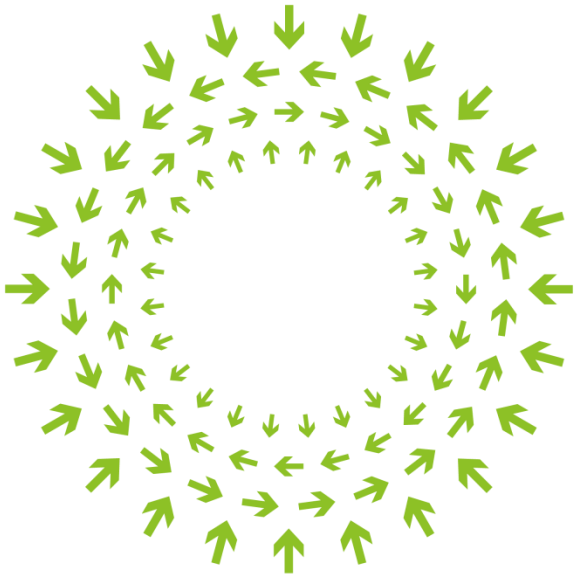


OPEN
Compute Project

Project Cerberus
Firmware Challenge Specification

Author:
Bryan Kelly, Principal Firmware Engineering Manager, Microsoft
Christopher Weimer Senior Firmware Engineer, Microsoft
Akram Hamdy Firmware Engineer, Microsoft

Revision History

Date	Description
28-08-2017	V0.01 - Initial Draft
28-09-2017	V0.02 - Add References section
28-10-2017	V0.03 - Move message exchange from protocol to register based
02-12-2018	V0.04 – Add MCTP Support and update session
04-30-2018	V0.05 – Incorporate Supplier feedback
10-15-2018	V0.06 – Update Authentication flow. Change measurement to PMR and attestation integration.
01-10-2019	V0.07 – Change PMR naming to PM due to static requirements on extension.
02-15-2019	V0.08 – Add Firmware Recovery image update commands. Clarify Error Response
06-26-2019	V0.09 – Add Reset Configuration command. Identify commands subject to the cryptographic timeout.
08-05-2019	V0.10 – Update Cerberus-defined MCTP message definition.
10-21-2019	V0.11 – Add detail on Mfg pairing for devices. Add commands to get RIoT, chip, and host reset information.
12-27-2019	V0.12 – Clarification regarding required and optional commands.
03-17-2020	V0.13 – Add commands to get manifest platform IDs and PMR measured data. Update unseal and device capabilities commands. Clarifications around command packet format. Add log formats.
04-30-2020	V0.14 – Update format of several commands, add extended update status. Clarifications around certificates. Add details about encrypted messages.
05-22-2020	V0.15 – Add unseal ECDH seed parameters. Define a range of reserved commands.
08-19-2020	V1.00 – Update session establishment and secure device binding. Add back Rq bit.

Table of Contents

Summary **7**

1 Physical Communication Channel **7**

1.1 Power Control *9*

2 Communication **10**

2.1 RSA/ECDSA Key Generation 10
2.2 Chained Measurements 12

3 Protocol and Hierarchy **13**

3.1 Attestation Message Interface 13
3.2 Protocol Format 15
3.3 Packet Format 16
3.4 Transport Layer Header 16
3.5 MCTP Messages 18
3.6 EID Assignment 19

4 Certificates **19**

4.1 Format 19
4.2 Certificate Chain 20

5 Authentication **20**

5.1 PA-RoT and AC-RoT Authentication 21
5.2 Secure Session Establishment 22
5.3 Secure Device Binding 24

6 Command Format **27**

6.1 Attestation Protocol Format 27
6.2 Command Set Type Code 28
6.3 RoT Commands 29
6.4 Message Body Structures 30
6.5 Error Message 30
6.6 Firmware Version 31
6.7 Device Capabilities 31
6.8 Device Id 34
6.9 Device Information 34
6.10 Export CSR 35
6.11 Import Certificate 35
6.12 Get Certificate State 35
6.13 GET DIGESTS 36
6.14 GET CERTIFICATE 37
6.15 CHALLENGE 37
6.16 Key Exchange 38
6.17 Session Sync 40
6.18 Get Log Info 40
6.19 Get Log 40
6.20 Clear Debug/Attestation Log 42

6.21 Get Attestation Data 42
6.22 Get Host State 43
6.23 Get Platform Firmware Manifest Id 43
6.24 Get Platform Firmware Manifest Supported Firmware 44
6.25 Prepare Platform Firmware Manifest 44
6.26 Update Platform Firmware Manifest 44
6.27 Activate Platform Firmware Manifest 45
6.28 Get Component Firmware Manifest Id 45
6.29 Prepare Component Firmware Manifest 46
6.30 Update Component Firmware Manifest 46
6.31 Activate Component Firmware Manifest 46
6.32 Get Component Firmware Manifest Component IDs 46
6.33 Get Platform Configuration Data Id 47
6.34 Prepare Platform Configuration Data 47
6.35 Update Platform Configuration Data 47
6.36 Activate Platform Configuration Data 48
6.37 Platform Configuration 48
6.38 Prepare Firmware Update 49
6.39 Update Firmware 49
6.40 Update Status 49
6.41 Extended Update Status 50
6.42 Activate Firmware Update 50
6.43 Reset Configuration 50
6.44 Get Configuration Ids 51
6.45 Recover Firmware 52
6.46 Prepare Recovery Image 52
6.47 Update Recovery Image 52
6.48 Activate Recovery Image 52
6.49 Get Recovery Image Id 53
6.50 Platform Measurement Register 53
6.51 Update Platform Measurement Register 54
6.52 Reset Counter 54
6.53 Message Unseal 54
6.54 Message Unseal Result 55
7 Platform Active RoT (PA-RoT) 56
7.1 Platform Firmware Manifest (PFM) and Component Firmware Manifest 56
7.2 RoT External Communication interface 57
7.3 Host Interface 58
7.4 Out Of Band (OOB) Interface 58
8 Legacy Interface 59
8.1 Protocol Format 59
8.2 PEC Handling 59
8.3 Message Splitting 59
8.4 Payload Format 60
8.5 Register Format 60

8.6 Legacy Active Component RoT Commands 61
8.7 Legacy Command Format 61
9 References 64
9.1 DICE Architecture....64
9.2 RIoT....64
9.3 DICE and RIoT Keys and Certificates 64
9.4 USB Type C Authentication Specification 64
9.5 PCIe Device Security Enhancements specification 64
9.6 NIST Special Publication 800-108....64
9.7 TCG PC Client Platform Firmware Profile Specification 64

List of Figures

Figure 1 Motherboard I2C lane diagram 8
Figure 2 RioT Core Key Generation 11
Figure 3 Certificate Generation 11
Figure 4 Measurement Calculation 12
Figure 5 BMC/UEFI Attestation Seed 12
Figure 6 Root of Trust Hierarchy 14
Figure 7 MCTP Encapsulated Message 15
Figure 8 Transport Layer Header 16
Figure 9 Authentication 22
Figure 10 Session Establishment 24
Figure 11 Secure Device Binding 26
Figure 12 External Communication Interface 58
Figure 13 Host Interface 58
Figure 14 Register Read Flow 60
Figure 15 Register Write Flow 60

List of Tables

Table 1 Field Definitions 16
Table 2 Vendor Defined Message 18
Table 3 Recommended Algorithms for Interoperability 20
Table 4 MCTP Message Format 28
Table 5 Encrypted Cerberus message body 28
Table 6 Command Types 28
Table 7 Command List 29
Table 8 Error Response 30
Table 9 Error Codes 31
Table 10 Firmware Version Request 31
Table 11 Firmware Version Response 31
Table 12 Device Capabilities Request 32
Table 13 Device Capabilities Response 33
Table 14 Device Id Request 34
Table 15 Device Id Response 34
Table 16 Device Information Request 34
Table 17 Device Information Response 34
Table 18 Export CSR Request 35
Table 19 Export CSR Response 35

Table 20 Import Certificate Request 35
Table 21 Get Certificate State Request 36
Table 22 Get Certificate State Response 36
Table 23 GET DIGEST Request 36
Table 24 GET DIGEST Response 36
Table 25 GET CERTIFICATE Request 37
Table 26 GET CERTIFICATE Response 37
Table 27 CHALLENGE Request 37
Table 28 CHALLENGE Response 37
Table 29 Key Exchange Request 38
Table 30 Key Exchange Response 38
Table 31 Key Exchange Type 0 Request Data 39
Table 32 Key Exchange Type 0 Response Data 39
Table 33 Key Exchange Type 1 Request Data 39
Table 34 Key Exchange Type 1 Response Data 39
Table 35 Key Exchange Type 2 Request Data 39
Table 36 Key Exchange Type 2 Response Data 40
Table 37 Session Sync Request 40
Table 38 Session Sync Response 40
Table 39 Get Log Info Request 40
Table 40 Get Log Info Response 40
Table 41 Log Types 40
Table 42 Get Log Section Request 41
Table 43 Get Debug/Attestation Log Response 41
Table 44 Log Entry Header 41
Table 45 Attestation Entry Format 41
Table 46 Debug Entry Format 42
Table 47 Clear Debug/Attestation Log Request 42
Table 48 Get Attestation Data Request 42
Table 49 Get Attestation Data Response 43
Table 50 Get Host State Request 43
Table 51 Get Host State Response 43
Table 52 PFM Information Request 43
Table 53 PFM Version Id Response 43
Table 54 PFM Platform Id Response 43
Table 55 PFM Supported Firmware Request 44
Table 56 Supported Firmware Response 44
Table 57 Prepare PFM Request 44
Table 58 Update PFM Request 44
Table 59 Update PFM Request 45
Table 60 Get CFM Id Request 45
Table 61 CFM Version Id Response 45

Table 62 CFM Platform Id Response 45
Table 63 Update Component Firmware Manifest Request 46
Table 64 Active CFM Request 46
Table 65 Get Platform Configuration Data Request 47
Table 66 Get Platform Configuration Data Version Id Response 47
Table 67 Get Platform Configuration Data Platform Id Response 47
Table 68 Update Platform Configuration Data Request 47
Table 69 Active PCD Request 48
Table 70 PCD Structure 48
Table 71 Prepare Firmware Update 49
Table 72 Update Firmware Request 49
Table 73 Update Status Request 49
Table 74 Update Status Response 49
Table 75 Extended Update Status Request 50
Table 76 Extended Update Status Response 50
Table 77 Activate Firmware Update 50
Table 78 Reset Configuration Request 51
Table 79 Reset Configuration Response with Authorization Token 51
Table 80 Get Configuration Ids Request 51
Table 81 Get Configuration Ids Response 51
Table 82 Recover Firmware Request 52
Table 83 Prepare Recovery Image Request 52
Table 84 Update Component Firmware Manifest Request 52
Table 85 Activate Recovery Image 52
Table 86 Recovery Image Id Request 53
Table 87 Recovery Image Version Id Response 53
Table 88 Recovery Image Platform Id Response 53
Table 89 Platform Measurement Request 53
Table 90 Platform Measurement Response 53
Table 91 Update Platform Measurement Request 54
Table 92 Reset Counter Request 54
Table 93 Reset Counter Response 54
Table 94 Unseal Message Request 54
Table 95 Unseal Message Request 55
Table 96 Unseal Message Pending Response 55
Table 97 Unseal Message Completed Response 56
Table 98 PFM Attributes 56
Table 99 Commands 61
Table 100 Status Register 61
Table 101 Challenge Register 62
Table 102 Measurement Register 62

Summary

Throughout this document, the term “Processor” refers to all Central Processing Unit (CPU), System On Chip (SOC), Micro Control Unit (MCU), and Microprocessor architectures. The document details the required challenge protocol required for Active Component and Platform RoTs. The Processor must implement all required features to establish a hardware based Root of Trust. Processors that intrinsically fail to meet these requirements must implement the flash protection Cerberus RoT described Physical Flash Protection Requirements document.

Active Components are add-in cards and peripherals that contain Processors, Microcontrollers or devices that run soft-logic.

This document describes the protocol used for attestation measurements of firmware for the Platform’s Active RoT. The specification encompasses the pre-boot, boot and runtime challenge and verification of platform firmware integrity. The hierarchical architecture extends beyond the typically UEFI measurements, to include integrity measurements of all Active Component firmware. The document describes the APIs needed to support the attestation challenge for Project Cerberus.

1 Physical Communication Channel

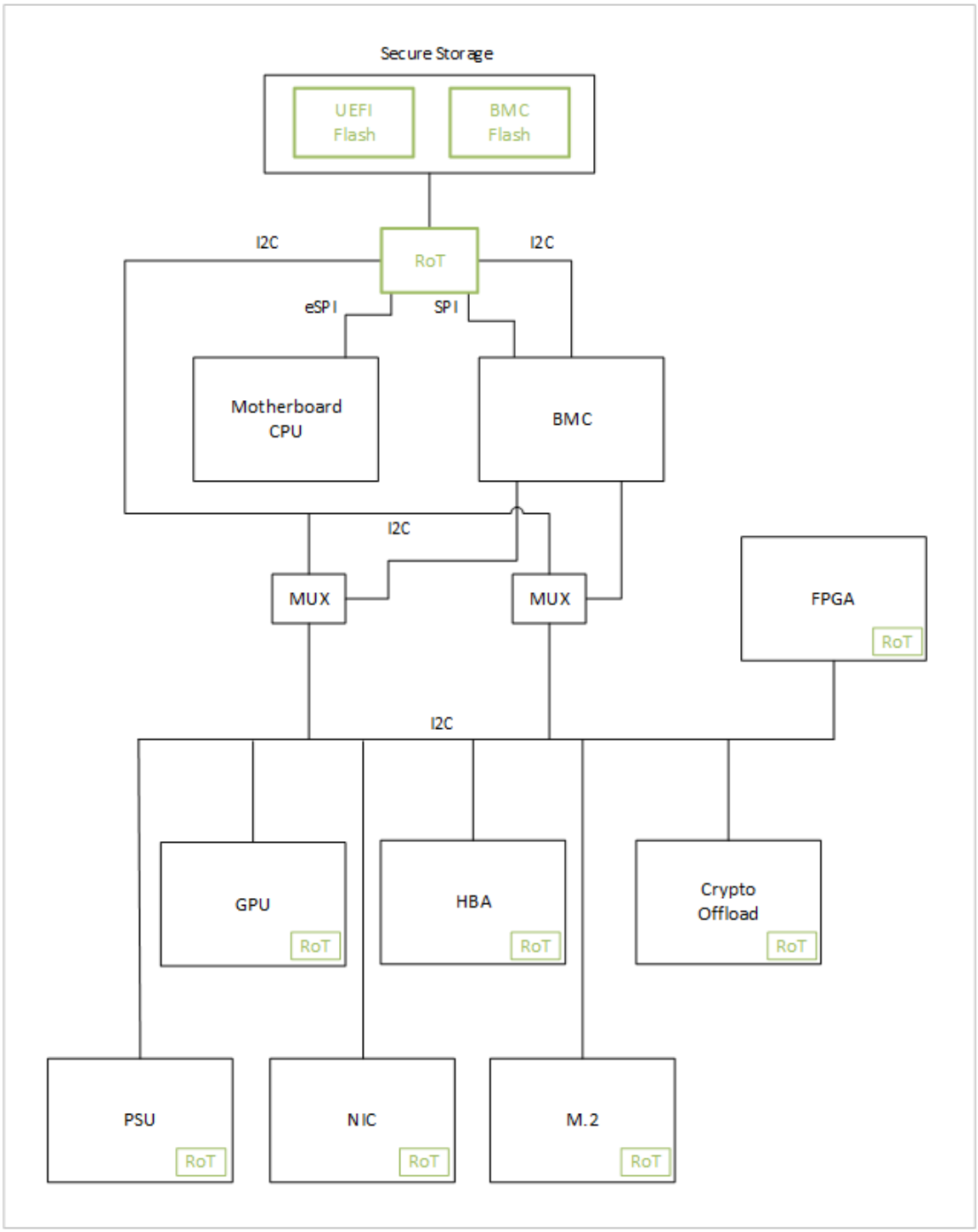
The typically cloud server motherboard layout has I2C buses routed to all Active Components. These I2C buses are typically used by the Baseboard Management Controller (BMC) for the thermal monitoring of Active Components. In the Cerberus board layout, the I2C lanes are first used by the platform Cerberus microcontroller during boot and pre-boot, then later mux switched back to the BMC for thermal management. Cerberus can at any time request for the BMC to yield control for runtime challenge and attestation. Cerberus controls the I2C mux position, and coordinates access during runtime. It is also possible for Cerberus to proxy commands through the BMC at runtime, with the option for link encryption and asymmetric key exchange, making the BMC blind to the communications. The Cerberus microcontroller on the motherboard is referred to as the Platform Active Root-of-Trust (PA-RoT). This microcontroller is head of the hierarchical root-of-trust platform design and contains an attestable hash of all platform firmware kept in the Platform Firmware Manifest (PFM) and Component Firmware Manifest (CFM).

