition:

- The same software implementation can be used for both USB and PCIe Device Authentication.
- The same hardware implementation block can be used for both USB and PCIe Devices for Authentication purposes.

3 PCIe Device Measurement Architecture

PCIe Devices may be composed of immutable and mutable elements. Examples of immutable elements are hardcoded logic, and ROM-based microcode or firmware. Examples of mutable elements are reprogrammable logic (FPGA), and reprogrammable microcode, firmware/software. It is common for implementations to include a CPU or a microcontroller or other programmable element such as FPGA logic, which we will refer to as an “execution logic”, whereas the mutable elements (e.g., programming information) will be referred to as “firmware”. Examples of firmware components include (but not limited to) boot controller startup code, microcontroller kernel, or bit files programmed on an FPGA device. Furthermore, firmware components can include not only code and data segments, but also persistent data that makes up the overall firmware identity.
For a PCI/PCIe Function, the Vendor ID/Device ID/Revision ID registers and other mechanisms such as Class Code or Subsystem Vendor ID convey the hardware identify of a PCIe Device. *PCIe Device Measurement* complements this by exposing the identity of Device firmware. This section describes the architecture for *PCIe Device Measurement*, the first and necessary step to establish a standard mechanism for the Host to query the firmware identity running on the execution logics of PCIe Devices.

3.1 Threat Model

PCIe Device Measurement considers all immutable components of a Device to be trusted, whereas any mutable firmware is not. Therefore, it is assumed that an would-be attacker can access and/or modify mutable firmware and any associated values, parameters or saved contexts along with the mutable firmware, but not ROM components. An attacker with physical access to the Device that is able to replace or swap either the Device hardware or ROM components is considered to be out-of-scope for the PCIe Device Measurement architecture.

3.2 PCIe Device Measurement Requirements

3.2.1 DIGEST Register

Each Device shall implement one or more DIGEST register(s) that expose the cryptographic identities of all of its firmware components. Every mutable firmware component that the Device executes must be measured in at least one DIGEST register. In this context, cryptographic identity implies that a secure hash function is applied to the firmware images. The Device manufacturers should select the algorithm based on the prevailing recommendations, such as those from NIST (NIST's Policy on Hash Functions). The DIGEST register itself is considered part of the Root-of-Trust for Measurement (RTM) and therefore the underlying implementation of this register cannot have any dependency on mutable firmware. Additionally this register must be implemented such that the mutable portion of the firmware cannot modify the register value or intercept Host accesses to this register.

The Host software may make use of the DIGEST value in a variety of ways, and the specifics are outside the scope of this specification.

Implementation options for the DIGEST register are described in the following subsections.

3.2.1.1 Single DIGEST Per Execution Logic

One implementation option is to use a single DIGEST register to capture all firmware components associated with an execution logic. As each component gets measured, the DIGEST register is updated to reflect the component identity. The recommended way to do that is to implement this register such that a write causes it to be extended using an appropriate secure hash algorithm, instead of it being overwritten.

For example, a SHA256 EXTEND operation is defined as:

REGISTER = SHA256 (REGISTER || measurement)

where || represents the concatenation operation, and *measurement* is a SHA256 hash computed over the firmware component image and SHA256 is defined in NIST FIPS PUB 180-4. Effectively, the new measurement value, which is an output of the SHA256 operation, is concatenated with the current register value and passed through the SHA256 operation. The result is stored as a new value of the register.

Since this register is part of the RTM, the EXTEND operation above must be implemented entirely in hardware or in immutable firmware. An implementation where mutable firmware component reads the current value, concatenates the new measurement and computes SHA256 over it, is considered *non-compliant* with this specification. This is because such an implementation implicitly trusts the mutable firmware component and the host cannot tell, through cryptographic means, if that component is correctly behaving or not. Nevertheless, mutable firmware components can request extension of the measurements, to allow a chain of measurements to be captured in a single DIGEST register.

Alternatively, as opposed to using the EXTEND operation defined above, multiple measurement values can be reported via a single DIGEST register field by the Device reporting the total number of measurement for one capability structure through the NUM_DIGEST field and selecting specific measurement value by programming the DIGEST_SEL field.

After a reset that causes firmware reload, the DIGEST register shall default to 0. Some Devices may reload firmware on a warm reset, whereas others may require a cold reset or PCIe D3 transition.

If the boot flow consists of a single firmware stage, the EXTEND functionality is not needed. ROM can compute the digest, hash it with the ROM version and write the resulting value to the DIGEST register.

3.2.1.2 Multiple Registers Per Execution Logic

An alternate implementation option is to implement multiple DIGEST registers, one per firmware stage or per hashing algorithm. In this case, multiple instances of the *Device Measurement* PCIe capability register block (see Section 3.2.2) must be implemented.

3.2.2 PCIe Measurement Register Interface

The measurement(s) is/are exposed through the optional Digest PCIe extended capability structure(s). Digest structures must appears in the Physical Function (PF) that is associated with the execution logics. It is permitted for a single PF to implement multiple instances of this capability structure. There are several reasons why this might be appropriate:

- The PF, for Device-specific reasons, may be implemented to divide the firmware components into multiple groups and report digest values for each group separately. In this case, each instance of the Digest Capability structure (Table 2) must be associated with a unique Firmware ID (byte offset 11, Bits 4:0). Below are some concrete examples of why a Device may do this, but this is by no means an exhaustive list.
    - The PF is associated with multiple independent execution logics. The Device manufacturer prefers to report the measurement of each firmware image separately. As a result, the PF instantiates one Digest Capability structure per execution logic.
    - The firmware load process may consist of *n* stages. The PF may choose to report the measurement of each stage separately.
- The Device supports computing and reporting its measurement using more than one secure hash algorithm ("cryptographic algorithmic agility"). In this case, the PF instantiates one Digest Capability structure per supported algorithm for every firmware ID.

Host software shall not draw any conclusions about the internal architecture of the Device based on the number of such structures.

Virtual Function (VF) configuration space must not implement the Digest Capability structure.

A Device shall complete all measurement process and assert DIGEST_VALID and ALL_DIGESTS_VALID within one second after reset.

Table 1 provides the respective bit definitions of the register fields in the Digest Capability Header; Table 2 provides the respective bit definitions of the registers fields in the Digest Capability Structure.

Table 1: Digest Capability Header.

Bit Locations	**Register Description**	**Register Attribute**
15:0	PCIe Capability ID - Must be 023h (DVSEC).	RO
19:16	Capability version, must be 1.	RO
31:20	Next Capability offset---See PCIe Base Specification for definition.	RO
47:32	DVSEC Vendor ID---This field is the Vendor ID associated with the vendor that defined the contents of this capability. Must report 8086h, the Intel vendor ID assigned by PCISIG.	RO
51:48	DVSEC Revision---This field is a vendor-defined version number that indicates the version of the DVSEC structure. Implementations that comply with this version of the specification must report 1.	RO
63:52	DVSEC Length---This field indicates the number of bytes in the entire DVSEC structure.	RO

	Must return 16+*N* where *N* is the length of the digest field in bytes.	
79:64	DVSEC ID---This field is a vendor-defined ID that indicates the nature and format of the DVSEC structure. Implementations that comply with this version of the specification must return 03Eh. This DVSEC ID is assigned by Intel corporation.	RO

Table 2: Digest Capability Structure.

Byte Offsets	**Register Description**	**Register Attribute**
9:0	PCIe Capability header including DVSEC header bytes– See Table 1.	RO
10	Bit 7:2---Reserved for future use. Bit 1 (ANY_DIGEST_MODIFIED). If NUM_DIGEST is non-zero, the controller shall set this bit whenever it modifies the measurement values of all values that can be pointed to by the DIGEST_SEL field. Host software must not be able to set this bit. Host software may clear this bit by writing one to it. This bit must be set to 1 by hardware after a reset or power transition that results in all measurements being cleared. Bit 0 (DIGEST_MODIFIED). The controller shall set this bit whenever it modifies the contents of the DIGEST register that is pointed to by the DIGEST_SEL field. This bit must read 1 before the change to the DIGEST field is made visible to the Host. Host software must not be able to set this bit. Host software may clear this bit by writing one to it. This bit must be set to 1 by hardware after a reset or power transition that results in the DIGEST field being cleared.	RW1C
11	Bit 7 (DIGEST_VALID) = 1 implies that the measurement process is complete and the DIGEST field holds the final measurement value that is pointed to by the DIGEST_SEL field. This field can be set by the controller firmware, but is read-only to Host software. This bit must be cleared by hardware after a reset or power transition that results in the DIGEST field being cleared. Bit 6: (ALL_DIGESTS_VALID) = 1 implies that the measurement process is complete and all measurement values that can be pointed to by the DIGEST_SEL field are final and valid. This field can be set by the controller firmware, but is read-only to Host software. This bit must be cleared by hardware after a reset or power transition that results in the DIGEST field being cleared. Bit 5---Reserved for future use. Bits 4:0---Firmware ID. Each ID represents the logical grouping of firmware components that are measured as a single unit. The	RO