Most cloud server motherboards route I2C to Active Components for thermal monitoring, the addition of the mux logic is the only modification to the motherboard. An alternative to adding the additional mux, is to tunnel a secure challenge channel through the BMC over I2C. Once the BMC has been loaded and attested by Cerberus, it can act as an I2C proxy. This approach is less desirable, as it limits platform attestation should the BMC ever fail attestation. In either approach, physical connectors to Active Component interfaces do not need to change as they already have I2C.

Active Components with the intrinsic security attributes described in the “Processor Secure Boot Requirements” document do not need to place the physical Cerberus microcontroller between their

Processor and Flash. Active Components that do not meet the requirements described in the "Processor Secure Boot Requirements" document are required to implement the Cerberus micro-controller between their Processor and Flash to establish the needed Root-of-Trust. Figure 1 Motherboard I2C lane diagram, represents the pre-boot and post-boot measurement challenge channels between the motherboard PA-RoT and Active Component RoTs (AC-RoT).

Figure 1 Motherboard I2C lane diagram



The Project Cerberus firmware attestation is a hierarchical architecture. Most Active Components in the modern server boot to an operational level before the platform's host processors complete their initialization and become capable of challenging the devices. In the Cerberus design, the platform is held in pre-power-on or reset state, whereby Active Components are quarantined and challenged for their firmware measurements. Active Components must respond to challenges from the PA-RoT confirming the integrity of their firmware before they are taken out of quarantine.

In this version of the Cerberus platform design, the PFM and CFMs are static. The manifest is programmable through the PA-RoT's communication interface. Auto-detection of Active Components and computation of the PFM/CFM will be considered in future version of the specification. The PFM and CFM are manifests of allowed firmware versions and their corresponding firmware measurements. The manifests contain a monotonic identifier used to restrict rollbacks.

The PA-RoT uses the measurements in the CFM to challenge the Active Components and compare their measurements. The PA-RoT then uses the digest of these measurements as the platform level measurement, creating a hierarchical platform level digest that can attest the integrity of the platform and active component firmware.

The PA-RoT will support Authentication, Integrity and Confidentiality of messages. Active Components RoT's (AC-RoT) will support Authentication and Integrity of messages and challenges. To facilitate this, AC-RoT are required to support certificate authentication. The Active Component will support a component unique CA signed challenge certificate for authentication.

Note: I2C is a low speed link, there is a performance tradeoff between optimizing the protocol messages and strong cryptographic hashing algorithms that carry higher bit counts. RoT's that cannot support certificate authentication are required to support hashing algorithms and either RSA or ECDSA signatures of firmware measurements.

1.1 Power Control

In the Cerberus motherboard design, power and reset sequencing is orchestrated by the PA-RoT. When voltage is applied to the motherboard, it passes through in-rush circuity to a CPLD that performs time sensitive sequencing of power rails to ensure stabilization. Once a power good level is established, the platform is considered powered on. Upon initial powering of the platform in the Cerberus design, the only active component powered-on is the PA- RoT. The RoT first securely loads and decompresses its internal firmware, then verifies the integrity of Baseboard Management Controller (BMC) firmware by measuring the BMC flash. When the BMC firmware has been authenticated, the Active RoT enables power to be applied to the BMC. Once the BMC has been powered, the Active RoT authenticates the firmware for the platform UEFI, during which time the RoT sequences power to the PCIe slots and begins Active Component RoT challenge. When the UEFI has been authenticated, the platform is held in system reset and the Active RoT will keep the system in reset until AC-RoTs have responded to the measurement challenge. Any PCIe ports that do not respond to their measurement challenge will be subsequently unpowered. Should any of the expected Active Components fail to respond to the

measurement challenge, Cerberus policies determine whether the system should boot with the Active Component powered off, or the platform should remain on standby power, while reporting the measurement failure to the Data Center Management Software through the OOB path.

2 Communication

The Cerberus PA-RoT communicates with the AC-RoT's over I2C. The protocol supports an authentication and measurement challenge. The Cerberus PA-RoT generates a secure asymmetric key pair unique to the microcontroller closely following the DICE architecture. Private keys are inaccessible outside of the secure region of the Cerberus RoT. Key generation and chaining follows the RIoT specification, described in section: 9.3 DICE and RIoT Keys and Certificates. Derived platform alias public keys are available for use in attestation and communication from the AC-RoT's during the challenge handshake for establishing communication.

2.1 RSA/ECDSA Key Generation

The Cerberus platform Active RoT should support the Device Identifier Composition Engine (DICE) architecture. In DICE, the Compound Device Identifier (CDI) is used as the foundation for device identity and attestation. Keys derived from the Unique Device Secret (UDS) and measurement of first mutable code are used for data protection within the microcontroller and attestation of upper firmware layers. The Device Id asymmetric key pair is derived cryptographically from the CDI and associated with the Device Id Certificate. The CDI uses the UDS derived from PUF and other random entropy including the microcontroller unique id and Firmware Security Descriptors.

Cerberus implements the RIoT Core architecture for certificate generation and attestation. For details on key generation on DICE and RIoT, review section: 9.3 DICE and RIoT Keys and Certificates.

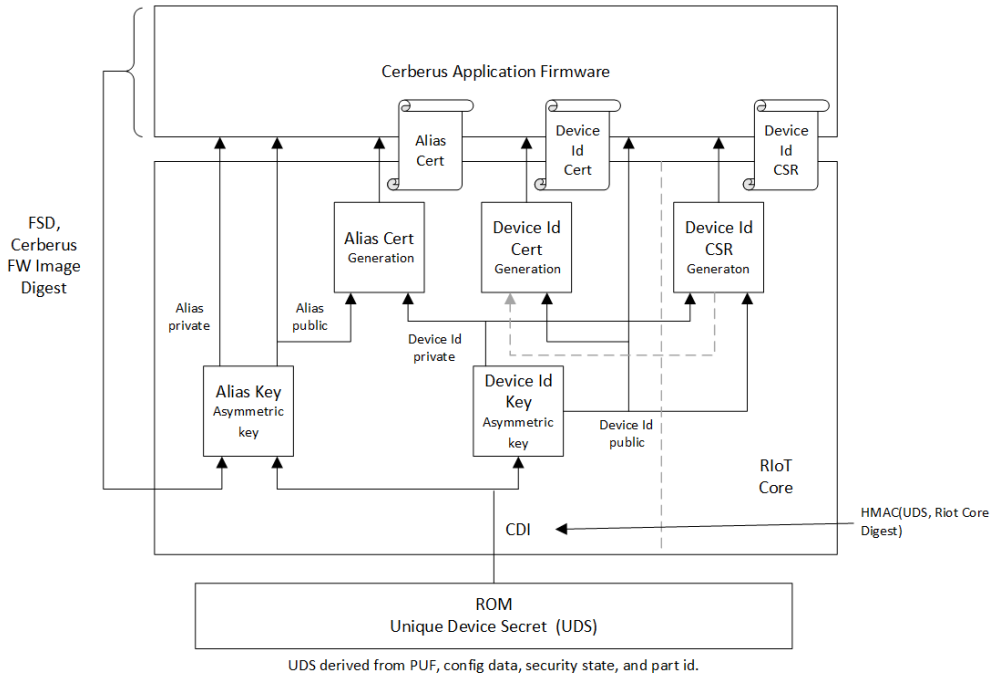
Note: The CDI is a compound key based on the UDS, Microcontroller Security Descriptors and second stage bootloader (mutable) measurement. The second stage bootloader is mutable code, but not typically updated with firmware updates. Changes to the second stage bootloader will result in a different CDI, resulting in different asymmetric Device Id key generation. Certificates associated with the previous Device key will be invalidated and a new Certificate would need to be signed.

A second asymmetric key pair certificate is created in the RIoT Core layer of Cerberus and passed to the Cerberus Application firmware. This key pair forms the Alias Certificate and is derived from the CDI, Cerberus Firmware Security Descriptor and measurement of the next stage Cerberus Firmware.

Proof-of-knowledge of the CDI derived private key known as the Device Id private key is used as a building-block in a cryptographic protocol to identify the device. The Device Id private key is used to sign the Alias Certificate, thus verifying the integrity of the key. During initial provisioning, the Device Id

Certificate is CA signed by the Microsoft Certificate Authority. When provisioned, the Device Id keys must match the previously signed public key.

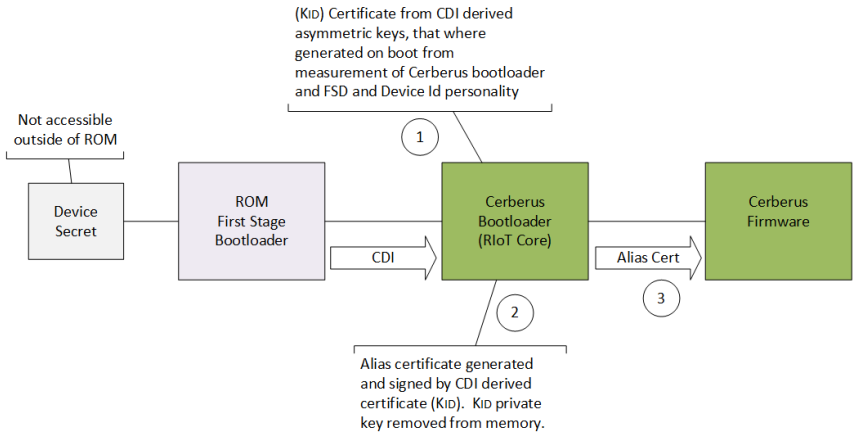
Figure 2 RioT Core Key Generation



Note: The CDI and Device Id private key are security erased before exiting RIoT Core.

Each layer of the software can use its private key certificate to sign and issue a new certificate for the next layer, each successive layer continues this chain. The certificate in the application layer (Alias Certificate) can be used when authenticating with devices and establishing a secure channel. The certificate can also establish authenticity. Non-application layer private keys used to sign certificates must be accessible only to the layer they are generated. The Public keys are persisted or passed to upper layers, and eventually exchanged with upstream and downstream entities.

Figure 3 Certificate Generation

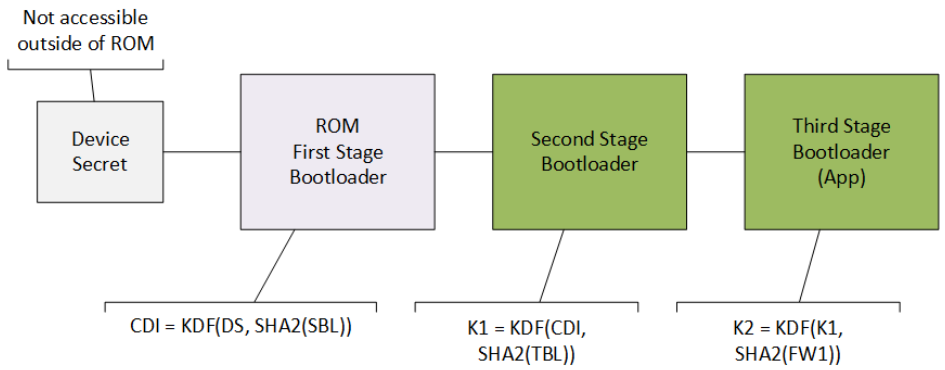


2.2 Chained Measurements

The Cerberus firmware measurements are based on the Device Identifier Composition Engine (DICE) architecture: https://trustedcomputinggroup.org/work-groups/dice-architectures

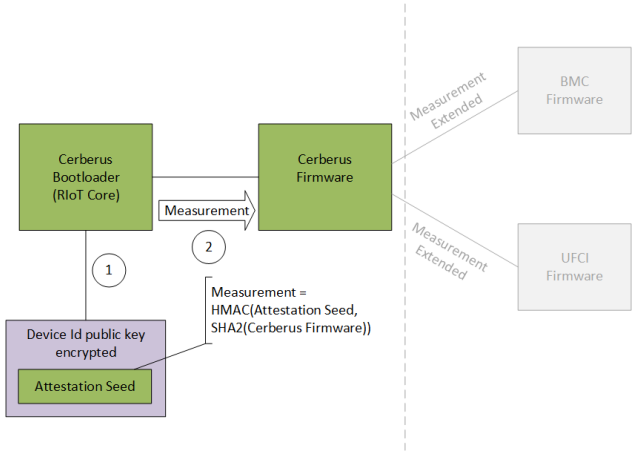
The first mutable code on the RoT is the Second Bootloader (SBL). The CDI is a measurement of the HMAC(Device Secret Key + Entropy, H(SBL)). This measurement then passes to the second stage boot loader, that calculates the digest of the Third Bootloader (TBL). On the Cerberus RoT this is the Application Firmware: HMAC(CDI, H(TBL)).

Figure 4 Measurement Calculation



The Third Stage Bootloader (TBL) which runs the Cerberus Application Firmware will take additional area measurements of the SPI/QSPI flash for the Processor it protects, measuring both active and inactive areas. The TBL measurements are verified and extended with an attestation freshness seed. The final measurement is signed, sealed, and made available to the challenge software.

Figure 5 BMC/UEFI Attestation Seed



Seeds for attesting firmware by the application firmware can be extended from Cerberus Firmware measurements, or using the Alias Certificates a dedicated freshness seed can be provided for measuring the protected processor firmware.

The measurements are stored in either firmware or hardware register values within the PA-RoT. The seed is typically transferred to the device using the Device Cert, Alias Cert or Attestation Cert.

3 Protocol and Hierarchy

The following section describes the capabilities and required protocol and Application Programming Interface (API) of the motherboard's Platform Active RoT (PA-RoT) and Active Component to establish a platform level RoT. The Cerberus Active RoT and Active Component RoTs are required to support the following I2C protocol.