	mapping between the Firmware ID field and the underlying firmware component is Device-specific. It is recommended that this field be implemented in hardware and controller firmware is not able to modify it. Host software can assume that this field is static. All possible 5-bit values are available except for 0x1F; the Firmware ID value 0x1F is used to indicate a non-existent firmware component. If the firmware component to be measured cannot be found, the Device should assert DIGEST_VALID and set Firmware ID to 0x1F at the same time.	
13:12	TCG_ALG_ID---The algorithm used for computing DIGEST. Enumeration is defined in the *TCG Algorithm Registry*. It is recommended that this field be implemented in hardware and controller firmware is not able to modify it. Host software can assume that this field is static. If the controller implements algorithmic agility, it will expose *n* instances of this capability structure for each Firmware ID, where *n* is the number of supported algorithms.	RO
14	NUM_DIGEST---This field indicates the number of firmware component measurements reported by this capability structure. A value of *K* in this field indicates that (*K + 1*) measurements are reported.	RO
15	DIGEST_SEL---If NUM_DIGEST is non-zero, this field selects the Firmware ID measurement reported in the DIGEST field. A value of *M* in this field selects the measurement value for (Firmware ID + *M*). Default value for the DIGEST_SEL field is zero.	RW
15+*N*:16	DIGEST---Holds the digest of the firmware components that are measured as a single unit per the rules in section 3.2.1. The Firmware ID field identifies the logical group number of firmware components that are measured as a single unit. *N* is the length of the digest in bytes. For example, *N* = 32 if the digest represents a SHA256 measurement of the firmware image.	RO

3.2.3 Race Conditions

3.2.3.1 Race Condition with Host Boot

The execution logics and the Host may boot in parallel. For PCIe controllers, the Host must poll for the DIGEST_VALID and ALL_DIGESTS_VALID flags prior to consuming the DIGEST value. This will ensure that the pre-OS boot software does not consume an intermediate value.

3.2.3.2 Race Condition with Execution Logics Reset

The Digest register read operation is unlikely to be atomic from the Host's perspective. A controller may support reset of an execution logics or reload of execution logics firmware without Host interface reset (Sections 3.2.5 and 3.2.6). In addition, an execution logics reset can happen due to a variety of other reasons that are not under the control of the Host software, e.g., fatal error condition, out-of-band command, watchdog timer expiry, etc. If an execution logic is reset without Host interface reset, it is possible that the DIGEST register content may change while the Host software is in the middle of the read operation. DIGEST_MODIFIED and ANY_DIGEST_MODIFIED flags help avoid this race condition.

Host software shall clear DIGEST_MODIFIED and ANY_DIGEST_MODIFIED flags before reading the DIGEST. After software has read all the bytes in the DIGEST register, software must consult DIGEST_MODIFIED. If DIGEST_MODIFIED=0, the software can assume the DIGEST value is valid. If DIGEST_MODIFIED=1, the software must discard the previously read DIGEST value, clear DIGEST_MODIFIED, wait for DIGEST_VALID to be set and re-read DIGEST.

3.2.4 Device Configuration and Policy

It is recommended that the dynamic variable portion of the Device configuration and policy data not be included in the measurement. This is to prevent the measurement value from becoming brittle and hence less useful. Security software can use Device-specific mechanisms to access dynamic device configuration and policy data if needed. It is assumed that any firmware code that is involved in returning the configuration and policy data is part of the previous measurement, thus included in the measurement and hence considered trustworthy.

Any static configuration data/policy data that is set up by the Device/system vendor and not under end user control may be considered as firmware components and be included in the DIGEST. For example, debug modes can be incorporated into the Measurement. See Section 3.2.7 for more discussions on the debug modes.

3.2.5 Device Power Transition and Reset

The Device implementation must make sure that the DIGEST register accurately reflects the firmware identity. For example, if the same firmware stack is reloaded after a D3Hot->D0 transition, the DIGEST register value, if read when DIGEST_VALID=1, must not change across this transition. One way to achieve that is to reset the DIGEST register to 0 and the execution logics re-executes from the immutable firmware so that it gets restored to the correct value during firmware load.

3.2.6 Runtime Active Image Update without System Reset

Some Devices may allow a runtime update of the active image without a system reset. Some Devices may support live patching, where part of the firmware may be patched without system reset. In these scenarios, the value of at least one of the DIGEST registers must be reset and updated to reflect the new firmware stack. In this case, the Device must re-execute the RTM in conjunction with the reset of the DIGEST register in order to update the DIGEST register.

If and when the re-execution of the RTM is not possible or when a non-disruptive live update is required, extra DIGEST registers and Firmware IDs can be assigned to capture the measurement values for the live update images. For example, currently running firmware can compute the digest of new live update firmware components and write the measurement values and Firmware IDs to the extra DIGEST registers before transferring control to the live update firmware components.

3.2.7 Debug Mode

Some Devices allow an invasive debug mode whereby the debugger is granted low level access to the hardware and may be able to influence the security properties of the Device. Such an invasive debug mode should be contrasted with a general debug mode where, for example, a software debugger can be run to collect debug traces. A general debug mode would allow debug information to be exposed to debug a functional issue. On the other hand, an invasive debug mode would allow both the functional features and security properties of a Device to be altered, e.g., signature verification could be bypassed for firmware components in an invasive debug mode. In other words, such a debug mode may allow the debugger the ability to influence the measurement process itself. While entering such invasive debug modes, it is important that the previous DIGEST values are invalidated and reset to 0. A Device should report that an invasive debug mode is active on the Device. For example, similar to the mechanism employed in the TCG PC Client Platform Firmware Profile Specification, a Device could use the EXTEND operation defined in Section 3.2.1.1 to extend the Measurement with a generic string "DEBUG: <Debug Mode Enabled>" (string between brackets can be product-specific). However, a general reporting mechanism is out-of-scope for this specification.

3.3 PCIe Device Measurement Verification

The PCIe Device Measurements collected by the Host may be verified against a database containing the whitelist measurement hashes for all PCIe devices. The Host itself or a remote service may be pre-populated with such a database in a trusted manner. The pre-population mechanism itself is considered out-of-scope of this specification.

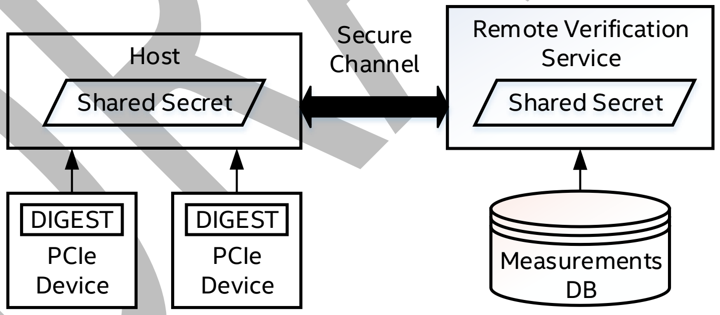


Figure 2: PCIe Device Measurement Verification.

If a remote service is used for the expected measurement database (DB) as illustrated in Figure 2, a secure channel based on shared secrets, shall be established for transfer of measurements from the Host to the remote service. The security requirements for a remote service are out-of-scope for this specification.

4 PCIe Device Authentication Architecture

Having established the mechanism to query the Device's firmware identity in Section 3, this section describes the PCIe Device Authentication architecture to provide cryptographic guarantees to the Device's hardware and firmware identities as well as its capabilities. The underlying architecture, protocol and messages are adapted from the USB Type-C Authentication Specification. Therefore, this section focuses on the PCIe interface that is unique to PCIe Device Authentication, as well as pointing out the differences to the USB Type-C Authentication Specification. We also state the architectural scope and assumptions made by this section and leave any implementation-specific considerations and requirements to Section 0.

PCIe Device Authentication defines an optional Extended Capability structure, to provide software and/or firmware, running on the Host, a way to query the cryptographic identity of a Device, such that the unique identity of the Device as well as the Device's capability can be determined in a cryptographically secure manner, allowing software and/or firmware to make a trust decision (examples given in Section 1).

4.1 Architectural Overview

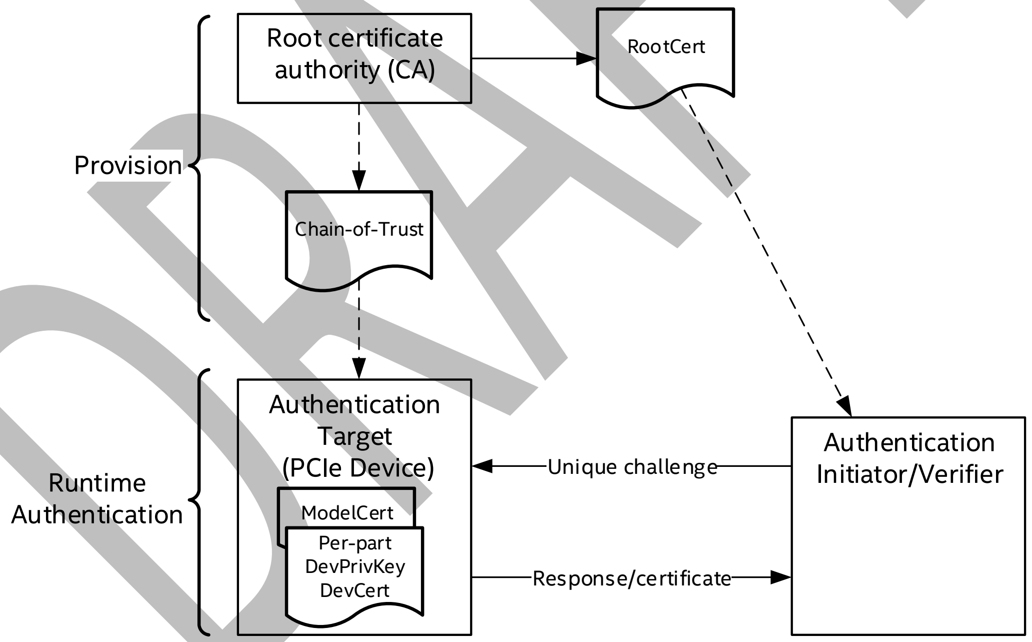

Root certificate authority (CA)
RootCert
Provision
Chain-of-Trust
Runtime Authentication
Authentication Target (PCIe Device)
ModelCert
Per-part DevPrivKey DevCert
Unique challenge
Response/certificate
Authentication Initiator/Verifier

Figure 3: High-level Device Authentication architecture. The architecture includes Provisioning and Runtime Authentication.

Figure 3 depicts the high-level architecture for the PCIe Device Authentication, including the Authentication Provisioning and the Runtime Authentication.

Authentication Provisioning is a process followed by the Device vendor as part of the manufacturing flow, and may be extended through additional provisioning performed after Device manufacturing. A trusted root certificate authority (CA) generates a root certificate (*RootCert*) that is provisioned to the Authentication Initiator/Verifier to allow the Authentication Initiator to verify the validity of the signatures generated by the Device during Runtime

Authentication. The root CA also indirectly endorses (through a trust hierarchy, Section 4.4) a per-part Device public/private key pair, where the Device private key is provisioned to the Device or generated by the Device (Section 5.1.4.1) and the Device public key is contained in the Device Certificate (*DeviceCert*) that is signed using a private key that can be verified using the root CA's public key in the *RootCert*.

Runtime Authentication is the process by which host software interacts with the Device in a running system. After the Device public/private key and *RootCert* provisioning, the Authentication Initiator can retrieve the certificate(s) from the Device, send a unique challenge, in the form of a nonce---number used once---into the Device and the Device can authenticate its identity and capability by signing the challenge along with other Authentication Data with the Device private key. The Authentication Initiator verifies the signature using the public keys of the Device and the root CA, as well as any intermediate public keys.

4.2 Architectural Assumptions

The Device Authentication architecture makes the following assumptions:

- The Device provides adequate protections for the per-part Device private key in-use (secure signing) and at-rest (secure storage), where the per-part Device private key is the Device Root of Trust (RoT). See Section 5.1 for the Device RoT protection requirements in detail.
- The Device implements the measurement of its first mutable firmware using only hardware or immutable firmware that is stored in Read-only Memory (ROM). This is the Device Root of Trust for Measurement (RTM). For details on the Device Measurement, refer to Section 3.
- The Device Root of Trust for Reporting (RTR) is the combination of the Device RoT and the Device RTM in this specification.
- The Authentication Initiator has the genuine root certificate (*RootCert*), where the RoT for Device Authentication is the root CA's private key.

4.3 Threat Model

In addition to following the threat model employed by PCIe Device Measurement in Section 3.1, PCIe Device Authentication considers any physical extraction of the Device private key in-scope for the threat model. In other words, Device manufacturers shall provide physical protection mechanisms for the Device RoT. For different protection levels defined in this specification, please refer to Section 5.1.4. Furthermore, for the Device private key, Device manufacturers shall employ adequate protections against malicious insider attacks where adversary can gain access to the Device private key generation and provisioning process to compromise the PCIe Device Authentication architecture.

4.4 Trust Hierarchy

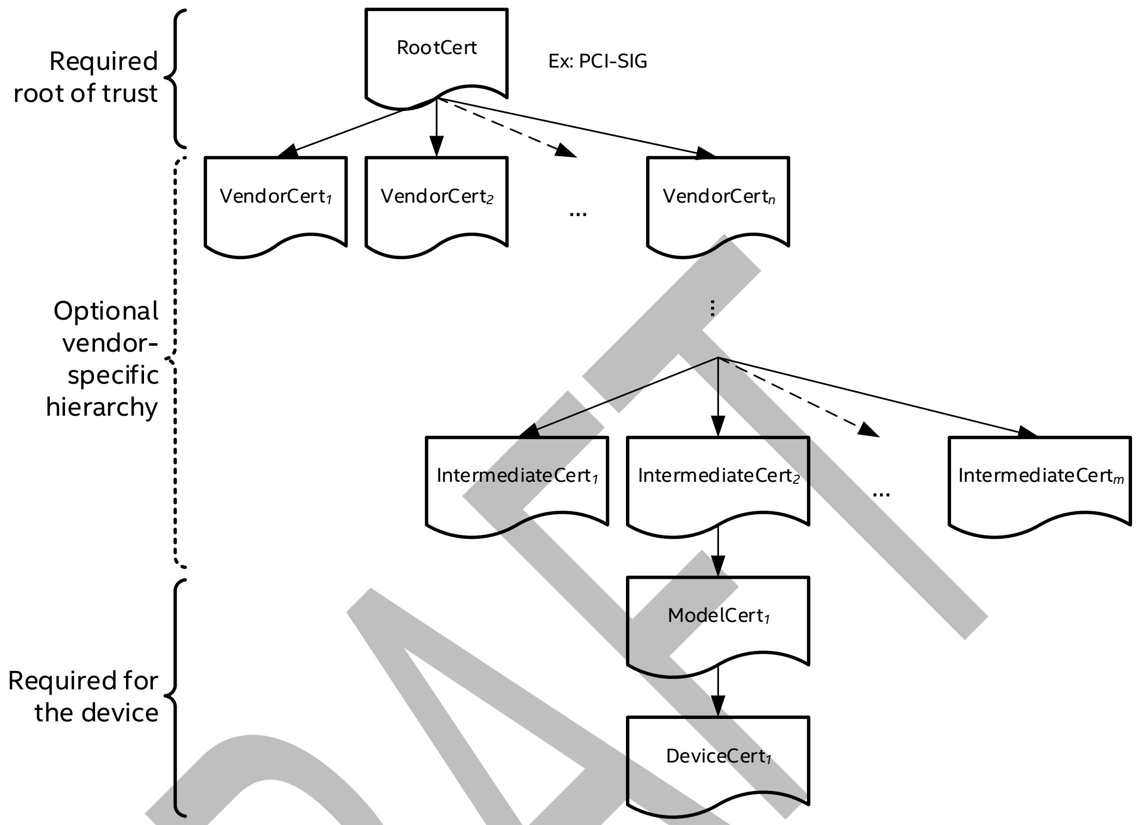


Figure 4: Trust hierarchy for the PCIe Device Authentication.

Figure 4 depicts the trust hierarchy for the PCIe Device Authentication, where a chain of trust is established through the signing of certificates from the *DeviceCert* all the way up to the *RootCert*. The entire certificate chain is required for the Authentication Initiator when verifying the signature generated by the Device. At the minimum, the *RootCert*, *ModelCert* and the *DeviceCert* are required to form a minimal certificate chain, while the intermediate certificates form an expanded chain-of-trust. Requiring a unique per-part *DeviceCert* allows the Device Authentication Host software to detect advanced Device-cloning attacks, as well as to revoke an individual Device when necessary. A PCIe vendor can choose to have an arbitrary number of intermediate certificates, as long as the total length of the certificate chain is within the maximum length defined (See Section 3 in the USB Type-C Authentication specification).