The protocol is derived from the MCTP SMBus/I2C Transport Binding Specification. A limited version of the protocol is defined for devices that do not support MCTP. If an AC-RoT implements the Attestation Protocol over MCTP, it may also optionally implement the minimum attestation protocol over native SMBus/I2C.

3.1 Attestation Message Interface

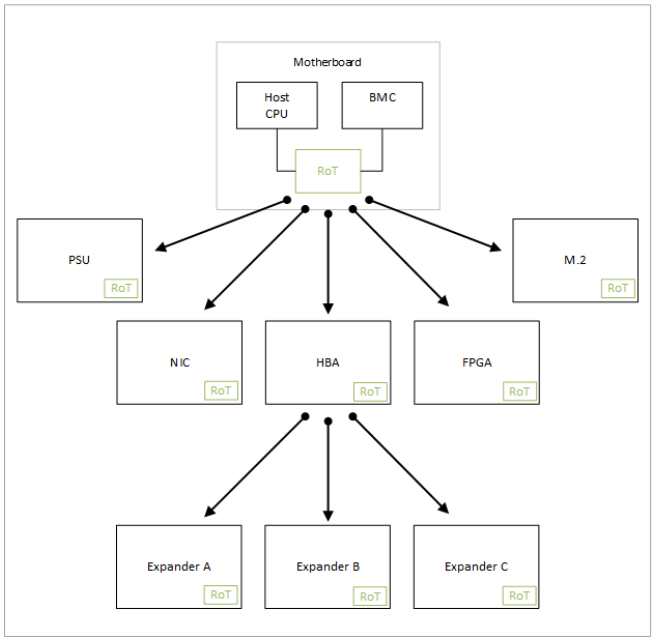
The Attestation Message Interface uses the MCTP over I2C message protocol for transporting the Attestation payloads. The AC-RoT MCTP Management Endpoint should implement the required behaviors detailed in the Management Component Transport Protocol (MCTP) Base Specification, in relation to the MCTP SMBus/I2C Transport Binding Specification. The following section outlines additional requirements upon the expected behavior of the Management Endpoint:

- The Message Interface Request and Response Messages are transported with a custom message type.
- MCTP messages will be transmitted in a synchronous Request and Response manner only. An Endpoint (AC-RoT) should never initiate a Request Message to the Controller (PA-RoT).
- MCTP Endpoints must strictly adhere to the response timeout defined in this specification. When an Endpoint receives a standard message, it should be transmitting the response within 100ms. If the Endpoint has not begun transmitting the Response Message within 100ms, it should drop the message and not respond.
- MCTP Endpoints must strictly adhere to the response timeout advertised for cryptographic commands. Cryptographic commands include transmission of messages signature generation and verification. The cryptographic command timeout multiplier is negotiated in the Device Capabilities command.
- MCTP leaves Authentication to the application implementation. This specification partially follows the flow of USB Authentication Specification flow, when authentication has been established attestation seeds can be exchanged.
- It is not required that the Management Endpoint response to ARP messages. AC-RoT Endpoints should not generate any ARP messages to Notify Master. Devices should be aware they are normally behind I2C muxes and should not master the I2C bus outside of the allotted time they are provided to response to an MCTP Request Message.
- MCTP Endpoint devices should be response only.
- Irrespective as to whether Endpoints are ARP capable, they should operate in a Non-ARP-capable manner.

- MCTP specifications use big endian byte ordering while this specification uses little endian byte ordering. This ordering does not change the payload order in which bytes are sent out on the physical layer.
- Endpoints should support Fixed Addresses; Endpoint IDs are supported to permit multiple MCTP Endpoints behind a single physical address.
- As defined in the MCTP SMBus/I2C Transport Binding Specification Endpoints should support fast-mode 400KHz.
- Endpoint devices that do not support multi-master should operate in slave mode. The PA-RoT PCD will identify the mode of the device. The Master will issue SMBUS Write Block in MCTP payload format, the master will then issue an I2C Read for the response. The Master read the initial 12 bytes and use byte 3 and the MCTP EOM header bits to determine if additional SMBUS read commands are required to collect the remainder of the response message.
- Endpoints should support EID assignment using the MCTP Set Endpoint ID control message.
- Endpoints should support the MCTP Get Vendor Defined Message Support control message to indicate support level for the Cerberus protocol.

The Platform Cerberus Active RoT is always the MCTP master. Active Component RoT's can be configured as Endpoint, or Endpoint and Master. An Active Component RoT Endpoint and Master should interface to separate physical busses. There is no requirement for master arbitration as master and slave definitions are hierarchically established. The only hierarchy whereby the Active Component RoT becomes both Endpoint and Master is when there is a downstream sub-device, such as the Host Bus Adapter (HBA) depicted in the following block diagram:

Figure 6 Root of Trust Hierarchy



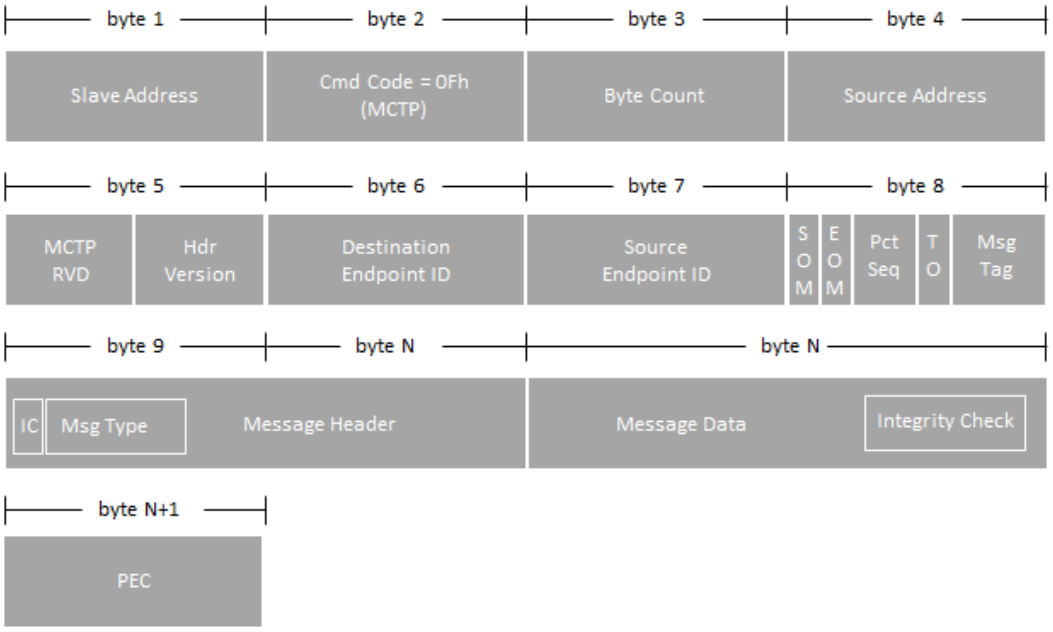
In this diagram, the HBA RoT is an Endpoint to the Platform Active RoT and Master to the downstream HBA Expanders. To the Platform's Active RoT, the HBA is an Endpoint RoT. To the HBA Expanders, the HBA Controller is a Master RoT.

The messaging protocol encompasses Management Component Transport Protocol (MCPT) Base Specification, in relation to the MCTP SMBus/I2C Transport Binding Specification, whereby the Active Component RoT is Endpoint and the Platform's Active RoT as Master.

3.2 Protocol Format

All MCTP transactions are based on the SMBus Block Write bus protocol. The following diagram shows MCTP encapsulated message.

Figure 7 MCTP Encapsulated Message



A package should contain a minimum of 1 byte of payload, with the maximum not to exceed the negotiated MCTP Transmission Unit Size. The byte count indicates the number of subsequent bytes in the transaction, excluding the PEC.

The PEC at the end of each transaction is calculated using a standard CRC-8, which uses polynomial $x^8 + x^2 + x + 1$, initial value of 0, and final XOR of 0. This CRC is independent of the message integrity check defined by the MCTP protocol. The CRC is calculated over the entire encapsulated MCTP packet, which includes all headers and the destination I2C address as it was sent over the I2C bus (bytes 1 through N in Figure 7 MCTP Encapsulated Message). Since the I2C slave address is not typically included as part of the transaction data, the CRC is equal to CRC8 (7-bit address << 1 || I2C payload). For example, an MCTP packet sent to a device with 7-bit I2C address 0x41 would have a CRC calculated as CRC8 (0x82 || packet).

3.3 Packet Format

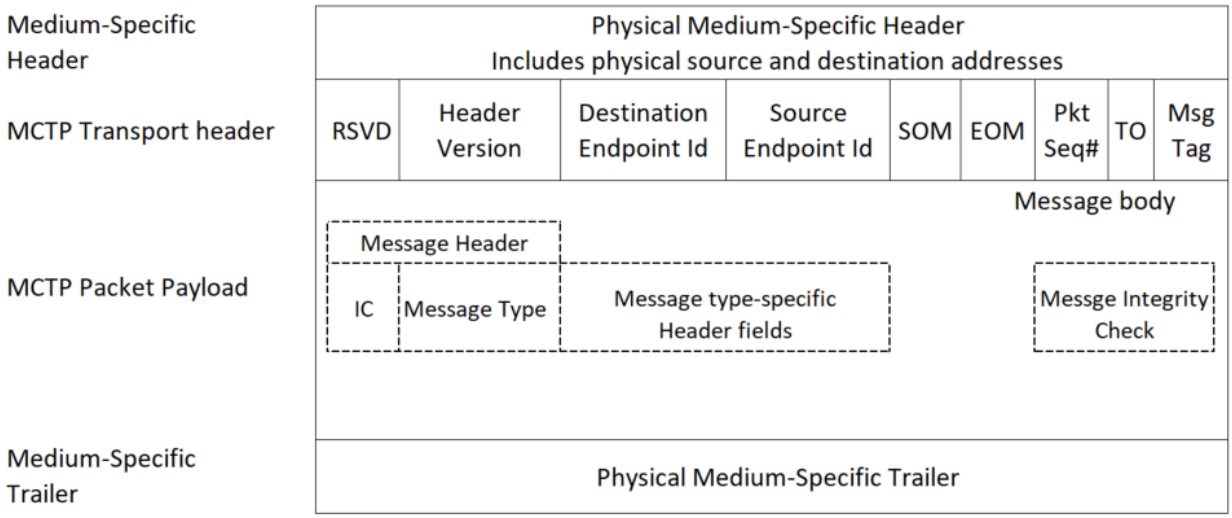
The Physical Medium-Specific Header and Physical Medium-Specific Trailer are defined by the MCTP transport binding specification utilized by the port. Refer to the MCTP transport binding specifications.

A compliant Management Endpoint shall implement all MCTP required features defined in the MCTP base specification.

The base protocol's common fields include message type field that identifies the higher layer class of message being carried within the MCTP protocol.

3.4 Transport Layer Header

Figure 8 Transport Layer Header



The Management Component Transport Protocol (MCTP) Base Specification defines the MCTP packet header (refer to DSP0236 for field descriptions). The fields of an MCTP Packet are shown in Table 1 Field Definitions.

Table 1 Field Definitions

Field Name	Description	Field Size
Medium-Specific Header	This represents the header for the protocol that encapsulates MCTP packets over a physical medium	Variable
Medium-Specific Trailer	This represents the trailer fields for the protocol that encapsulates MCTP packets over a physical medium	Variable
MCTP Transport Header	Provides version and addressing for the packet.	32 bits

RSVD	Reserved	4 bits
Header Version	Header Version Identifies the format of physical framing and data integrity.	4 bits
Destination Endpoint Id	The EID to the endpoint to receive the MCTP packet.	8 bits
Source Endpoint Id	The EID of the originator of the MCTP packet	8 bits
SOM	Start of Message is set to true (1b) for the first packet of a message.	1 bit
EOM	End of Message is set to true (1b) for the last packet of a message.	1 bit
Pkt Seq#	Packet Sequence Number for messages that span multiple packets. Increments modulo 4 on each successive packet up through the packet contained the EOM flag set.	2 bits
Message Tag	Combined with Source Endpoint Id and TO field to identify unique message at MCTP transport layer. For messages that are split up into multiple packets, the TO and Message Tag bits remain the same for all packets from the SOM to the EOM.	3 bits
TO	Tag Owner bit identifies whether the message tag was originated by the endpoint that is the source of the message or by the endpoint that is the destination of the message. MCTP message types use this for Request/Response messages.	1 bit
Message body	Payload of the MCTP message, can span multiple MCTP packets	Variable
IC	MCTP Integrity check bit 0 = No MCTP message integrity 1 = MCTP message integrity check is present	1 bit
Message Type	Defines the type of payload within the MCTP message header and data. Message type codes are defined in the MCTP ID and Codes	7 bits
Message header	Header data for the message type.	Variable
Message Data	Data for the message defined by the message type	Variable
MCTP Packet Payload	Payload of the message body carried in the packet. Limited by the transfer unit size. Review MCTP Base Specification for further details.	Variable
Message Integrity Check	Message type specific integrity check over the contest of the message body	Variable

Null (0) Source and Destination EIDs are typically supported, however AC-RoT devices that have multiple MCTP Endpoints may specify an EID value greater than 7 and less than 255. The PA-RoT does not broadcast any MCTP messages.

3.5 MCTP Messages

An MCTP message consists of one or more MCTP packets. There are typically two types of Messages, MCTP Control Messages and MCTP Cerberus Messages. Control Messages are standard MCTP messages with a maximum message body of 64 bytes. Cerberus Messages are those defined in this specification. The maximum message body for these messages is 4096 bytes, but this size can be negotiated to be smaller based on device capabilities.

MCTP Control Messages do not contain the per-packet CRC described in section 3.2.

3.5.1 Message Type

The message type should be 0x7E as per the Management Component Transport Protocol (MCTP) Base Specification. The message type is used to support Vendor Defined Messages where the Vendor is identified by the PCI based Vendor ID. The initial message header is specified in the Management Component Transport Protocol (MCTP) Base Specification, and detailed below for completeness:

Table 2 Vendor Defined Message

Message Header	Byte	
Request Data	1:2	PCI/PCIe Vendor ID. The MCTP Vendor Id formatted per 00h Vendor ID format offset.
	3:N	Vendor-Defined Message Body. 0 to N bytes.
Response Data	1:2	PCI/PCIe Vendor ID, the value is formatted per 00h Vendor ID offset
	3:M	Vendor-Defined Message Body. 0 to M bytes

The Vendor ID is a 16-bit Unsigned Integer, described in the PCI 2.3 specification. The value identifies the device manufacturer.

The message body and content are described in Table 4 MCTP Message Format.

3.5.2 Message Fields

The format of the MCTP message consists of a message header in the first two bytes, followed by the message data, and ending with the Message Integrity Check.

The Message header contains a Message Type (MT) field and Integrity Check (IC) that are defined by the MCTP Base Specification. The Message Type field indicate

3.5.3 Message Integrity Check

The Message Integrity Check field contains a 32-bit CRC computed over the contents of the message

3.5.4 Packet Assembly into Messages

An MCTP message may be split into multiple MCTP Packet Payloads and sent as a series of packets. Refer to the MCTP Base Specification for packetization and message assembly rules.

3.5.5 Request Messages

Request Messages are messages that are generated by a Master MTCP Controller and sent to an MCTP Endpoint. Request Messages specify an action to be performed by the Endpoint. Request Messages are either Control Messages or Cerberus Messages.