4.5 PCIe Configuration Space Authentication Mechanism Register Interface

This section defines a Designated Vendor-Specific Extended Capability (DVSEC) that provides a register interface that can be used by Host software to retrieve the identity and Device capability via cryptographically secure messages.

If the Authentication Extended Capability is supported on a Device, the Device must implement the Capability on Function 0 of the Device, and is permitted to implement the capability on some or all other Physical Functions if needed.

Table 3 details allocation of register fields in the Authentication Extended Capability structure.

Table 3: Device Authentication Designated Vendor-Specific Extended Capability

31 ... 0	
PCI Express DVSEC Header 1	00h
PCI Express DVSEC Header 2	04h
PCI Express DVSEC Header 3	08h
Authentication Header	0Ch
Authentication Capabilities	10h
Authentication Status	14h
Authentication Control	18h
Write Data Mailbox	1Ch
Read Data Mailbox	20h

4.5.1 PCI Express DVSEC Header 1 (Offset 00h)

Figure 5 details allocation of register fields in the PCI Express DVSEC Header 1; Table 4 provides the respective bit definitions.

31 ... 20	19 ... 16	15 ... 0
Next Capability Offset	Capability Version	PCI Express Extended Capability ID

Figure 5: PCIe DVSEC header 1.

Table 4: PCIe DVSEC header 1.

Bit Location	Register Description	Attributes
15:0	PCI Express Extended Capability ID— Must be 023h (DVSEC).	RO

19:16	Capability Version— Must be 1h for this version of the specification.	RO
31:20	Next Capability Offset— See PCIe Base Specification for definition.	RO

4.5.2 PCI Express DVSEC Header 2 (Offset 04h)

Figure 6 details allocation of register fields in the PCI Express DVSEC Header 2; Table 5 provides the respective bit definitions.

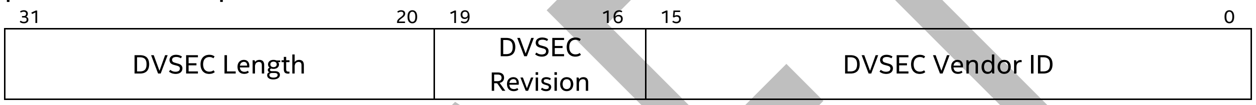


Figure 6: PCIe DVSEC header 2.

Table 5: PCIe DVSEC header 2.

Bit Location	Register Description	Attributes
15:0	DVSEC Vendor ID--- This field is the Vendor ID associated with the vendor that defined the contents of this capability. Must report Intel vendor ID (8086h). Assigned by PCI SIG.	RO
19:16	DVSEC Revision---This field is a vendor-defined version number that indicates the version of the DVSEC structure. Implementations that comply with this version of the specification must report 1. Defined by this specification.	RO

31:20	DVSEC Length---This field indicates the number of bytes in the entire DVSEC structure. Implementations that comply with this version of the specification must return 40.	RO

4.5.3 PCI Express DVSEC Header 3 (Offset 08h)

Figure 7 details allocation of register fields in the PCI Express DVSEC header 3; Table 6 provides the respective bit definitions.

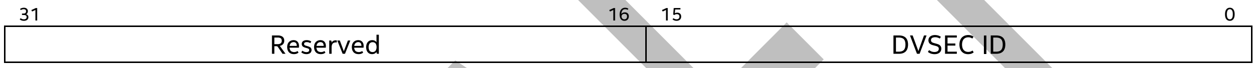


Figure 7: PCIe DVSEC header 3.

Table 6: PCIe DVSEC header 3.

Bit Location	Register Description	Attributes
15:0	DVSEC ID---This field is a vendor-defined ID that indicates the nature and format of the DVSEC structure. Implementations that comply with this version of the specification must return 02Eh. This DVSEC ID is assigned by Intel corporation.	RO
31:16	Reserved	RsvdZ

4.5.4 Authentication Header (Offset 0Ch)

Figure 8 details allocation of register fields in the Authentication Header; Table 7 provides the respective bit definitions.

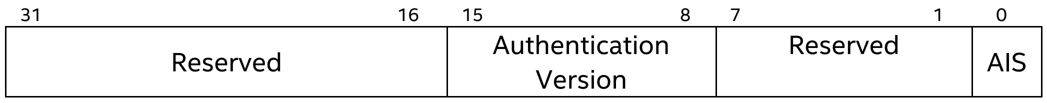


Figure 8: Authentication Header. AIS: Authentication Interrupt Support.

Table 7: Authentication Header.

Bit Location	Register Description	Attributes
0	Authentication Interrupt Support—When Set, this bit indicates Device support for using MSI/MSI-X to indicate completion of an authentication message by the Device.	RO
7:1	Reserved	RsvdZ
15:8	Authentication Version—This field indicates the highest version of Authentication supported by the Device. 00h: Reserved 01h: Device Authentication Version 1.0 02h-FFh: Reserved	RO
31:16	Reserved	RsvdZ

4.5.5 Authentication Capabilities (Offset 10h)

Figure 9 details allocation of register fields in the Authentication Capabilities; Table 8 provides the respective bit definitions.

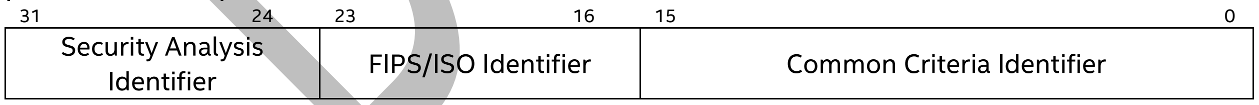


Figure 9: Authentication Capabilities.

Table 8: Authentication Capabilities.

Bit Location	Register Description	Attributes
15:0	Common Criteria Identifier—As defined in USB 3.1 Type C Authentication.	RO

23:16	FIPS/ISO Identifier—As defined in USB 3.1 Type C Authentication.	RO
31:24	Security Analysis Identifier—As defined in USB 3.1 Type C Authentication.	RO

4.5.6 Authentication Status (Offset 14h)

Figure 10 details allocation of register fields in the Authentication Status; Table 9 provides the respective bit definitions.

31	30 … 6	5 … 1	0
Response Ready	Reserved	IMN	B

Figure 10: Authentication Status. B: Busy. IMN: Interrupt Message Number.

Table 9: Authentication Status.

Bit Location	Register Description	Attributes
0	Busy—When Set, this bit indicates the Device is unable to start an authentication session. See the Abort and Go bit in Authentication Control for usages of the Busy bit.	RO

5:1	Interrupt Message Number (IMN) —This field indicates which MSI/MSI-X vector is used for the Valid interrupt message. For MSI this field indicates the offset between the base Message Data and the interrupt message that is generated. Hardware must update this field when software Sets the Multiple Message Enable field in the MSI Message Control register if the number of MSI Messages assigned to the Function is less than the value of this register. For MSI-X this field indicates the index of the MSI-X Table entry is used to generate the interrupt message. For a given MSI-X implementation, the entry must remain constant. If both MSI and MSI-X are implemented, they are permitted to use different vectors, though software is permitted to enable only one mechanism at a time. When MSI-X is enabled this register indicates the vector for MSI-X. If MSI-X is disabled this register indicates the vector for MSI. If software enables both MSI and MSI-X at the same time, the value in this register is undefined.	RO
30:6	Reserved	RsvdZ
31	Response Ready—When Set, this bit indicates the Device has a valid message to be read by the Host. The Device must clear this bit only after the entire message has been consumed.	RO

4.5.7 Authentication Control (Offset 18h)

Figure 11 details allocation of register fields in the Authentication Control; Table 10 provides the respective bit definitions.

31	30 … 2	1	0
Go	Reserved	IE	A

Figure 11: Authentication Control. A: Abort. AIE: Interrupt Enable. IMN: Interrupt Message Number.

Table 10: Authentication Control.

Bit Location	Register Description	Attributes

0	Abort—When set, the current message sent by the Host is aborted by the Device. The Device asserts the Busy bit in Authentication Status when processing the Abort and clears the Busy bit when the Abort is completed. Reads from this bit must always return 0b.	RW
1	Interrupt Enable—When set and MSI/MSI-X is enabled, the Device must issue an MSI/MSI-X interrupt to indicate the 0b to 1b transition of the Response Ready bit. Default value of this bit is 0b.	RW
30:2	Reserved	RsvdZ
31	Go—A write of 1b to this bit indicates to the Device that the Device can start consuming the message sent in through the Write Data Mailbox. The Device asserts the Busy bit in Authentication Status when processing the message and clears the Busy bit when the entire message is consumed.	WO

4.5.8 Write Data Mailbox (Offset 1Ch)

Table 11 provides the definition.