3.5.6 Response Messages

Response Messages are messages that are generated when an MCTP Endpoint completes processing of a previously issued Request Message. The Response Message must be completed within the allocated time or discarded.

3.6 EID Assignment

The BMC will assign EIDs to the different Active Component RoT devices. All Active Component RoT devices should support the MCTP Set Endpoint ID control request and response messages. The Platform Active RoT will have a static EID of 0x0B.

4 Certificates

The PA-RoT and AC-Rot will have a minimum of two certificates: Device Id Certificate (typically CA signed by offline CA) and the Alias Certificate (signed by the Device Id Certificate). The PA-RoT may also have an additional Attestation Certificate signed by the Device Id Certificate.

Certificates follow the 9.3 DICE and RIoT Keys and Certificates.

4.1 Format

All Certificates shall use the X509v3 ASN.1 structure. All Certificates shall use binary DER encoding for ASN.1. All Certificates shall be compliant with RFC5280. To facilitate certificate chain authentication, Authority and Subject Key Identifier extensions must be present. Further certificate requirements are defined in the 9.3 DICE and RIoT Keys and Certificates. Extensions beyond those required by RFC5280 or DICE are allowed so long as they conform to the appropriate standards. Custom extensions must be marked non-critical.

4.1.1 Textual Format

All text ASN.1 objects contained within Certificates, shall be specified as either a UTF8String, PrintableString, or IA5String. The length of any textual object shall not exceed 64 bytes excluding the DER type and DER length encoding.

4.1.2 Distinguished Name

The distinguished name consists of many attributes that uniquely identify the device. Distinguished name uniqueness can be accomplished by including attributes such as the serial number.

4.1.3 Object Identifier

Object Identifier should follow 9.3 DICE and RIoT Keys and Certificates

4.1.4 Serial Number

As per 9.3 DICE and RIoT Keys and Certificates, the Certificate *Serial Numbers* MUST be statistically unique per-Alias Certificate.

If the security processor has an entropy source, an 8-octet random number MAY be used.

If the security processor has does not have an entropy source, then an 8-octet *Serial Number* MAY be generated using a cryptographically secure key derivation function based on a secret key, such as those described in SP800-108 [9.6 NIST Special Publication 800-108]. The *Serial Number* MUST be unique for each generated certificate. For the Alias Certificate, this SHOULD be achieved by incorporating the FWID into the key derivation process (e.g. as the *Context* value in SP-800-108)

If the 8-octet serial number generated is not a positive number, it may be padded with an additional octet to maintain compliance with RFC5280.

4.2 Certificate Chain

The maximum Certificate Chain Size is 4096 bytes. There is no additional encoding necessary for the certificate chain as each certificate can be retrieved individually.

The certificate recommended cryptographic methods for interoperability are defined in Table 3 Recommended Algorithms for Interoperability

Table 3 Recommended Algorithms for Interoperability

Method	Use
X509v3, DER encoding	Certificate format
ECDSA, NIST P256, secp256r1 curve, uncompressed point	Digital signing of Certificate
SHA256	Hash algorithm

5 Authentication

Authentication is the process used to establish trust in a specific device and prove integrity of the device firmware. Once a device is authenticated, a secure session can optionally be established to provide confidentiality. For some devices, a secure session can also be used to enable a cryptographic binding that can be used for additional security or to enable additional functionality.

A device only needs to support a single session from another endpoint. In single session devices, the establishment of a new session will supersede and terminate the existing session to permit the new session. The previous session will be terminated upon receiving an out of session (unencrypted) Get Digests request specifying a requested key exchange. Get Digests requests received in session (encrypted) or without requesting key exchange will not cause the active session to terminate.

5.1 PA-RoT and AC-RoT Authentication

Devices are authenticated using the Alias certificate chain and signed firmware measurements. The certificate chain is endorsed by the Certificate Authority signing the Device Id Certificate. The certificate hierarchy is explained in the TCG Implicit Identity Based Device Attestation Reference.

The certificate authentication flow closely follows the USB Authentication Architecture and Authentication Messages. The command structures defined in this specification are derived from the USB-C Authentication Protocol. Relevant sections of the specification are as follows:

Section 3 Authentication Architecture of USB Authentication Specification

Section 4 Authentication Protocol of USB Authentication Specification

Section 5 Authentication Messages of USB Authentication Specification

The authentication sequence starts with the master (e.g. PA-RoT) issuing a Get Digests command. The slave (e.g. AC-RoT) responds with a list of SHA256 hashes for each certificate in the device's Alias certificate chain. If the master has cached the certificate chain, it may optionally skip requesting these certificates.

If the master does not have the correct certificates, it will issue a Get Certificate request for each certificate in the slave's Alias certificate chain. The slave will respond with the requested certificates, which must have origination from a trusted CA.

The master will verify the Alias certificate chain of the slave. If verification fails or the certificate has been revoked, the authentication will fail. The PA-RoT and AC-RoT can be updated with revocation patches, see firmware update specification for further details.

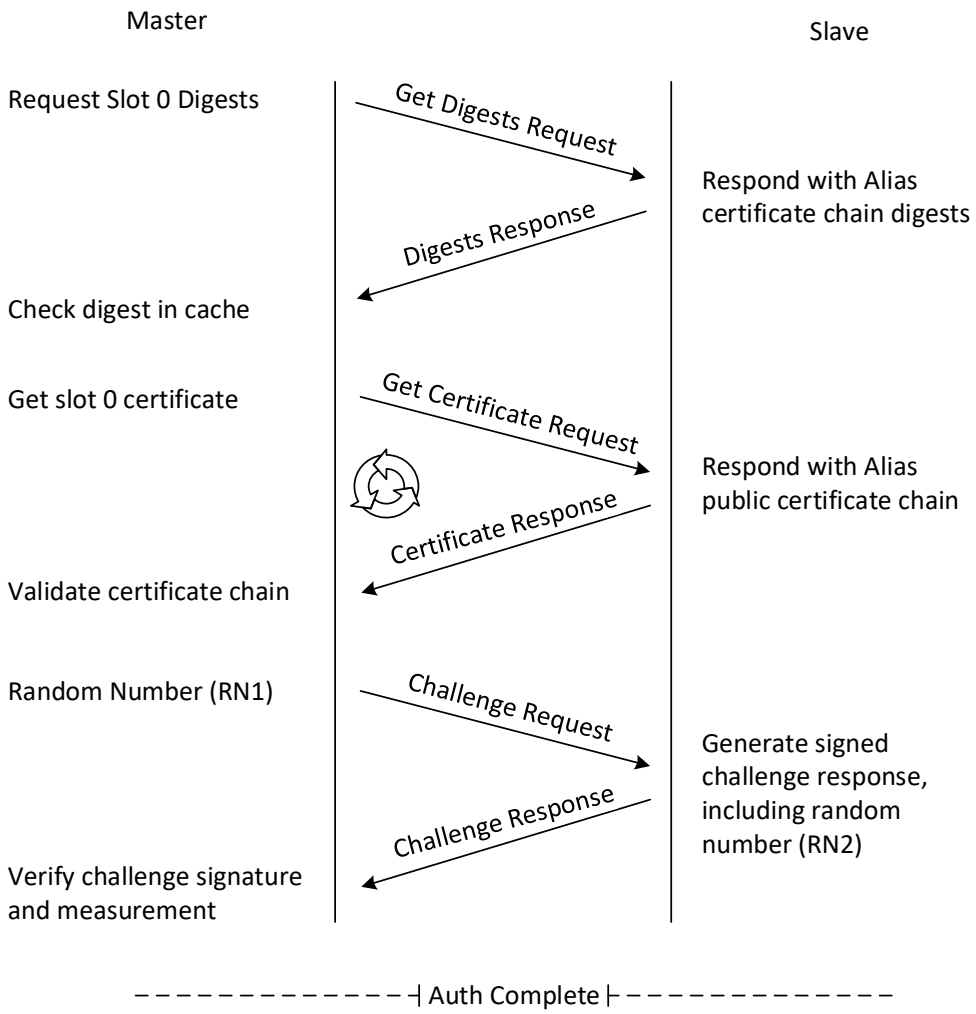
Once the certificate chain has been authenticated, the master will verify firmware integrity through the Challenge request. The slave will generate a Challenge response signed with its Alias key which includes the measurement of device firmware. The master can verify this measurement against a Component Firmware Manifest (CFM) or some other reference to confirm the firmware is valid.

5.1.1 Authentication Sequence

1. Master issues Get Digests command for Slot 0
2. Slave responds with digests for the Alias certificate chain
3. For each certificate in the chain, Master checks if the certificate has been cached
4. If the certificate has not been cached, Master issues Get Certificate request
5. Slave responds with the request certificate
6. Master verifies the Alias certificate chain against a trusted root CA
7. Master issues Challenge command containing a nonce (RN1)
8. Slave generates a response containing
    - Random nonce (RN2)
    - Collective firmware measurement (PMR0)
    - Signature over request/response pair using the Alias key

9. Master verifies the signature and firmware measurement

Figure 9 Authentication



5.2 Secure Session Establishment

Cerberus devices may optionally support secure session establishment to provide confidentiality between two devices or between external data center software and the device. Session establishment leverages the basic authentication flow with an additional key exchange to establish the secure session. Device authentication uses the identity certificate chain, while the session key exchange is based on ephemeral ECDH keys. The Device Capabilities command will determine if a device supports secure sessions.

After querying the for the device capabilities, the master will begin the authentication sequence by issuing a Get Digests request. When using authentication to establish a secure session, the Digests request will also indicate the type of key exchange that will be ultimately be used. If the device does not support the requested key exchange, it will respond with an error.

After authentication has completed, the master will generate an ephemeral key and send it to the slave. The slave will generate its own ephemeral session key and complete the key exchange. Session keys will be derived by running ECDH and subsequent KDFs over the session keys and nonces that were included in the challenge.

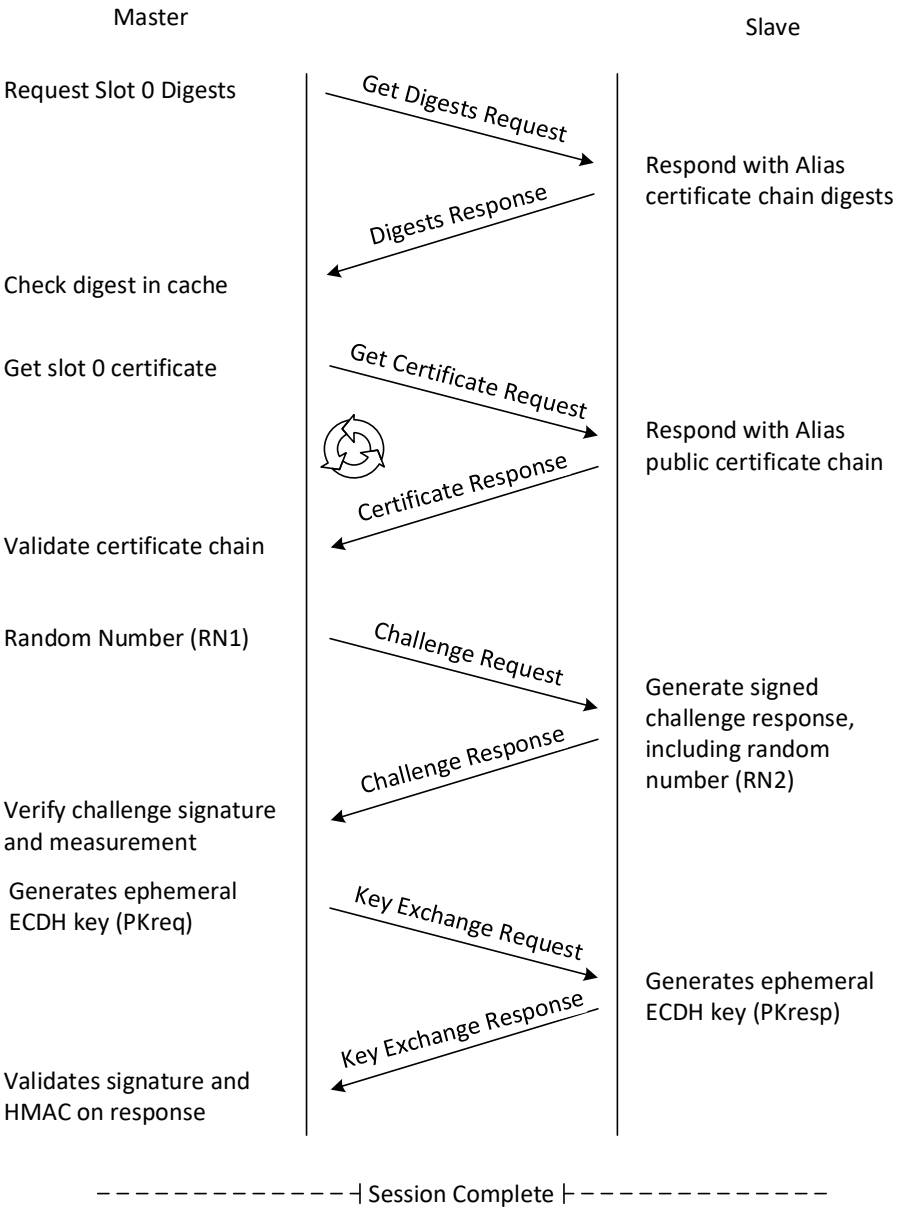
The KDF used for deriving session and HMAC keys is NIST SP800-108 Counter Mode. Session state is volatile. Each end of the session must be able to handle a loss of session context and support session re-establishment.

5.2.1 Session Establishment Sequence

The session establishment flow starts with standard authentication described is Section 5.1.1. Once the slave has been authenticated, the key exchange is used to establish the session.

1. Master generates an ephemeral ECC key pair and sends a Key Exchange request with Type 0 that includes the public key (PKreq).
2. Slave generates an ephemeral ECC key pair of equivalent strength (PKresp).
3. Slave generates a 256-bit AES session key (K_S) using NIST SP800-108 Counter Mode
    a. K_I = ECDH (PKreq, PKresp)
    b. Label = RN1
    c. Context = RN2
    d. r = 32
4. Slave generates a 256-bit MAC key (K_M) using NIST SP800-108 Counter Mode
    a. K_I = ECDH (PKreq, PKresp)
    b. Label = RN2
    c. Context = RN1
    d. r = 32
5. Slave sends a Key Exchange response that includes:
    a. The slave session public key (PKresp).
    b. A signature using the Alias key over PKreq and PKresp.
    c. An HMAC of its Alias key certificate using K_M.
6. Master generates the same session and MAC keys using the public key in the Key Exchange response.
7. Master validates the signature and HMAC on the response.
8. Messages can now be encrypted with 256-bit AES-GCM using the shared session key.

Figure 10 Session Establishment



5.3 Secure Device Binding

In some scenarios, it is necessary to have an additional level of authentication between the devices. This additional authentication mechanism is used to pair two devices such that replacement of either device would be detected. Detection of unauthorized part replacement is flagged as a security violation.