Table 11: Write Data Mailbox

Bit Location	Register Description	Attributes
31:0	Write Data Mailbox—The Device received data from consecutive 32-bit data writes to this register. Reads of this register must return all 0's.	RW

4.5.9 Read Data Mailbox (Offset 20h)

Table 12 provides the definition.

Table 12: Read Data Mailbox.

Bit Location	Register Description	Attributes

31:0	Read Data Mailbox—Response Ready is asserted when the FIFO is populated with a complete response message.	RO

4.6 Authentication Over Management Component Transport Protocol (MCTP)

This section defines how PCIe Device Authentication messages can be exchanged over the Management Component Transport Protocol (MCTP), in addition to exchanging the messages over the PCIe Configuration Space registers defined in Section 4.5. Allowing the authentication messages over the MCTP provides several benefits. MCTP is defined over several different physical mediums, e.g., SMBus/I^{2}C, serial links and PCIe; therefore, the Device Authentication mechanism can easily be extended to accommodate devices not connected through PCIe, or when a PCIe device is not powered by the main interface in inventory or during supply-chain transportation. The definition of "Host" is greatly expanded in the Host-Device relationship shown in Figure 1, as the "Host" becomes any MCTP endpoint; this allows the Device Authentication mechanism to scale to peer-to-peer scenarios.

Specifically, we describe how the Authentication Messages defined in Section 4.9 are encapsulated in the Intel Vendor Defined Messages that are exchanged between MCTP endpoints.

4.6.1 Intel Vendor Defined Message for Device Authentication

Intel Vendor Defined Messages for Device Authentication utilize MCTP Message Type of "Vendor Defined – PCI", Message Type Code 0x7E. Please refer to the MCTP Base Specification [DSP0236] for details (See Section 1.1 for a link to the MCTP Base Specification). Per MCTP Base Specification, the *Get Vendor Defined Message Support* command enables endpoints to discover whether the Target Endpoint supports vendor-defined messages, and, if so, the vendors or organizations that defined those messages. All endpoints that implement Intel Vendor Defined Messages for Device Authentication shall respond to *Get Vendor Defined Message Support* and return data as shown in Table 13.

Table 13: *Get Vendor Defined Message Support* response packet. IC: Integrity Check bit.

+0								+1								+2								+3							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
IC	Msg Type = 0							Rsvd			Instance ID					Command Code = 0x6								Completion Code = 0							
Vendor ID Set Selector = 0xFF								Vendor ID Format = 0x0								PCI Vendor ID[1] = 0x80								PCI Vendor ID[2] = 0x86							
Command Set Type = 0x100																															

The Vendor ID Set Selector shall have a value of 0xFF if the endpoint does not support any other vendor defined command sets (besides the one listed in the response). The Vendor ID Format shall have a value of 0x0 to indicate that the format used to identify the vendor is the

PCI Vendor ID format. The PCI Vendor ID field shall have a value 0x8086 to indicate that Intel Corporation defined the Vendor Defined Message (VDM). The Command Set Type shall have a value of 0x100 indicating it follows Sections 4.6.1 and Section 4.6.2 in this specification. The *Get Vendor Defined Message Support* command is an MCTP control command and thus the response packet shall set IC=0.

4.6.2 Device Authentication Command Set

Intel Vendor Defined Messages for Device Authentication can be referenced via its unique 1-byte handle, 0x07.

4.6.2.1 Completion Codes

For an endpoint supporting Device Authentication over MCTP, the endpoint shall send an Authentication Response for each Authentication Request that requires a response. For all error conditions encountered by the endpoint, the endpoint shall send the Authentication Error Response message as defined in Section 4.9. Therefore, for all MCTP Device Authentication command packets, no specific completion code is required.

4.6.2.2 Discovery of Device Authentication Command Set Support

Prior to issuing any Device Authentication commands, the source endpoint must ensure the endpoint supports the command set by using the following algorithm.

1. Issue *Get Message Type* support to ensure Message Type 0x7E is supported.
2. Issue *Get Vendor Defined Message Support* command (Section 4.6.1) and make sure the destination endpoint supports Intel Proprietary command set. If the endpoint does not support Intel (0x8086) vendor defined commands, do not issue any Authentication Request command.

4.6.2.3 Device Authentication Command Format

An example Device Authentication Request Command packet layout is shown in Table 14, where the shaded fields in Table 14 are described in detail in Table 15. A Device Authentication Response packet follows a similar format, except for the Completion Code field as well as differences defined in the MCTP Transport Header, which includes the MCTP Header Version, Destination Endpoint ID and other fields as defined by MCTP.

Table 14: Device Authentication Request Command packet layout.

+0								+1								+2								+3							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Physical transport header. For details on the PCIe VDM and SMBus transport header definitions, please refer to the MCTP PCIe VDM Transport Binding Specification and the MCTP SMBus/I2C Transport Binding Specification (Section 1.1), respectively.																															
MCTP Rsvd				Header Version				Destination Endpoint ID								Source Endpoint ID								SOM	EOM	Pkt Seq #		TO	Msg Tag		
Msg Type = 0x7E								MCTP PCI Vendor Defined Vendor ID = 0x8086																Rq	D	C	Sequence Number				

Vendor Defined Handle = 0x07	Authentication Request Message as defined in Section 4.9.
Continuation of the Authentication Request Message as defined in Section 4.9.	

Table 15: Device Authentication Command format.

	Byte	Description
Authentication Request	1:2	0x8086, declares Intel as the vendor. MSB first.
	3	Bit 7: Rq=1b. Bit 6: D. Used to indicate whether the Sequence Number is used for tracking and matching requests and responses. Bit 5: C (Crypt) . Used to indicate whether or not the message body is encrypted, including the Sequence Number. Bits [4:0] : Sequence Number Refer to the definition of MCTP Control Message Fields MCTP Base Specification.
	4	Handle. Set to 0x07.
	(5:N)	Additional Command-specific Request Bytes. Byte 5 carries the opcode. The format is specified for each Authentication Request Message defined in Section 4.9, including the Authentication Request Header and the Authentication Request Payload, if there is any. Includes padding with zeroes to ensure packet size is a multiple of DWORD, if required by the physical transport medium.
Authentication Response	1:2	0x8086, declares Intel as the vendor. MSB first.
	3	Bit 7: Rq=0b. Bit 6: D Bit 5: C (Crypt) Bits [4:0] : Sequence Number Refer to the definition of MCTP Control Message Fields MCTP Base Specification.
	4	Handle. Set to 0x07.
	5	Completion code, see Section 4.6.2.1 for definitions.
	(6:M)	Additional Command-specific Response Bytes. The format is specified for each Authentication Response Message defined in Section 4.9, including the Authentication Response Header and the Authentication Response Payload, if there is any. Includes padding with zeroes to ensure packet size is a multiple of DWORD, if required by the physical transport medium.

4.6.3 Implementation Requirements for MCTP

This section lists the requirements for implementing Intel Vendor Defined Message for Device Authentication over MCTP.

- When the requestor receives a response with completion code ERR_NOT_READY, it indicates the responder is busy and not ready to respond to an Authentication Request. This behavior is different from the host software polling the "Busy" bit in the Authentication Status register (Table 9).
- A requestor shall ensure in-order behavior; in other words, a requestor cannot issue a new request before response from the last request is received. The only exception to this requirement is when the requestor issues the abort command. An endpoint shall return "ERR_NOT_READY" when this rule is violated.

4.6.4 Timing Requirements for MCTP

For the request response timing and time-out requirements, please refer to "Table 9 – Timing Specifications for MCTP control messages on SMBus" in the MCTP SMBus/I2C Transport Binding Specification (Section 1.1).

4.7 Certificates, Certificate Chains and Device Private Keys

A PCIe Device shall contain a certificate chain and implement the Device private key, as defined in Section 3.2 of the USB Type-C Authentication Specification. For details regarding the format of certificates, the certificate chains and the Device private keys, please refer to Section 3 Authentication Architecture of the USB Type-C Authentication Specification.

4.8 Authentication Protocol

A PCIe Device shall receive Authentication Messages and reply to Authentication Messages using the PCIe Configuration Space Authentication Mechanism Registers defined in Section 4.5 or SMBus Authentication Mechanism Registers defined in Section 4.6. For example, in response to a CHALLENGE Request (see USB Type-C Authentication Specification, Section 5.2.3), a PCIe Device shall return a CHALLENGE_AUTH Response (see USB Type-C Authentication Specification, Section 5.3.3) with the modifications defined in Section 4.10. For details regarding the certificate digest query, reading of the certificate chain, the challenge-response protocol and the error conditions, please refer to Section 4 Authentication Protocol of the USB Type-C Authentication Specification.