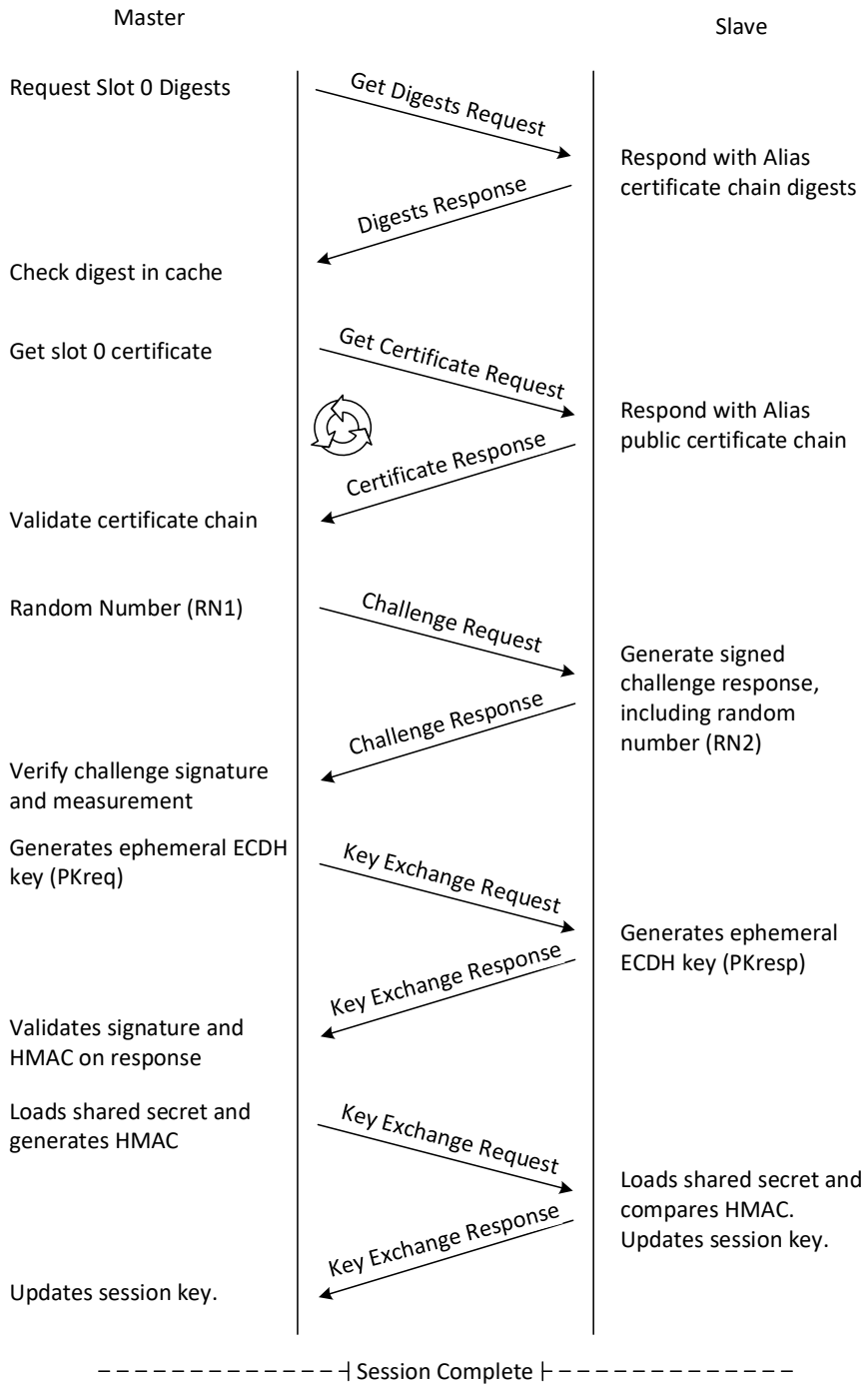
The binding comes from an additional KDF run on the session key to derive the final session key. Each device partaking in the tightly coupled pairing contain a common HMAC key that is used in a KDF to device the paired session key. This common HMAC key is derived from session data available the first time the two devices establish a secure session.

5.3.1 Secure Device Binding Sequence

A paired session with device binding begins with a standard secure session. Once the session has been established, an additional key exchange is used to update the session key. This additional key exchange must be encrypted.

1. Master loads the pairing key (K_P)
    a. If this is the first pairing request, the Master generates the key using NIST SP800-108 Counter Mode
        i. $K_I = K_S$
        ii. Label = pairing
        iii. No Context
        iv. r = 32
    b. If the Master has already been paired to the Slave, the Master loads key from secure storage.
2. Master generates a new session key ($K_{S'}$) using NIST SP800-108 Counter Mode.
    a. $K_I = K_P$
    b. Label = K_S
    c. No Context
    d. r = 32
3. Master sends a Key Exchange request with Type 1 that includes the length and HMAC of the pairing key using K_M. This message is encrypted with the current session key (K_S).
4. Slave loads K_P
    a. If this is the first paring request from this Master, the Slave generates the key using the same inputs as the Master.
    b. If the Slave has already paired with the Master, the Slave loads the key from secure storage.
5. Slave calculates the pairing key HMAC and compares it to the received data.
6. If this is the first paring request from this Master, Slave securely stores K_P.
7. Slave generates a new session key ($K_{S'}$).
8. Slave generates a Key Exchange response encrypted with $K_{S'}$.
9. Master verifies the response using $K_{S'}$. Failure to decrypt with $K_{S'}$ will require the Master to decrypt with K_S since the Slave will not update the session key on failure.
10. The secure session continues using $K_{S'}$. Messages encrypted with K_S will not be processed by either side.

Figure 11 Secure Device Binding



6 Command Format

The following section describes the MCTP message format to support the Authentication and Challenge and Attestation protocol. The Request/Response message body describes the Vendor Defined MCTP message encapsulated inside the MCTP transport. This section does not describe the MCTP Transport Header, which includes the MCTP Header Version, Destination Endpoint ID and other fields as defined by MCTP protocol. The MCTP message encapsulation is described in section 3.2

The MCTP Get Vendor Defined Message Support command enables discovery of what Endpoint vendor defined messages are supported. The discovery identifies the vendor organization and defined messages types. The format of this request is described in the MCTP base protocol specification.

For the Cerberus protocol, the following information shall be returned in response to the MCTP Get Vendor Defined Message Support request:

- Vendor ID Format = 0
- PCI Vendor ID = 0x1414
- Command Set Version = 4

6.1 Attestation Protocol Format

The messages from PA-RoT to AC-RoT will have the following fields

Field Name	Description
IC	(MCTP integrity check bit) Indicates whether the MCTP message is covered by an overall MCTP message payload integrity check
Message Type	Indicates MCTP Vendor defined message
MCTP PCI Vendor	Id for PCI Vendor. Cerberus messages use the Microsoft PCI ID of 0x1414.
Request Type	This field indicates what type of request is contained in the message. Messages defined in this specification shall have this bit set to 0. Setting this bit to 1 provides a mechanism, aside from different vendor IDs, to support a device-specific command set. Devices that don't have any additional command support will return an error if this bit is 1.
Crypt	Message Payload and Command are encrypted
Command	The command ID for command to execute
Msg Integrity Check	This field represents the optional presence of a message type-specific integrity check over the contents of the message body. If present (indicated by IC bit) the Message integrity check field is carried in the last bytes of the message body

Table 4 MCTP Message Format

+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
MCTP Rsvd				Header Version				Destination Endpoint ID								Source Endpoint ID								SOM	EOM	Pkt Seq #		TO	Msg Tag		
IC	Msg Type = 7E							MCTP PCI Vendor ID = 0x1414																Rq	Rsvd	Crypt	Reserved				
Command								Message Payload																							

The protocol header fields are to be included only in the first packet of a multiple packet MCTP message. After reconstruction of the message body, the protocol header will be used to interpret the message contents. Reserved fields must be set to 0.

6.1.1 Encrypted Messages

If the crypt field is set in the protocol header, the message body contains encrypted data. The command code byte and full message payload, reconstructed from individual MCTP packets, is encrypted. The session establishment flow described in section 5 is used to generate the encryption keys. An encrypted message will have a 16-byte GCM authentication tag and 12-byte initialization vector in plaintext at the end of the message body. The following table shows the body of an encrypted Cerberus message, with the encryption trailer. Segments shaded in grey indicate ciphertext, and white indicate plaintext.

Table 5 Encrypted Cerberus message body

IC	Msg Type = 7E	MCTP PCI Vendor ID = 0x1414	Rq	Rsvd	Crypt	Reserved
Command		Message Payload				
GCM Tag			Initialization Vector			

6.2 Command Set Type Code

The type codes associated with the commands determine whether the command can be executed outside of an obfuscated session:

Table 6 Command Types

Type	Description
1	Accepted inside or outside session.
2	Authentication and session setup commands.

3	Session required commands, obfuscated by session encryption or KDF, message body content is normally scrambled.
8xh	Any of the other command types, but the command uses the timeout allowed for Cryptographic commands.

6.3 RoT Commands

The following table describes the commands defined under this specification. There are three categories: (1) Required commands (R) that are mandatory for all implementations, (2) Optional commands (O) that may be utilized if the specific implementation requires it, (3) Master commands (M) that are required for all implementations that can act as an attestation master for slave devices. All MCTP commands are master initiated. The following section describes the command codes.

Table 7 Command List

Message Name	Type	Command	R/O/M	Description
ERROR	01h	7Fh	R	Status Response message.
Firmware Version	01h	01h	R	Retrieve firmware version information
Device Capabilities	01h	02h	R	Retrieves Device Capabilities
Device Id	01h	03h	R	Retrieves Device Id
Device Information	01h	04h	R	Retrieves device information
Export CSR	01h	20h	R	Exports CSR for device keys
Import Certificate	81h	21h	R	Imports CA signed Certificate
Get Certificate State	01h	22h	R	Checks the state of the signed Certificate chain
GET DIGESTS	82h	81h	R	PA-RoT retrieves session information
GET CERTIFICATE	02h	82h	R	PA-RoT sets session variables based on Session Query
CHALLENGE	82h	83h	R	PA-RoT retrieves and verifies AC-RoT certificate
Key Exchange	82h	84h	O[1]	Exchange pre-master session keys and mfg device pairing key
Session Sync	83h	85h	O[1]	Check status of a secure session
Get Log Info	01h	4Fh	O	Get Log Information
Get Log	01h	50h	O	Retrieve debug, attestation and tamper log
Clear Log	01h	51h	O	Clear log information
Get Attestation Data	01h	52h	O[2]	Retrieve raw data for an entry in the attestation log
Get Host State	01h	40h	O	Get reset state of the host processor
Get PFM Id	01h	59h	O	Get PFM Information
Get PFM Supported	01h	5Ah	O	Retrieve the PFM
Prepare PFM	01h	5Bh	O	Prepare PFM payload on PA-RoT
Update PFM	01h	5Ch	O	Set the PFM
Activate PFM	01h	5Dh	O	Force Activation of supplied PFM
Get CFM Id	01h	5Eh	M	Get Component Manifest Information
Prepare CFM	01h	5Fh	M	Prepare Component Manifest Update
Update CFM	01h	60h	M	Update Component Manifest
Activate CFM	01h	61h	M	Activate Component Firmware Manifest Update
Get CFM Supported	01h	8Dh	M	Retrieve supported CFM IDs
Get PCD Id	01h	62h	M	Get Platform Configuration Data Information

Prepare PCD	01h	63h	M	Prepare Platform Configuration Data Update
Update PCD	01h	64h	M	Update Platform Configuration Data
Activate PCD	01h	65h	M	Activate Platform Configuration Data Update
Prepare Firmware Update	01h	66h	O	Prepare for receiving firmware image
Update Firmware	01h	67h	O	Firmware update payload
Update Status	01h	68h	M[3]	Firmware, PFM/CFM/PCD update status
Extended Update Status	01h	8Eh	M[3]	Firmware, PFM/CFM/PCD extended status
Activate Firmware Update	01h	69h	O	Activate received FW update
Reset Configuration	81h	6Ah	O	Reset configuration to default state
Get Config IDs	81h	70h	M[4]	Get manifest IDs and signed digest of request nonce and response ids.
Recovery Firmware	01h	71h	O	Restore Firmware Index using backup.
Prepare Recovery Image	01h	72h	O	Prepare storage for Recovery Image
Update Recovery Image	01h	73h	O	Updates the Recover image
Activate Recovery Image	01h	74h	O	Activate the received Recovery image
Get Recovery Image Id	01h	75h	O	Get Recovery firmware information
Platform Measurement Register	81h	80h	O	Returns the Platform Measurement
Update Platform Measurement Register	83h	86h	O	Extends Platform Measurements
Reset Counter	01h	87h	R	Reset Counter
Unseal Message	81h	89h	O	Unseal attestation challenges.
Unseal Message Result	01h	8Ah	O	Get unsealing status and result
Unsupported Commands		F0h – FFh		Reserved commands that must be rejected by the device

[1] Key Exchange and Session Sync are Required if encrypted sessions are supported.
[2] Get Attestation Data is Required if an Attestation Log is supported.
[3] For implementations that support any update command, Update Status and Extended Update Status are Required
[4] For implementations that use PFM, CFM, or PCD configuration files, Get Config IDs is Required.

6.4 Message Body Structures

The following section describes the structures of the MCTP message body.

6.5 Error Message

The error command is returned for command responses when the command was not completed as proposed, it also acts as a generic status for commands without response whereby “No Error” code would indicate success. The Msg Tag, Seq and Command match the response to the corresponding request. The Message Body is returned as follows:

Table 8 Error Response

Payload	Description
1	Error Code
2:5	Error Data

Table 9 Error Codes

Error Code	Value	Description	Data
No Error	0h	Success [Reserved in USB Type C Authentication Specification]	00h
Invalid Request	01h	Invalidated data in the request	00h
Busy	03h	Device cannot response as it is busy processing other commands	00h
Unspecified	04h	Unspecified error occurred	Vendor defined
Reserved	05h-EFh	Reserved	Reserved
Invalid Checksum	F0h	Invalid checksum	Checksum
Out of Order Message	F1h	EOM before SOM	00h
Authentication	F2h	Authentication not established	00h
Out of Sequence Window	F3h	Message received out of Sequence Window	00h
Invalid Packet Length	F4h	Packet received with unexpected size	Packet Length
Message Overflow	F5h	Message exceeded maximum length	Message Length

If an explicit response is not defined for the command definitions in the following sections, the Error Message is the expected response with “No Error”. The Error Message can occur as the response for any command that fails processing.

6.6 Firmware Version

This command gets the target firmware the version.

Table 10 Firmware Version Request

Payload	Description
1	Area Index: 00h = Entire Firmware 01h = RIoT Core Additional indexes are firmware specific

Table 11 Firmware Version Response

Payload	Description
1:32	Firmware Version Number ASCII Formatted

6.7 Device Capabilities

Device Capabilities provides information on device functionality. The message and packet maximums are negotiated based on the shared capabilities. Some additional considerations apply when determining appropriate maximum sizes:

1. Per the MCTP specification, the packet payload size must be no less than 64 bytes.

2. Per section 3.5, the message payload size must be no more than 4096 bytes.
3. The total packet size supported by the device will include the encapsulation defined in section 3.2, excluding the first byte of destination address. For example, a maximum packet payload of 247 bytes will result in 256 bytes being transmitted for every packet: 1 byte destination address, 3 bytes SMBus header, 4 bytes MCTP header, 247 bytes payload, and 1 byte CRC-8.
4. Some commands do not support being separated across messages. Devices must support a maximum message size that is compatible with the expected payloads for supported commands.
5. Master devices, such as PA-RoTs, should support the maximum message size of 4096 bytes to ensure full compatibility with any slave device.