4.9 Authentication Messages

For details regarding the authentication message header and payload format and definitions, authentication request and response message types, please refer to Section 5 Authentication Messages of the USB Type-C Authentication Specification. For any modification with respect to PCIe, please refer to Section 4.10.

4.10 PCIe Adaptations of USB Type-C Authentication

This section describes in detail all the PCIe adaptations of and differences with the USB Type-C Authentication Specification.

4.10.1 Cryptographic Algorithms

PCIe Device Authentication requires a minimum level of 192-bit security. Table 16 lists the cryptographic algorithm usage differences between PCIe Authentication and USB Type-C Authentication.

Table 16: Cryptographic algorithm differences between PCIe Authentication and USB Type-C Authentication.

Cryptographic Algorithm	**PCIe Authentication**	**USB Type-C Authentication**
Digital Signature	EC DSA NIST P-384 or RSA 3072	EC DSA NIST P-256
Hash Algorithm	SHA2-384 or SHA2-512 SHA3-384 or SHA3-512	SHA-256

4.10.2 Authentication Messages

In addition to the Authentication Messages defined by the USB Type-C Authentication specification, PCIe Authentication expands the message definitions to include extra Authentication Request and Authentication Response message types. Table 17 lists the Authentication Request message types for PCIe Authentication and Table 18 lists the Authentication Response message types for PCIe Authentication.

Table 17: PCIe Authentication Request message types. Gray rows highlight the differences from the USB Type-C Authentication specification.

Value	**Description**
00h – 7Fh	Shall only be used for Authentication Responses
80h	Reserved
81h	GET_DIGESTS
82h	GET_CERTIFICATE
83h	CHALLENGE
84h – DFh	Reserved
E0h	GET_MEASUREMENT
E1h	GET_CAPABILITY
E2h	SET_CERTIFICATE
E3h – FFh	Reserved

Table 18: PCIe Authentication Response message types. Gray rows highlight the differences from the USB Type-C Authentication specification.

Value	**Description**
00h	Reserved

01h	DIGESTS
02h	CERTIFICATE
03h	CHALLENGE_AUTH
04h – 5Fh	Reserved
60h	MEASUREMENT
61h	CAPABILITY
62h – 7Eh	Reserved
7Fh	ERROR
80h – FFh	Shall only be used for Authentication Requests

4.10.2.1 GET_MEASUREMENT Authentication Request Message

PCIe Authentication expands the USB Type-C Authentication Request message type definitions to include a GET_MEASUREMENT request message for the host software to request for all of the measurement values reported by the Device, in accordance to the definitions given in Section 3. The header for a GET_MEASUREMENT Request is defined in Table 19. The payload for the GET_MEASUREMENT Request is defined in Table 20.

Table 19: GET_MEASUREMENT Request Header.

Offset	Field	Size	Value
0	*ProtocolVersion*	1	V1.0
1	*MessageType*	1	GET_MEASUREMENT
2	*Param1*	1	Reserved
3	*Param2*	1	Reserved

Table 20: GET_MEASUREMENT Request Payload

Offset	Field	Size	Value
4	*Reserved*	2	Reserved to be compatible with the Cerberus definition.
6	*Nonce*	32	Random 32-byte nonce chosen by the Authentication Initiator. This field is little endian.

4.10.2.2 MEASUREMENT Authentication Response Message

PCIe Authentication expands the USB Type-C Authentication Response message type definitions to include a MEASUREMENT response message for the host software to receive all of the measurement values reported by the Device, in accordance to the definitions given in Section 3. The header for a MEASUREMENT Response is defined in Table 21. The payload for a MEASUREMENT Response is defined in Table 22.

Table 21: MEASUREMENT Response Header

Offset	Field	Size	Value
0	*ProtocolVersion*	1	V1.0
1	*MessageType*	1	MEASUREMENT

2	*Param1*	1	Reserved
3	*Param2*	1	Reserved

Table 22: MEASUREMENT Response Payload

Offset	Field	Size	Value
4	*Length*	2	Length in bytes
6	*NumberofMeasurements (N)*	1	Number of Measurement hashes
7	*MeasurementLength (L)*	1	Length in bytes for each Measurement hash
8	*Measurements*	*L*N*	Concatenation of all Measurement hashes
8 + (*L*N*)	*Signature*	*	Signature of the GET_MEASUREMENT Request and MEASUREMENT Response messages, excluding the Signature field and signed using the Device Private Key. The size of the Signature field depends on the asymmetric signing algorithm used.

4.10.2.3 GET_CAPABILITY Authentication Request Message

PCIe Authentication expands the USB Type-C Authentication Request message type definitions to include a GET_CAPABILITY request message for the host software to request for the supported cryptographic algorithms by the Device. The header for a GET_CAPABILITY Request is defined in Table 23. A GET_CAPABILITY Request has no payload.

Table 23: GET_CAPABILITY Request Header.

Offset	Field	Size	Value
0	*ProtocolVersion*	1	V1.0
1	*MessageType*	1	GET_CAPABILITY
2	*Param1*	1	Reserved
3	*Param2*	1	Reserved

4.10.2.4 CAPABILITY Authentication Response Message

PCIe Authentication expands the USB Type-C Authentication Response message type definitions to include a CAPABILITY response message for the host software to receive the supported cryptographic algorithms supported by the Device. The header for a CAPABILITY Response is defined in Table 24. The payload for a CAPABILITY Response is defined in Table 25.

Table 24: CAPABILITY Response Header.

Offset	Field	Size	Value
0	*ProtocolVersion*	1	V1.0
1	*MessageType*	1	CAPABILITY
2	*Param1*	1	Reserved
3	*Param2*	1	Reserved

Table 25: CAPABILITY Response Payload

Offset	Field	Size	Value
4	*MaxPayloadSize*	2	Maximum payload size
6	*Reserved*	2	Reserved to be compatible with the Cerberus definition.
9	*AsymmetricKeyLength*	1	Asymmetric key strength: [7] RSA [6] ECDSA [5:3] ECC 000: 256-bit 001: 384-bit 010: 512-bit 100: Reserved [2:0] RSA 000: None 001: RSA 2048 010: RSA 3072 100: RSA 4096
10	*SymmetricKeyLength*	1	Symmetric key strength: [7] Reserved [6] Reserved [5:3] AES 000: None 001: 128-bit 010: 256-bit 100: 384-bit
11	*HashLength*	1	Hash strength: [7:6] SHA2 00: None 01: 384-bit 10: 512-bit [5:4] SHA3 00: None 01: 384-bit 10: 512-bit [3:0] Reserved

4.10.2.5 SET_CERTIFICATE Authentication Request Message

PCIe Authentication expand the USB Type-C Authentication Request message type definitions to include a SET_CERTIFICATE request message for the host software to provision new certificate chains onto the Device. Note that the SET_CERTIFICATE Request only allows updating slot 1 through 7 of the 8 certificate chain slots on the Device. Slot 0 shall only be updatable in a manufacturing environment where the Device vendor provisions or updates the slot 0 of the certificate chains. The header for a SET_CERTIFICATE Request is defined in Table 26. The payload for a SET_CERTIFICATE Request is one entire certificate chain defined in the USB Type-C Authentication specification. There is no partial update to a certificate chain allowed.

Table 26: SET_CERTIFICATE Request Header

Offset	Field	Size	Value
0	*ProtocolVersion*	1	V1.0
1	*MessageType*	1	SET_CERTIFICATE
2	*Param1*	1	Slot number of the target Certificate Chain to read from. The value in this field shall be between 1 and 7 inclusive. If 0 is used in this parameter, the Device shall return an Error Response message.
3	*Param2*	1	Reserved

4.10.3 Sending and Receiving of Authentication Messages

The sending and receiving of the Authentication Messages across the PCIe link are performed through the mailboxes defined in Sections 4.5.8 and 4.5.9. The Authentication Messages defined in Section 4.9 shall be broken down into 32-bit chunks to be repeatedly populated into the mailboxes, starting from byte 0 of the message in little-endian manner, until the last byte of the message is populated in the mailboxes. The Device should implement adequate buffer and storage to account for the execution speed difference between the Host and the Device to ensure proper sending and receiving of the entire Authentication Messages.

An example software flow is given below for illustration purposes.

1. Host discovers Device support for authentication on extended capability registers and configures the authentication algorithms used.
2. Host checks the status register.
3. Host writes message into mailbox and asserts the “Go” bit.
4. Device consumes the message from mailbox.
5. Device generates a response message and asserts “Response Ready”.
6. Host polls on the “Response Ready” bit and reads data from mailbox.
7. Repeat steps 1-6 if needed.