Table 12 Device Capabilities Request

Payload	Description
1:2	Maximum Message Payload Size
3:4	Maximum Packet Payload Size
5	Mode: [7:6] 00 = AC-RoT 01 = PA-RoT 10 = External 11 = Reserved [5:4] Master/Slave 00 = Unknown 01 = Master 10 = Slave 11 = both master and slave [3] Reserved [2:0] Security 000 = None 001 = Hash/KDF 010 = Authentication [Certificate Auth] 100 = Confidentiality [AES]
6	[7] PFM support [6] Policy Support [5] Firmware Protection [4-0] Reserved
7	PK Key Strength: [7] RSA [6] ECDSA [5:3] ECC 000: None 001: 160bit 010: 256bit 100: Reserved [2:0] RSA: 000: None

	001: RSA 2048 010: RSA 3072 100: RSA 4096
8	Encryption Key Strength: [7] ECC [6:3] Reserved [2:0] AES: 000: None 001: 128 bit 010: 256 bit 100: 384 bit

Table 13 Device Capabilities Response

Payload	Description
1:2	Maximum Message Payload Size
3:4	Maximum Packet Payload Size
5	Mode: [7:6] 00 = AC-RoT 01 = PA-RoT 10 = External 11 = Reserved [5:4] Master/Slave 00 = Unknown 01 = Master 10 = Slave 11 = both master and slave [3] Reserved [2:0] Security 000 = None 001 = Hash/KDF 010 = Authentication [Certificate Auth] 100 = Confidentiality [AES]
6	[7] PFM support [6] Policy Support [5] Firmware Protection [4-0] Reserved
7	PK Key Strength: [7] RSA [6] ECDSA [5:3] ECC 000: None 001: 160bit 010: 256bit 100: Reserved [2:0] RSA: 000: None

	001: RSA 2048 010: RSA 3072 100: RSA 4096
8	Encryption Key Strength: [7] ECC [6:3] Reserved [2:0] AES: 000: None 001: 128 bit 010: 256 bit 100: 384 bit
9	Maximum Message timeout: multiple of 10ms
10	Maximum Cryptographic Message timeout: multiple of 100ms

6.8 Device Id

Eight bytes response.

Table 14 Device Id Request

Payload	Description

Table 15 Device Id Response

Payload	Description
1:2	Vendor ID; LSB
3:4	Device ID; LSB
5:6	Subsystem Vendor ID; LSB
7:8	Subsystem ID; LSB

6.9 Device Information

This command gets information about the target device.

Table 16 Device Information Request

Payload	Description
1	Information Index: 00h = Unique Chip Identifier Additional indexes are firmware specific

Table 17 Device Information Response

Payload	Description
1:N	Requested information in binary format

6.10 Export CSR

Exports the Device Identification Certificate Self Signed Certificate Signing Request. The initial Certificate is self-signed, until CA signed and imported. Once the CA signed version of the certificate has been imported to the device, the self-signed certificate is replaced.

Table 18 Export CSR Request

Payload	Description
1	Index: Default = 0

Table 19 Export CSR Response

Payload	Description
1:N	Certificate

6.11 Import Certificate

Imports the signed Device Identification certificate chain into the device. Each import command sends a single certificate in the chain to the device, and there is no requirement on the order in which these certificates must be imported. Upon verification of a complete chain, the device is sealed, and no further imports can occur without changes to firmware. A response message is returned before certificate chain validation has been performed with the new certificate and only indicates if the certificate was accepted by the device. The complete state of certificate provisioning can be queried with the Get Certificate State request.

Upon receiving a certificate, the device will start to authenticate the stored certificate chain. When sending multiple certificates, it may be necessary to ensure the previous authentication step has completed before sending a new certificate. The authentication status can be checked with the Get Certificate State command.

Table 20 Import Certificate Request

Payload	Description
1	Index: 0 = Device Identification Certificate 1 = Root CA Certificate 2 = Intermediate CA Certificate Additional certificate indices are implementation specific.
2:3	Certificate Length
4:N	Certificate

6.12 Get Certificate State

Determine the state of the certificate chain for signed certificates that have been sent to the device. The request for this command contains no additional payload.

Table 21 Get Certificate State Request

Payload	Description

Table 22 Get Certificate State Response

Payload	Description
1	State: 0 = A valid chain has been provisioned. 1 = A valid chain has not been provisioned. 2 = The stored chain is being validated.
2:4	Error details if chain validation has failed.

6.13 GET DIGESTS

This command is derived from the similar USB Type C-Authentication Message. The Protocol Version byte is not included, the Message Type is present in the Command byte of the MCTP message. See: Table 4 MCTP Message Format. The response is different, since certificate handling deviates from USB Type C, and the reserved bytes are repurposed.

This command can be sent at any time. If authentication was previously established, this command will renegotiate/override the previous authentication and session establishment. The byte 2, reserved USB Type C – authentication specification, is used to describe the key exchange algorithm. This is relevant when the requester and responder support multiple key exchange algorithms.

Slots that do not contain a valid certificate chain will generate a response with 0 digests. Payload byte 2 will indicate that no digests are returned.

Table 23 GET DIGEST Request

Payload	Description
1	Param1: Slot Number of the target Certificate Chain to read. The value should be 0-7.
2	Key Exchange Algorithm: 0 = None 1 = ECDH

Table 24 GET DIGEST Response

Payload	Description
1	Capabilities Field; shall be set to 01
2	The number of certificate digests returned. Each digest represents a single certificate in the chain, starting from the certificate closest to the root.
3:N	Digest[0] 32 byte SHA256 digest of the root Certificate in the Chain
N+	Digest[1] 32 byte SHA256 digest of N Certificate in the Chain

6.14 GET CERTIFICATE

This command retrieves the public attestation certificate chain for the AC-RoT. It follows closely the USB Type C Authentication Specification.

If the device does not have a certificate for the requested slot or index, the certificate contents in the response will be empty.

Table 25 GET CERTIFICATE Request

Payload	Description
1	Param1: Slot Number of the target Certificate Chain to read. The value should be 0-7.
2	Certificate number. This a 0-based index starting with the root certificate in the chain.
3:4	Offset: offset in bytes from start of the Certificate chain where read request begins.
5:6	Length: number of bytes to read

Table 26 GET CERTIFICATE Response

Payload	Description
1	Param1: Slot Number of the target Certificate Chain returned.
2	Certificate number of the returned certificate
3:N	Requested contents of target Certificate Chain. See section 4 Certificates.

6.15 CHALLENGE

The PA-RoT will send this command providing the first nonce in the key exchange.

Table 27 CHALLENGE Request

Payload	Description
1	Slot number of the recipient's Certificate Chain that will be used for Authentication. The value should be 0-7.
2	Reserved
3:35	Random 32 byte nonce chosen by PA-RoT

Table 28 CHALLENGE Response

Payload	Description
1	Shall contain the Slot number in the Param1 field of the corresponding CHALLENGE Request
2	Certificate slot mask
3	MinProtocolVersion supported by device
4	MaxProtocolVersion supported by device
5:6	Reserved
7:38	Random number chosen by AC-RoT (^{RN2})

39	Number of components used to generate the PMR0 measurement
40	Length of each digest in PMR0 (L)
41:40+L	Value of Platform Measurement Register 0 (Aggregated Firmware Digest)
41+L:N	Signature of combined request and response message payloads. See USB Type C Authentication Protocol for details of request/response signature.

The firmware digests are measurements of the security descriptors and the firmware of the target components. This firmware measurement data does not include Cerberus PCD, CFM or PFM. These measurements are returned in PMR1. The numbers are retrieved in the 6.44 Get Configuration Ids. The USB-C Context Hash is not included in the CHALLENGE. This is replaced with contextual measurements for the device. Note: The attestation Certificate derivation will include the measurement of firmware and security descriptors. PMR0 is anticipated to be the least changing PMR as it contains the measurement of security descriptors and device initial boot loader.

6.16 Key Exchange

Key Exchange is used to establish an encrypted channel with a device. The session establishment flow is detailed in Section 5 Authentication.

Upon receiving this message with Key Type 0, both sides can establish an encrypted session. If Key Type 1, the paired key is also compared, and paired functionality is unlocked if verified to match the expected key. The Key Type 1 can only be performed under an encrypted session, as the session key is required for the HMAC. When initially calculating a pairing key (K_P) for device binding, the HMAC specified in the Key Type 0 request for the session will be used with the KDF.

When closing an established session (Key Type 2), the response will be transmitted in plain text if the session was successfully terminated.

Table 29 Key Exchange Request

Payload	Description
1	Key Type: 0 = Session Key 1 = Paired Key HMAC 2 = Delete Session Key (close session)
2:N	Key data. Format is defined by the type of request.

Table 30 Key Exchange Response

Payload	Description
1	Key Type: 0 = Session Key 1 = Paired Key HMAC
2:N	Response data. Format is defined by the type of request.

Table 31 Key Exchange Type 0 Request Data

Payload	Description
1	HMAC Type: 0 = SHA256 1 = SHA384 2 = SHA512 The HMAC type specified in this message applies to all HMAC operations for the established session, including any subsequent pairing messages. Since session keys (K_S and K_M) are 256-bit keys, they will always be generated using SHA256 regardless of the type of HMAC used for key exchange messages.
2:N	ASN.1 DER encoded ECC public key (PKreq)

Table 32 Key Exchange Type 0 Response Data

Payload	Description
1	Reserved. Set to 0.
2:3	Key Length
4:N	ASN.1 DER encoded ECC public key (PKresp)
N+1:N+2	Signature Length
N+3:M	Signature using Alias Key over ephemeral session keys: $SGN^{(Alias)}$(PKreq || PKresp)
M+1:M+2	HMAC Length
M+3:H	HMAC of the Alias Key certificate: HMAC (K_M, ASN.1 DER encoded Alias Certificate)

Table 33 Key Exchange Type 1 Request Data

Payload	Description
1:2	Length in bytes of the pairing key
3:N	HMAC of the pairing key: HMAC (K_M, K_P)

Table 34 Key Exchange Type 1 Response Data

Payload	Description

Table 35 Key Exchange Type 2 Request Data

Payload	Description
1:N	HMAC of session key: HMAC (K_M, K_S)

Table 36 Key Exchange Type 2 Response Data

Payload	Description

6.17 Session Sync

Check the state of an encrypted session. The message must always be encrypted.

Table 37 Session Sync Request

Payload	Description
1:4	Random number (RNreq)

Table 38 Session Sync Response

Payload	Description
1:N	HMAC of the request number: HMAC (K_M, RNreq)

6.18 Get Log Info

Get the internal log information for the RoT.

Table 39 Get Log Info Request

Payload	Description

Table 40 Get Log Info Response

Payload	Description
1:4	Debug Log (01h) Length in bytes
5:8	Attestation Log (02h) Length in bytes
9:12	Tamper Log (03h) Length in bytes

6.19 Get Log

Get the internal log for the RoT. There are 3 types of logs available: The Debug Log, which contains Cerberus application information and machine state. The Attestation measurement log, this log format is like the TCG log, and the Tamper log. It is not possible to clear or reset the tamper counter.

Table 41 Log Types

Log Type	Description
1	Debug Log
2	Attestation Log

3	Tamper Log

Table 42 Get Log Section Request

Payload	Description
1	Log Type
2:5	Offset

Table 43 Get Debug/Attestation Log Response

Payload	Description
1:N	The contents of the log

Length determined by end of log, or packet size based on device capabilities see section: 6.7 Device Capabilities. If response spans multiple MCTP messages, end of response will be determined by an MCTP message which has a payload less than maximum payload supported by both devices. To guarantee a response will never fall exactly on the max payload boundary, the responder must send back an extra packet with zero payload.

6.19.1 Attestation Log Format

The attestation log will report individual measurements for all components of each PMR. Each measurement will be a single entry in the log. The entire log consists of each entry sequentially concatenated. The structure of the entry closely resembles TCG events. See TCG PC Client Platform Firmware Profile Specification.

Table 44 Log Entry Header

Offset	Description
1	Log entry start marker: [7:4]: 0xC [3:0]: Header format, 0xB per this specification.
2:3	Total length of the entry, including the header
4:7	Unique entry identifier

Table 45 Attestation Entry Format

Offset	Description
1:7	Log Entry Header
8:11	TCG Event Type
12	Measurement index within a single PMR
13	Index of the PMR for the measurement
14:15	Reserved, set to 0
16	Number of digests (1)
17:19	Reserved, set to 0
20:21	Digest algorithm Id (0x0B, SHA256)

22:53	SHA256 digest used to extend the measurement
54:57	Measurement size (32)
58:89	Measurement

6.19.2 Debug Log Format

The debug log reported by the device has no specified format, as this can vary between different devices and it is not necessary for attestation. It is expected that diagnostic utilities for the device will be able to understand the exposed log information. A recommended entry format is provided here.

Table 46 Debug Entry Format

Offset	Description
1:7	Log Entry Header
8:9	Format of the entry, currently 1
10	Severity of the entry
11	Identifier for the component that generated the message
12	Identifier for the entry message
13:16	Message specific argument
17:20	Message specific argument

6.20 Clear Debug/Attestation Log

Clear the log in the RoT.

Table 47 Clear Debug/Attestation Log Request

Payload	Description
1	Type: 01 or 02

Note: in clearing the attestation log, it is automatically recreated using current measurements.

6.21 Get Attestation Data

Get the raw data used to generate measurements reported in the attestation log. Not all measurements will have raw data available, as this is mostly intended for measurements generated over small amounts of data or text. Requests for entries that don't provide the measured data will generate a normal response with an empty payload. Requests for invalid entries will generate an error response.

While this command is intended to support small pieces of that data that should fit into a single MCTP message, an offset parameter is included in the request to support scenarios where the required data is too large.

Table 48 Get Attestation Data Request

Payload	Description
1	Platform Measurement Register
2	Entry Index

3:6	Offset

Table 49 Get Attestation Data Response

Payload	Description
1:N	The measured data

6.22 Get Host State

Retrieve the reset state of the host processor being protected by Cerberus.

Table 50 Get Host State Request

Payload	Description
1	Port Id

Table 51 Get Host State Response

Payload	Description
1	Host Reset State: 00h – Host is running (out of reset) 01h – Host is being held in reset 02h – Host is not being held in reset, but is not running

6.23 Get Platform Firmware Manifest Id

Retrieves PFM identifiers.

Table 52 PFM Information Request

Payload	Description
1	Port Id
2	PFM Region: 0 = Active 1 = Pending
3 (optional)	Identifier: 0 = Version Id (default) 1 = Platform Id

Table 53 PFM Version Id Response

Payload	Description
1	PFM Valid (0 or 1)
2:5	PFM Version Id

Table 54 PFM Platform Id Response

Payload	Description

1	PFM Valid (0 or 1)
2:N	PFM Platform Id as null-terminated ASCII

6.24 Get Platform Firmware Manifest Supported Firmware

Table 55 PFM Supported Firmware Request

Payload	Description
1	Port Id
2	PFM Region: 0 = Active 1 = Pending
3:6	Offset

Table 56 Supported Firmware Response

Payload	Description
1	PFM Valid (0 or 1)
2:5	PFM Version Id
6:N	PFM supported FW versions

If response spans multiple MCTP messages, end of response will be determined by an MCTP packet which has payload less than maximum payload supported by both devices. To guarantee a response will never fall exactly on the max payload boundary, the responder should send back an extra packet with zero payload.

6.25 Prepare Platform Firmware Manifest

Provisions RoT for incoming PFM.

Table 57 Prepare PFM Request

Payload	Description
1	Port Id
2:5	Total size

6.26 Update Platform Firmware Manifest

The flash descriptor structure describes the regions of flash for the device.

Table 58 Update PFM Request

Payload	Description
1	Port Id
2:N	PFM Payload

PFM payload includes PFM signature and monotonic forward only Id. PFM signature is verified upon receipt of all PFM payloads. PFMs are activated upon the activation command. Note if a system is rebooted after receiving a PFM, the PFM is atomically activated. To activate before reboot, issue the Activate PFM command.

6.27 Activate Platform Firmware Manifest

Upon valid PFM update, the update command seals the PFM committal method. If committing immediately, flash reads and writes should be suspended when this command is issued. The RoT will master the SPI bus and verify the newly updated PFM. This command can only follow a valid PFM update.

Table 59 Update PFM Request

Payload	Description
1	Port Id
2	Activation: 0 = Reboot only 1 = Immediately

If reboot only has been issued, the option for “Immediately” committing the PFM is not available until a new PFM is updated.

6.28 Get Component Firmware Manifest Id

Retrieves the Component Firmware Manifest Id

Table 60 Get CFM Id Request

Payload	Description
1	CFM Region: 0 = Active 1 = Pending
2 (optional)	Identifier: 0 = Version Id (default) 1 = Platform Id

Table 61 CFM Version Id Response

Payload	Description
1	CFM Valid (0 or 1)
2:5	CFM Version Id

Table 62 CFM Platform Id Response

Payload	Description
1	CFM Valid (0 or 1)
2:N	CFM Platform Id as null-terminated ASCII

6.29 Prepare Component Firmware Manifest

Provisions RoT for incoming Component Firmware Manifest.

Payload	Description
1:4	Total size

6.30 Update Component Firmware Manifest

The flash descriptor structure describes the regions of flash for the device.

Table 63 Update Component Firmware Manifest Request

Payload	Description
1:N	Component Firmware Manifest Payload

The CFM payload includes CFM signature and monotonic forward only Id. CFM signature is verified upon receipt of all CFM payloads. CFMs are activated upon the activation command. Note if a system is rebooted after receiving a CFM, the pending CFM is verified and atomically activated. To activate before reboot, issue the Activate CFM command.

6.31 Activate Component Firmware Manifest

Upon valid CFM update, the update command seals the CFM committal method. The RoT will master I2C and attest Components in the Platform Configuration Data against the CFM.

Table 64 Active CFM Request

Payload	Description
1	Activation: 0 = Reboot only 1 = Immediately

6.32 Get Component Firmware Manifest Component IDs

CFM Supported component IDs Request

Payload	Description
1	CFM Region: 0 = Active 1 = Pending
2:5	Offset

CFM Supported component IDs Response

Payload	Description
1	CFM Valid (0 or 1)

2:5	CFM Version Id
6:N	CFM supported component IDs

If response spans multiple MCTP messages, end of response will be determined by an MCTP packet which has payload less than maximum payload supported by both devices. To guarantee a response will never fall exactly on the max payload boundary, the responder should send back an extra packet with zero payload.

6.33 Get Platform Configuration Data Id

Retrieves the PCD Id

Table 65 Get Platform Configuration Data Request

Payload	Description
1 (optional)	Identifier: 0 = Version Id (default) 1 = Platform Id

Table 66 Get Platform Configuration Data Version Id Response

Payload	Description
1	PCD Valid (0 or 1)
2:5	PCD Version Id

Table 67 Get Platform Configuration Data Platform Id Response

Payload	Description
1	PCD Valid (0 or 1)
2:N	PCD Platform Id as null-terminated ASCII

6.34 Prepare Platform Configuration Data

Provisions RoT for incoming Platform Configuration Data.

Payload	Description
1:4	Total size

6.35 Update Platform Configuration Data

The flash descriptor structure describes the regions of flash for the device.

Table 68 Update Platform Configuration Data Request

Payload	Description
1:N	PCD Payload

The PCD payload includes PCD signature and monotonic forward only Id. PCD signature is verified upon receipt of all PCD payloads. PCD is activated upon the activation command. Note if a system is rebooted after receiving a PCD.

6.36 Activate Platform Configuration Data

Upon valid PCD update, the activate command seals the PCD committal.

Table 69 Active PCD Request

Payload	Description

6.37 Platform Configuration

The following table describes the Platform Configuration Data Structure

Table 70 PCD Structure

<table>
<tr><th>Payload</th><th>Description</th></tr>
<tr><td>1:3</td><td>Platform Configuration Data Id</td></tr>
<tr><td>4:5</td><td>Length</td></tr>
<tr><td>6</td><td>Policy Count</td></tr>
<tr><td>7:N</td><td>Each AC-RoT has 1 entry. The Configuration Data determines the feature enablement and attestation
<table>
<tr><td>Byte</td><td>Description</td></tr>
<tr><td>1</td><td>Device Id</td></tr>
<tr><td>4</td><td>Channel</td></tr>
<tr><td>5</td><td>Slave Address</td></tr>
<tr><td>6</td><td>[7:5] Threshold Count
[4] Power Control
0 = Disabled
1 = Enabled
[3] Debug Enabled
0 = Disabled
1 = Enabled
[2] Auto Recovery
0 = Disabled
1 = Enabled
[1] Policy Active
0 = Disabled
1 = Enabled
[0] Threshold Active
0 = Disabled
1 = Enabled</td></tr>
<tr><td>7</td><td>Power Ctrl Index</td></tr>
<tr><td>8</td><td>Failure Action</td></tr>
</table></td></tr>
</table>

N:N	Signature of payload

The Power Control Index informs the PA-RoT of the index assigned to power sequence the Component. This informs the PA-RoT which control register needs to be asserted in the platform power sequencer.

The Failure Action: 0 = Platform Defined, 1 = Report Only, 2 = Auto Recover 3 = Power Control.

6.38 Prepare Firmware Update

Provisions RoT for incoming firmware update.

Table 71 Prepare Firmware Update

Payload	Description
1:4	Total size

6.39 Update Firmware

The flash descriptor structure describes the regions of flash for the device.

Table 72 Update Firmware Request

Payload	Description
1:N	Firmware Update payload, header signature. See firmware update specification.

6.40 Update Status

The Update Status reports the update payload status. The update status will be status for the last operation that was requested. This status will remain the same until another operation is performed or Cerberus is reset.

Table 73 Update Status Request

Payload	Description
1	Update Type 00 = Firmware 01 = Platform Firmware Manifest 02 = Component Firmware Manifest 03 = Platform Configuration Data 04 = Host Firmware 05 = Recovery Firmware 06 = Reset Configuration
2	Port Id

Table 74 Update Status Response

Payload	Description

1:4	Update Status. See firmware update specification for details.

6.41 Extended Update Status

The Extended Update Status reports the update payload status along with the remaining number of update bytes expected. The update status will be status for the last operation that was requested. This status will remain the same until another operation is performed or Cerberus is reset.

Table 75 Extended Update Status Request

Payload	Description
1	Update Type 00 = Firmware 01 = Platform Firmware Manifest 02 = Component Firmware Manifest 03 = Configuration Data 04 = Host Firmware 05 = Recovery Firmware 06 = Reset Configuration
2	Port Id

Table 76 Extended Update Status Response

Payload	Description
1:4	Update Status. See firmware update specification for details.
5:8	Expected update bytes remaining.

6.42 Activate Firmware Update

Alerts Cerberus that sending of update bytes is complete, and that verification of update should start. This command has no payload, the ERROR response zero is expected.

Table 77 Activate Firmware Update

Payload	Description

6.43 Reset Configuration

Resets configuration parameters back to the default state. Depending on the request parameters, different amounts of types of configuration can be erased, and each type of configuration may require different levels of authorization to complete.

If authorization is required for the operation to complete, the response will contain a device-specific, one-time use, authorization token that must be signed with the PFM key to unlock the operation. The authorization token has the following behavior:

1. A request for the same operation without providing the signed authorization token will generate a new token that invalidates any old token. This is true even if the old token has not been used yet.
2. After an authorization token has been used to unlock an operation, it can never be used again. A new token must be requested. This is true even if the requested operation was not able to complete successfully.
3. A failure to authorize the request when providing a signed token does not invalidate the current authorization token in the device.

If authorization is not required, or the request is sent with a signed token, a standard error response will be returned indicating the status.

Table 78 Reset Configuration Request

Payload	Description
1	Type of reset operation to request: 0: Revert the device into the unprotected (bypass) state by erasing all PFMs and CFMs. 1: Perform a factory reset by removing all configuration. This does not include signed device certificates.
2:N	(Optional) Device-specific authorization token, signed with PFM key.

Table 79 Reset Configuration Response with Authorization Token

Payload	Description
1:N	Device-specific authorization token

6.44 Get Configuration Ids

This command retrieves PFM Ids, CFM Ids, PCD Id, and signed digest of request nonce and response ids.

Table 80 Get Configuration Ids Request

Payload	Description
1:32	32 byte Nonce

Table 81 Get Configuration Ids Response

Payload	Description
1:32	32 byte Nonce
33	Number of PFMs Ids (P)
34	Number of CFM Ids (C)
35:(P*4 + C*4 + 4) (V')	PFM Version Id[0] - PFM Version Id[N] CFM Version Id[0] - CFM Version Id[N] PCD Version Id
V'+1:M	PFM Platform Id[0] - PFM Platform Id[N], each null terminated CFM Platform Id[0] - CFM Platform Id[N], each null terminated

	PCD Platform Id, null terminated
M+1:SGN	$SGN^{(pk)}$(request message nonce + response message payload)

N is the number of measurements and L is the length of each measurement. The Signature should be a SHA2 over the request and response message body (excluding the signature). The signature algorithm is defined by the certificate exchanged in the DIGEST.

6.45 Recover Firmware

Start the firmware recovery process for the device. Not all devices will support all types of recovery. The implementation is device specific.

Table 82 Recover Firmware Request

Payload	Description
1	Port Id
2	Firmware image to use for recovery: 0: Exit Recovery 1: Enter Recovery

6.46 Prepare Recovery Image

Provisions RoT for incoming Recovery Image for Port.

Table 83 Prepare Recovery Image Request

Payload	Description
1	Port Id
2:5	Total size

The response back is the Error Code indicating Success or failure.

6.47 Update Recovery Image

The flash descriptor structure describes the regions of flash for the device.

Table 84 Update Component Firmware Manifest Request

Payload	Description
1	Port Id
2:N	Recovery Image Payload

6.48 Activate Recovery Image

Signals recovery image has been completely sent and verification of the image should start. Once the image has been verified, it can be used for host firmware recovery.

Table 85 Activate Recovery Image

Payload	Description
1	Port Id

6.49 Get Recovery Image Id

Retrieves the recovery image identifiers.

Table 86 Recovery Image Id Request

Payload	Description
1	Port Id
2 (optional)	Identifier: 0 = Version Id (default) 1 = Platform Id

Table 87 Recovery Image Version Id Response

Payload	Description
1:32	Recovery Image Version Id

Table 88 Recovery Image Platform Id Response

Payload	Description
1:N	Recovery Image Platform Id as null-terminated ASCII

6.50 Platform Measurement Register

Returns the Cerberus Platform Measurement Register (PMR), which is a digest of the Cerberus Firmware, PFM, CFM and PCD. This information contained in PMR0 is Cerberus firmware. PMR1-2 are reserved for PFM/CFM/PCD and other configuration. PMR3-4 are reserved for external usages.

Attestation requires that at least PMR0 be maintained, since that is the measurement reported in the challenge response message. At a minimum, PMR0 should contain any security configuration of the device and all firmware components. This includes any firmware actively running as well as firmware that was run to boot the device. For ease of attestation, PMR0 is intended to only contain static information. If variable data will also be measured, these measurements should be exposed through PMR1-2.

Table 89 Platform Measurement Request

Payload	Description
1	Platform Measurement Number
2:33	32 byte Nonce

Table 90 Platform Measurement Response

Payload	Description
1:32	32 byte Nonce

33	Measurement length (L)
34:33+L	Platform Measurement Value
34+L:N	$SGN^{(pk)}$(request message payload + response message payload)

PMR1-4 are cleared on component reset. PMR0 is cleared and re-built on Cerberus reset.

6.51 Update Platform Measurement Register

External updates to PMR3-4 are permitted. Attempts to update PMR0-2 will result error. Only SHA 2 is supported for measurement extension. SHA1 and SHA3 are not applicable. Note: The measurement can only be updated over an authenticated and secured channel.

Table 91 Update Platform Measurement Request

Payload	Description
1	Platform Measurement Number
2:N	Measurement Extension

6.52 Reset Counter

Provides Cerberus and Component Reset Counter since power-on.

Table 92 Reset Counter Request

Payload	Description
1	Reset Counter Type 0 = Local Device 1 = Protected External Devices (if applicable). These does not include external AC-RoTs that are challenged by the device. Other values are implementation specific.
2	Port Id

Table 93 Reset Counter Response

Payload	Description
1:2	Reset Count

6.53 Message Unseal

This command starts unsealing an attestation message. The ciphertext is limited to what can fit in a single message along with the other pieces necessary for unsealing.

Table 94 Unseal Message Request

Payload	Description
1	[7:5] Reserved

	[4:2] HMAC Type: 000 – SHA256 [1:0] Seed Type: 00 – RSA: Seed is encrypted with an RSA public key 01 – ECDH: Seed is an ECC public key, ASN.1/DER encoded
2	Additional Seed Parameters RSA: [7:3] Reserved [2:0] Padding Scheme: 000 – PKCS#1 v1.5 001 – OAEP using SHA1 010 – OAEP using SHA256 ECDH: [7:1] Reserved [0]: Seed Processing: 0 – No additional processing. Raw ECDH output is the seed. 1 – Seed is a SHA256 hash of the ECDH output.
3:4	Seed Length (S)
5:4+S (S')	Seed
S'+1:S'+2	Cipher Text Length (C)
S'+3:S'+2+C (C')	Cipher Text
C'+1:C'+2	HMAC Length (H)
C'+3:C'+2+H (H')	HMAC
H'+1:H'+64 (P0')	PMR0 Sealing, 0's to ignore. Unused bytes are first and must be set to 0.
P0'+1:P0'+64 (P1')	PMR1 Sealing, 0's to ignore. Unused bytes are first and must be set to 0.
P1'+1:P1'+64 (P2')	PMR2 Sealing, 0's to ignore. Unused bytes are first and must be set to 0.
P2'+1:P2'+64 (P3')	PMR3 Sealing, 0's to ignore. Unused bytes are first and must be set to 0.
P3'+1:P3'+64	PMR4 Sealing, 0's to ignore. Unused bytes are first and must be set to 0.

6.54 Message Unseal Result

This command retrieves the current status of an unsealing process.

Table 95 Unseal Message Request

Payload	Description

Table 96 Unseal Message Pending Response

Payload	Description
1:4	Unsealing status

Table 97 Unseal Message Completed Response

Payload	Description
1:4	Unsealing status
5:6	Encryption Key Length
7:N	Encryption Key

The Seal/Unseal flow is described in the Cerberus Attestation Integration specification.