4.10.4 Timing Requirements for Message Exchanges on PCIe

A Device shall complete processing the Authentication Request message, prepare the Authentication Response message and assert Response Ready in the Authentication Status register within one second after the Go bit is asserted in the Authentication Control register.

4.10.5 Context Hash for CHALLENGE_AUTH Response Messages

The Context Hash field in the CHALLENGE_AUTH Response Message is used for the Device to sign over Device-specific information that is included in the Authentication. This allows the verifier to ensure the validity and authenticity of any Device-specific information. To expand the usages of Authentication over different physical mediums and types of Devices, an “Organizational Namespace” is defined and used to distinguish between different CONTEXT_HASH definitions. For example, the USB Type-C Authentication specification defines two CONTEXT_HASH definition, one for the PD and one for the USB device; whereas PCIe Authentication defines PCIe-specific CONTEXT_HASH definitions.

To support the Organizational Namespace, the 7th byte (a reserved byte in the USB Type-C Authentication specification version 1.0) of the CHALLENGE_AUTH Response Payload is used. For completeness, Table 27 shows the CHALLENGE_AUTH Response Payload, including the definition of the Organizational Namespace. A Device manufacture shall only use one Organizational Namespace value for a Device supporting multiple interfaces defined by different organizations. For example, a PCIe Device supporting both the PCIe interface as well as an MCTP endpoint over SMBus interface shall return only one value for both the Organizational Namespace and the CONTEXT_HASH, regardless of which interface is used to Authenticate the Device. In other words, a Device manufacture shall determine the primary Organizational Namespace a Device belongs to, irrespective of the total number of other Organizational Namespaces that a Device could be assigned to. The only exception is for the existing definitions given in USB Type-C Authentication, where two Organizational Namespace values could be used.

Table 27: CHALLENGE_AUTH Response Payload

Offset	Field	Size	Value
4	*MinProtocolVersion*	1	Minimum protocol version supported by this Device
5	*MaxProtocolVersion*	1	Maximum protocol version supported by this Device
6	*Capabilities*	1	Set to 01h for this specification. All other values reserved
7	*OrganizationalNamespace*	1	00-07h: USB-IF 08h: PCI-SIG 09h: JEDEC 0Ah: DMTF 0Bh: MIPI All other values reserved.
8	*CertChainHash*	32	32-byte SHA256 hash of the Certificate Chain used for Authentication. This field is big endian.
40	*Salt*	32	32-byte value chosen by the Authentication Responder. This field is little endian. Note: the Salt can be random, fixed, or any other value.
72	*Context Hash*	32	See the USB Type-C Authentication specification for USB definitions. See the rest of this section for PCIe definitions.

104	*Signature*	Length of signature	The length of this field depends on the signing algorithms used. This field is little endian. Signed using the Device Private Key.

For PCIe Authentication, the Context Hash field captures the hash value calculated over the concatenation of two data structures, DEV_IDENTITY and FW_IDENTITY, shown in Table 28 and Table 29, respectively. The DEV_IDENTITY is a 3-DOWRD data structure, defined to capture data PCIe-specific information for a Device. The FW_IDENTITY includes a list of all the Device Measurement values, as well as their respective versions and Firmware IDs. Calculating the hash value of this data structure and signing over this data structure using the Device private key allow these measurement values to be verified as genuine according to the Device vendor's specification, as opposed to blindly trusting the values reported by the PCIe Device.

Table 28: DEV_IDENTITY data structure for the Context Hash field in the CHALLENGE_AUTH Message for PCIe.

31 16	15 8	7 0
Device/Vendor ID		
Class Code/Revision ID		
Subsystem ID/Vendor ID		

Table 29: FW_IDENTITY data structure for the Context Hash field in the CHALLENGE_AUTH Message. Multiple FW_IDENTITY structures are concatenated if NUM_DIGEST is greater than zero. The DIGEST field referred to in this table is the DIGEST register field defined in Section 3.2.1.

31 16	15 5	4 0
Firmware Version	Reserved	Firmware ID
DIGEST (bits 0 through 31)		
DIGEST (bits 32 through 63)		
... (Total DIGEST length determined by the hash algorithm used.)		

5 PCIe Device Authentication Implementation Requirements and Considerations

This section describes the implementation-specific requirements and considerations to achieve a full PCI Express Authentication solution. Specifically, this document describes the following aspects---1) Roots of Trust and their protections, 2) certification of trust-worthy Devices, 3) certificate revocation, and 4) key and certificate generation, distribution, provisioning and recovery. This document does not cover any policy decision that the Authentication Verifier makes based on the authentication result. This section discusses several practical implementation considerations with respect to implementing the Device Authentication support for both the Device and the Host software.

The PCI Express Authentication (Section 4) defines a mechanism for an Authentication Verifier to query the identity and capability of a PCI Express Device, where the Device signs the authentication report with a unique Device private key. Along with the Device certificate that contains the trusted public key that can be used to verify the validity of the signature, the Authentication Verifier can decide that only Devices with the correct identities and capabilities are connected to the computing platform. In other words, Section 4 describes *how* the Authentication Verifier performs the authentication and *what* data the Device and the Verifier exchange during the authentication. However, Section 4 *does not* describe *why* the Verifier should trust the Device certificate or the signature generated by the Device. For example, an attacker could extract the Device private key from a genuine Device and use the private key to sign the authentication for a counterfeit Device.

This section details the requirements for the following aspects of the authentication that are required to allow the Verifier to determine *why* the authentication result should be trusted. Specifically, these aspects are:

- Roots-of-Trust protection requirements
- Certification of a trust-worthy Device
- Certificate revocation
- Key and certificate generation, distribution, provisioning and recovery

5.1 Roots-of-Trust Protection Requirements

This section describes the protection requirements for the various Roots-of-Trust that are essential to the overall security of the Device Authentication solution.

Each private key of the public/private key pair in the certificate chain is considered a Root-of-Trust (RoT), as compromising each private key in the chain would break the security of the overall authentication solution. We describe the protection requirements for each private key in this section.

5.1.1 Root Certificate Authority RoT Protection

The root Certificate Authority's (CA) private key is of the utmost importance of the entire solution, as the CA is at the root of the trust chain. If an attacker can retrieve the root CA's private key, he/she can create an arbitrary trust hierarchy with an arbitrary amount of Devices with legitimate certificates, essentially breaking the entire authentication scheme. Therefore, it is the root CA's responsibility to ensure that the root CA's private key is adequately protected and only used when needed to minimize exposure. The protection adequacy needs to be evaluated on a per-implementation basis and is not elaborated in this specification.

5.1.2 Intermediate RoT Protections

In the trust hierarchy of the Device Authentication, there can be an arbitrary number of intermediate certificates between the root certificate and the model/Device certificates. Each of these intermediate certificates contains the public key portion of the public/private key pair, where the certificate is signed using the private key from the private key of the upper level in the chain. Therefore, a compromised intermediate private key in any level of the chain would lead to generation of trusted lower-level certificates that can be controlled by attackers. Since the number and implementation of the intermediate certificates are determined by a particular PCI Express Device vendor, it is the vendor's responsibility to ensure that the intermediate private keys are adequately protected and only used when needed to minimize exposure. The protection adequacy needs to be evaluated on a per-implementation basis and is not elaborated in this specification.

5.1.3 Model RoT Protection

Since the Model Certificate's private key is not stored on the Device, it is the vendor's responsibility to ensure that the model private keys are adequately protected and only used when needed to minimize exposure. The protection adequacy needs to be evaluated on a per-implementation basis and is not elaborated in this specification.

5.1.4 Device RoT Protection

The Device RoT, the Device private key of the Device public/private key pair, uniquely identifies a particular Device and is stored and used on the Device. Therefore, the Device private key shall be adequately protected to prevent leakage of the Device private key when the Device private key is stored (at-rest) or when in-use. We define the protection adequacy and levels of the Device RoT in this section.

5.1.4.1 Device RoT Protection Levels

There are different levels of Device RoT protections that a Device can implement, depending on the target usages of the Device as well as the target market segment of the Device, as well as cost and business considerations. The Device RoT protection level is one of the capabilities signed by Model Certificate's private key and reported by the Device during the Device Authentication.

The RoT, Device private key, shall never be exposed in plaintext form outside of the Device after the Device private key is provisioned on the Device for all levels of protection. Furthermore, the RoT shall never be modified after the Device private key is provisioned on the Device for all levels of protection.