7 Platform Active RoT (PA-RoT)

The PA-RoT is responsible for challenging the AC-RoT's and collecting their firmware measurements. The PA-RoT retains a private manifest of active components that includes addresses, buses, firmware versions, digests and firmware topologies.

The manifest informs the PA-RoT on all the Active Components in the system. It provides their I2C addresses, and information on how to verify their measurements against a known or expected state. Polices configured in the Platform RoT determine what action it should take should the measurements fail verification.

In the Cerberus designed motherboard, the PA-RoT orchestrates power-on. Only Active Components listed in the challenge manifest, that pass verification will be released from power-on reset.

7.1 Platform Firmware Manifest (PFM) and Component Firmware Manifest

The PA-RoT contains a Platform Firmware Manifest (PFM) that describes the firmware permitted on the Platform. The Component Firmware Manifest (CFM) describes the firmware permitted for components in the Platform. The Platform Configuration Data (PCD), specific to each SKU describes the number of Component types in the platform and their respective locations.

Note: The PFM and CFM are different from the boot key manifest described in the Processor Secure Boot Requirements specification. The PFM and CFM describe firmware permitted to run in the system across Platform and Active Components. The CFM is a complement to the PFM, generated by the PA-RoT for the measurement comparison of components in the system. This complement is as the Reported Firmware Manifest (RFM), which is like the TCG log. The PFM and RFM are stored encrypted in the PA-RoT. The symmetric encryption key for the PA-RoT is hardware generated and unique to each microcontroller. The symmetric key in the PA-Rot is not exportable or firmware readable; and only accessible to the crypto engine for encryption/decryption. The AES Galois/Counter Mode (GCM) encryption a unique auditable tag to any changes to the manifest at both an application level and persistent storage level.

The following table lists the attributes stored in the PFM for each Active component:

Table 98 PFM Attributes

Attribute	Description
Description	Device Part or Description
Device Type	Underlying Device Type of AC-RoT
Remediation Policy	Policy(s) defining default remediation actions for integrity failure.
Firmware Version	List of firmware versions
Flash Areas/Offsets	List of offset and digests, used and unused
Measurement	Firmware Measurements
Measurement Algorithm	Algorithm used to calculate measurement.
Public Key	Public keys in the key manifest
Digest Algorithm	Algorithm used to calculate
Signature	Firmware signature(s)

The PA-RoT actively takes measurements of flash from platform firmware, the PFM provides metadata that instructs the RoT on measurement and signature verification. The PA-RoT stores the measurements in the RFM. The PA-Rot then challenges the AC-RoTs for their measurements using the Platform Configuration Data. it compares measurements from the AC-RoT's to the CFM, while recording measurements in the RFM.

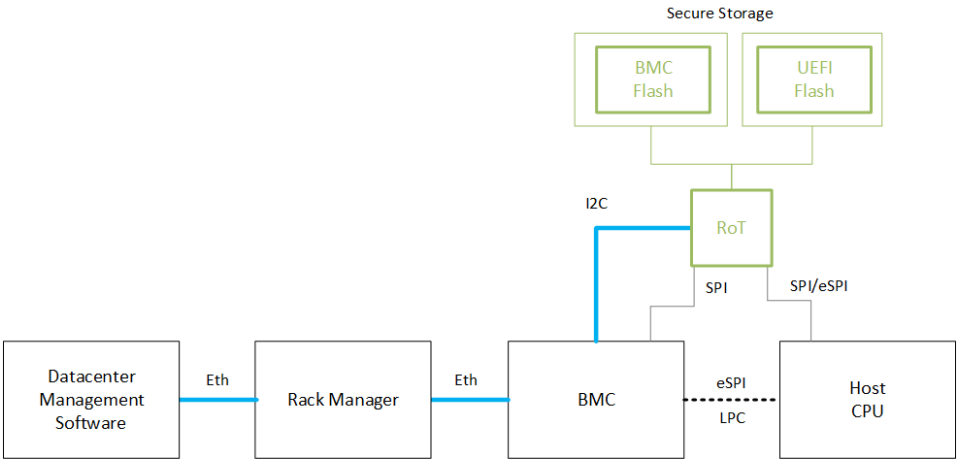
The measurements of the Platform firmware and Component firmware are compared to the PFM and CFM. Should a mismatch occur, the PA-RoT would raise an event log and invoke the policy action defined for the Platform and/or Component. A variety of actions can be automated for a PFM/CFM challenge failure. Actions are defined in the CFM and PCD files.

Note: The PA-RoT and AC-RoT enforce secure boot and only permit the download of digitally signed and unrevoked firmware. A PFM or CFM mismatch can only occur when firmware integrity is brought into question.

7.2 RoT External Communication interface

The PA-RoT connects to the platform through, either SPI, QSPI depending on the motherboard. Although the PA-RoT physically connects to the SPI bus, the microprocessor appears transparent to the host as it presents only a flash interface. The management interface into the PA-RoT and AC-RoTs is an I2C bus channeled through the Baseboard Management Controller (BMC). The BMC can reach all AC-RoTs in the platform. The BMC bridges the PA-RoT to the Rack Manager, which in-turn bridges the rack to the Datacenter management network. The interface into the PA-RoT is as follows:

Figure 12 External Communication Interface

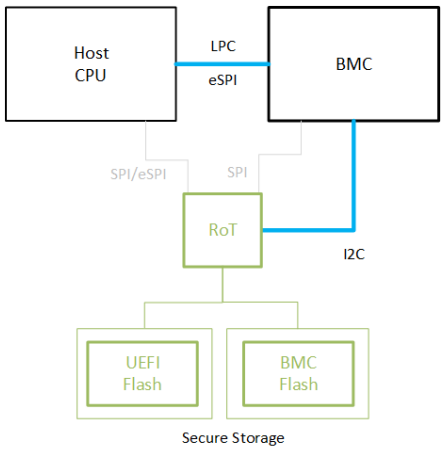


The Datacenter Management (DCM) software can communicate with the PA-RoT Out-Of-Band (OOB) through the Rack Manager. The Rack Manager allows tunneling through to the Baseboard Management Controller, which connects to the PA-RoT over I2C. This channel is assumed insecure, which is why all communicates are authenticated and encrypted. The Datacenter Management Software can collect the RFM measurements and other challenge data over this secure channel. Secure updates are also possible over this channel.

7.3 Host Interface

The host can communicate with the PA-RoT and AC-RoTs through the BMC host interface. Similar to the OOB path, the BMC bridges the host-side LPC/eSPI interface to the I2C interface on the RoT. The host through BMC is an unsecure channel, and therefore requires authentication and confidentiality.

Figure 13 Host Interface



7.4 Out Of Band (OOB) Interface

The OOB interface is essential for reporting potential firmware compromises during power-on. Should firmware corruption occur during power-on, the OOB channel can communicate with the DCM software while the CPU is held in reset. If the recovery policy determines the system should remain powered off,

it's still possible for the DCM software to interrogate the PA-RoT for detailed status and make a determination on the remediation.

The OOB communication to Cerberus requires TLS and Certificate Authentication.

8 Legacy Interface

The legacy interface is defined for backward combability with devices that do not support MCTP. These devices must provide a register set with specific offsets for Device Capabilities, Receiving Alias Certificate, accepting a Nonce, and providing an offset for Signed Firmware Measurements. The payload structures will closely match that of the MCTP protocol version. Legacy interfaces to no support session based authentication but permit signed measurements.

8.1 Protocol Format

The legacy protocol leverages the SMBus Write/Read Word and Block commands. The interface is register based using similar read and write subroutines of I2C devices. The data transmit and receive requirements are 32 bytes or greater. Large payloads can be truncated and retrieved recursively spanning multiple block read or write commands.

The block read SMBUS command is specified in the SMBUS specification. Slave address write and command code bytes are transmitted by the master, then a repeated start and finally a slave address read. The master keeps clocking as the slaves responds with the selected data. The command code byte can be considered register space.

8.2 PEC Handling

An SMBus legacy protocol implementation may leverage the 8bit SMBus Packet Error Check (PEC) for transactional data integrity. The PEC is calculated by both the transmitter and receiver of each packet using the 8-bit cyclic redundancy check (CRC-8) of both read or write bus transaction. The PEC accumulates all bytes sent or received after the start condition.

An Active RoT that receives an invalid PEC can optionally NACK the byte that carried the incorrect PEC value or drop the data for the transaction and any further transactions (read or write) until the next valid read or write Start transaction is received.

8.3 Message Splitting

The protocol supports Write Block and Read Block commands. Standard SMBus transactions are limited to 32 bytes of data. It is expected that some Active Component RoTs with intrinsic Cerberus capabilities may have limited I2C message buffer designed around the SMBus protocol that limit them to 32 bytes. To overcome hardware limitations in message lengths, the Capabilities register includes a buffer size for determining the maximum packet size for messages. This allows the Platform's Active RoT to send messages larger than 32 bytes. If the Active Component RoT only permits 32 bytes of data, the Platform's Active RoT can segment the Read or Write Blocks into multiple packets totaling the entire message. Each segment includes decrementing packet number that sequentially identifies the part of

the overall message. To stay within the protocol length each message segment must be no longer than 255 bytes.

8.4 Payload Format

The payload portions of the SMBus Write and Read blocks will encapsulate the protocol defined in this specification. The SMBus START and STOP framing and ACK/NACK bit conditions are omitted from this portion of the specification for simplification. To review the specifics of START and STOP packet framing and ACK/NACK conditions refer to the SMBus specification.

The data blocks of the Write and Read commands will encapsulate the message payload. The encapsulated payload includes a uint16 register offset and data section.

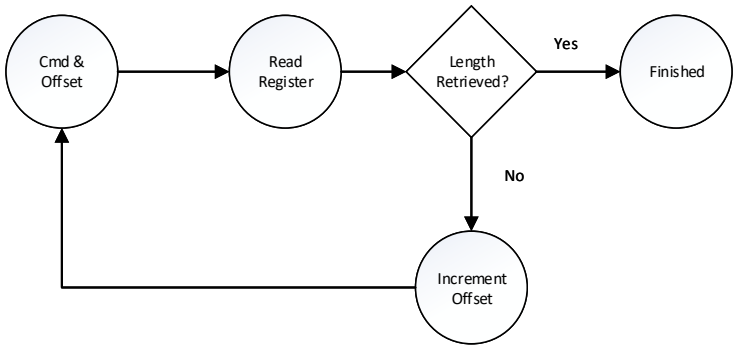
8.5 Register Format

The SMBUS command byte indexes the register, while additional writes offsets index inside the register space. The offset and respective response is encapsulated into the data portions of I2C Write and Read Block commands. The PA-RoT is always the I2C master, therefore Write and Read commands are described from the perspective of the I2C master.

Certain registers may contain partial or temporary data while the register is being written across multiple commands. The completion or sealing of register writes can be performed by writing the seal register to the zero offset.

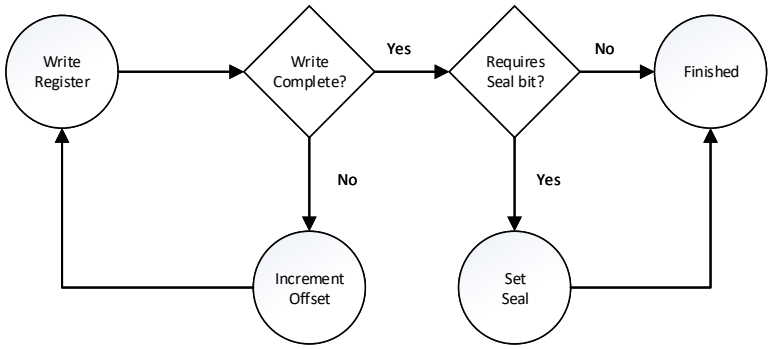
The following diagram depicts register read access flow for a large register space:

Figure 14 Register Read Flow



The following diagram depicts register write access flow for a large register space, with required seal (update complete bit):

Figure 15 Register Write Flow



8.6 Legacy Active Component RoT Commands

The following table describes the commands accepted by the Active Component RoT. All commands are master initiated. The command number is not representative of a contiguous memory space, but an index to the respective register

Table 99 Commands

Register Name	Command	Length	R/W	Description
Status	30h	2	R	Command Status
Firmware Version	32h	16	R/W	Retrieve firmware version information
Device Id	33h	8	R	Retrieves Device Id
Capabilities	34h	9	R	Retrieves Device Capabilities
Certificate Digest	3C	32	R	SHA256 of Device Id Certificate
Certificate	3D	4096	R/W	Certificate from the AC-Rot
Challenge	3E	32	W	Nonce written by RoT
Platform Configuration Register	03h	5Eh	R	Reads firmware measurement, calculated with S Nonce

8.7 Legacy Command Format

The following section describes the register format for AC-RoT that do not implement SMBUS and comply with the legacy measurement exchange protocol.

8.7.1 Status

The SMBUS read command reads detailed information on error status. The status register is issued between writing the challenge nonce and reading the Measurement. The delay time for deriving the Measurement must comply with the Capabilities command.

Table 100 Status Register

Payload	Description
1	Status: 00 = Complete 01 In Progress 02 Error
2	Error Data or Zero

8.7.2 Firmware Version

The SMBUS write command payload sets the index. The subsequent SMBUS read command reads the response. For register payload description see response: Table 11 Firmware Version Response

8.7.3 Device Id

The SMBUS read command reads the response. For register payload description see response: Table 1 Field Definitions

8.7.4 Device Capabilities

The SMBUS read command reads the response. For register payload description see response:

Table 13 Device Capabilities Response

8.7.5 Certificate Digest

The SMBUS read command reads the response. For register payload description see response: Table 24 GET DIGEST Response

The PA-Rot will use the digest to determine if it has the certificate already cached. Unlike MCTP, only the Alias and Device Id cert is supported. Therefore, it must be CA signed by a mutually trusted CA, as the CA Public Cert is not present

8.7.6 Certificate

The SMBUS write command writes the offset into the register space. For register payload description see response: Table 26 GET CERTIFICATE Response

Unlike MCTP, only the Alias and Device Id cert is supported. Therefore, it must be CA signed by mutually trusted CA, as the CA Public Cert is not present in the reduced challenge

The SMBUS write command writes a nonce for measurement freshness.

Table 101 Challenge Register

Payload	Description
1:32	Random 32 byte nonce chosen by PA-RoT

8.7.7 Measurement

The SMBUS read command that reads the signed measurement with the nonce from the hallenge above. The PA-RoT must poll the Status register for completion after issuing the Challenge and before reading the Measurement.

Table 102 Measurement Register

Payload	Description
1	Length (L) of following hash digest.
2:33	H(Challenge Nonce || H(Firmware Measurement/PMR0))

34:N	Signature of HASH [2:33]

9 References

9.1 DICE Architecture

https://trustedcomputinggroup.org/work-groups/dice-architectures

9.2 RIoT

https://www.microsoft.com/en-us/research/publication/riot-a-foundation-for-trust-in-the-internet-of-things

9.3 DICE and RIoT Keys and Certificates

https://www.microsoft.com/en-us/research/publication/device-identity-dice-riot-keys-certificates

9.4 USB Type C Authentication Specification

http://www.usb.org/developers/docs/

9.5 PCIe Device Security Enhancements specification

https://www.intel.com/content/www/us/en/io/pci-express/pcie-device-security-enhancements-spec.html

9.6 NIST Special Publication 800-108

Recommendation for Key Derivation Using Pseudorandom Functions.
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf

9.7 TCG PC Client Platform Firmware Profile Specification

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