- Level 1: the Device private key is stored in plaintext form inside the Device (e.g. fuse, internal NVRAM) or stored externally in an encrypted form. The Device private key is accessible to mutable Device component, such as firmware.
- Level 2: the Device private key is encrypted and stored on the Device. For example, during manufacturing, the Device private key is encrypted using a key encryption key (KEK) which is known only to the immutable Device hardware. The Device private key is accessible only to Device hardware or immutable firmware. Mutable firmware does not have access to plaintext Device private key.
- Level 3: The Device private key is generated in the Device (e.g., using physically unclonable function circuitry) and accessible only to immutable components in the Device. In this case, the Device private key is never exposed outside the Device, not even during manufacturing.

5.1.5 Device Root-of-Trust for Measurement Protection

In order for the Authentication Verifier to make meaningful policy decisions based on the authentication result, all of the measurements of the Device's capability shall be done in a secure manner. Therefore, the Root-of-Trust for Measurement (RTM) shall be implemented as pure hardware components and/or immutable firmware components on the Device. In the event that the device supports live firmware patching as mentioned in 3.2.6, the value of at least one of the DIGEST registers must be reset and updated to reflect the new firmware stack. In this case, the Device must re-execute the RTM in conjunction with the reset of the DIGEST register in order to update the DIGEST register.

5.1.6 Device Root-of-Trust for Reporting Protection

Once the Device RoT and Device RTM protections are in-place in the Device, the reporting of the measurements shall use the Device RoT to sign the measurements. Therefore, the Device Root-of-Trust for Reporting (RTR) is a combination of the Device RoT and Device RTM where the reporting mechanism shall be implemented as pure hardware components of immutable firmware components on the Device.

5.1.7 Side-Channel Protection Requirements

Device private key should be protected against software side-channel attacks as well as against hardware differential power analysis attacks, including all relevant keys and cryptographic algorithms related to the usage of the Device private key.

5.2 Device Certification

This section describes the security certification requirements for a Device in order for the Device to participate in Device Authentication. Two approaches can be taken for the security certification of a trust-worthy Device---1) Manufacturer/vendor process-based, and 2) External certification. For both 1) and 2), cost and business justifications as well as the expected privilege given to a Device will determine which certification will be required.

5.2.1 Manufacturer/Vendor Process-Based

Device manufactures or vendors can seek to establish a certification program to certify their own Devices or Devices from other manufacturers. The certification can be issued by a central authority such as the PCI-SIG, once the Devices pass certain implementation and validation criteria to ensure the correctness as well as the security of their implementations, e.g., the Device protecting the Device RoT at Level 3.

Furthermore, the Device vendor and Authentication Verifier can establish certification programs where the Authentication Verifier participates in a set of checkpoint reviews to ensure the Device's implementation complies with this specification. For example, security architecture reviews, source code reviews, and white-hat hacking and security validations can be embedded into the Device's product development cycles.

5.2.2 External Certification

In addition to the internal and cross-organizational certification approach, manufacturers or vendors can also seek external or third-party certification of Devices. Existing governmental or commercial certification programs exist to certify a Device's implementation and design, such as Common Criteria.

5.3 Certificate Revocation

If and when a compromised private key is detected, its associated certificate as well as any lower-level certificates below the compromised certificate need to be revoked and the revocation information needs to be communicated to the Authentication Verifier. Each certificate shall include the location where the Certificate Revocation List (CRL) is maintained. The CRL is signed with the root CA's private key. It is the Verifier's responsibility to ensure the latest CRL is used to check against any certificate retrieved from the Device.

5.4 Key and Certificate Generation, Distribution, Provisioning and Recovery

5.4.1 Device Public-Private Key Pair Generation

A unique per-part device public-private key pair can be generated by a Device vendor and provisioned and stored securely on a Device, as outlined by the different protection levels in Section 5.1.4; or the key pair can be generated by the Device on the Device itself. For example, the Device Identifier Composition Engine (DICE) architectures specified by the Trusted Computing Group (TCG) can be leveraged to generate a Device Certificate, which alleviates the burden of secure key delivery during manufacturing. The Device-generated certificate needs to be signed with the Model private key and provisioned back to the Device for the chain-of-trust to be established.

5.4.2 Device ownership claim using SET_CERTIFICATE

Each Device is capable of carrying 8 unique certificate chains in slots 0-7, with slot 0 being dedicated to the Device vendor and provisioned at manufacturing time and protected by the Device RoT against tampering. In some scenarios it may be useful to “claim ownership” of the Device and simplify the authentication protocol by leveraging private certificate chain owned by the Device owner/user. In such case, slots 1-7 can be used to carry non-vendor certificates chains. Each certificate chain is required to sign the leaf *DeviceCert* which cannot be modified, therefore any non-vendor chain simply endorses the *DeviceCert*. This model assumes that Device owner authenticates the Device and Vendor certificate chain after obtaining the Device, creates the certificate chain using local Certificate Authority and signing the leaf *DeviceCert* and provisions the Owner certificate chain using the SET_CERTIFICATE command described in Section 4.10.2.5. After the provisioning, the Device owner can implement the Device authentication using only information about its own Root CA, which would simplify authentication when multiple Devices from different vendors are present in the system.

Certificate chains in slots 1-7 are not required to be protected by the Device RoT against tampering; instead, the platform in which the Device is being used should ensure the certificates are not maliciously modified, as illustrated in Figure 12.

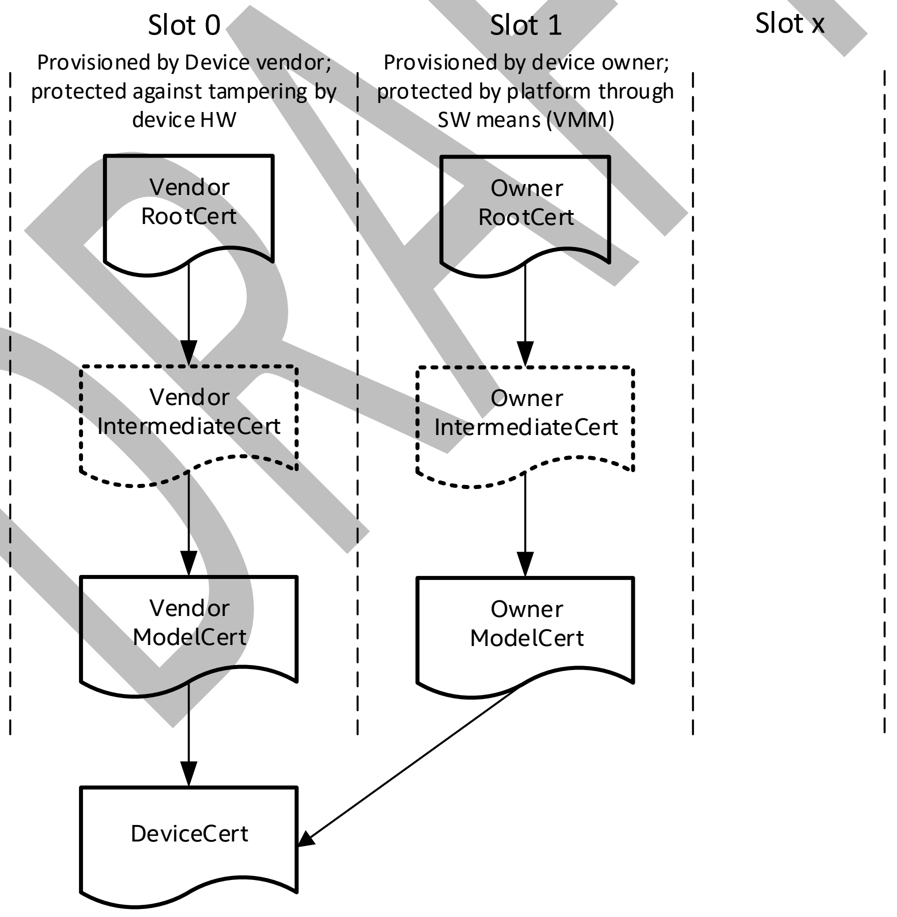

Slot 0
Provisioned by Device vendor; protected against tampering by device HW
Slot 1
Provisioned by device owner; protected by platform through SW means (VMM)
Slot x
Vendor RootCert
Vendor IntermediateCert
Vendor ModelCert
Owner RootCert
Owner IntermediateCert
Owner ModelCert
DeviceCert

Figure 12 Example use of certificate slots to claim ownership of the device.

5.5 Device Implementation for Authentication Support

To implement the Device Authentication capability, a Device must provide a hardware and/or firmware mechanisms to support the protocol and the mailbox-based messaging interface, and adequate protections to secure the Device private key both when in-use or at-rest, as described in Section 5.1.

5.6 Host Implementation for Authentication Support

For the Host, new firmware and/or software must be implemented to support the authentication protocol and mailbox-based messaging interface, and adequate mechanisms to ensure a valid root CA certificate (*RootCert*) is used when verifying the signature generated by a Device.