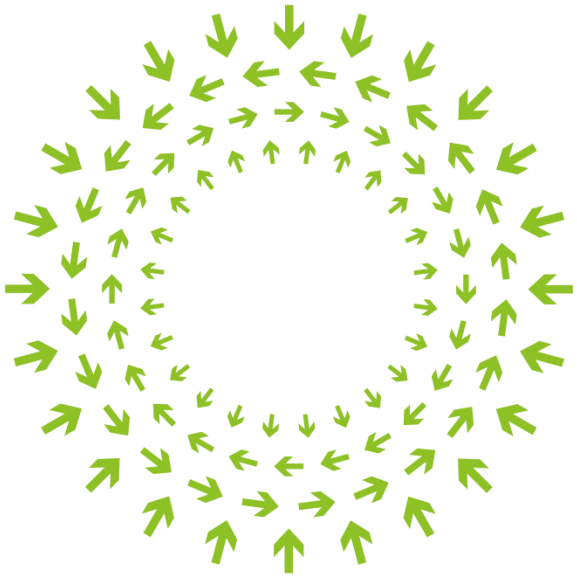


OPEN
Compute Project

Project Cerberus Firmware Challenge Specification

Author:
Bryan Kelly, Principal Firmware Engineering Manager, 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

Table of Contents

List of Figures

List of Tables

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 hieratical 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

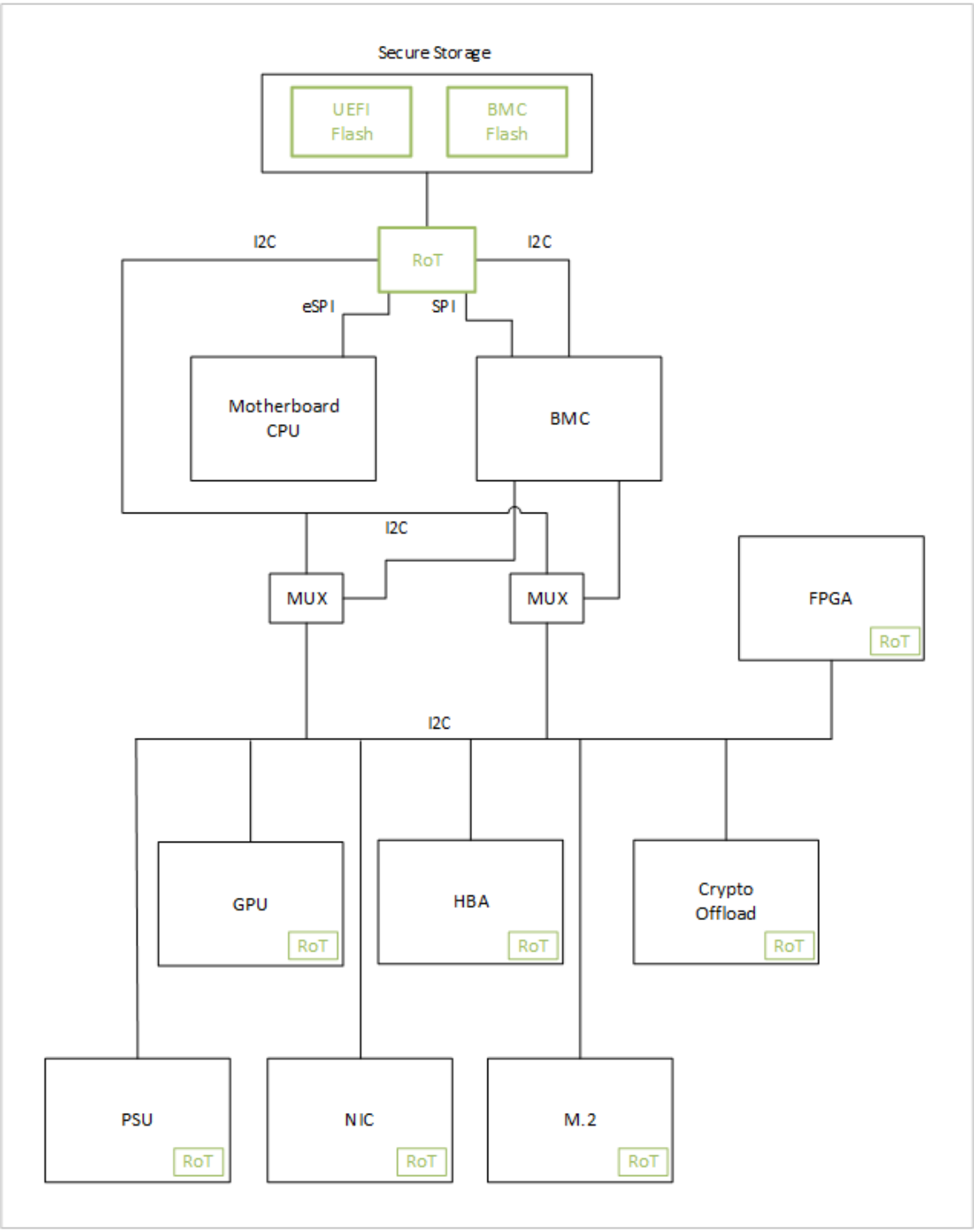
The typically cloud server motherboard layout has I2C buses routed to all Active Components. These I2C buses are typically used by the Baseboard Management Controller (BMC) for the thermal monitoring of Active Components. In the Cerberus board layout, the I2C lanes are first used by the platform Cerberus 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. 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).

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 capabilities 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 enforced firmware integrity 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 kept in power-on reset, and Active Components must respond to challenges from the PA-RoT confirming the integrity of their firmware before they receive full voltage to power their primary functional interfaces such as PCIe.

In this version of the Cerberus platform design, the Platform Firmware Manifest (PFM) is static. The manifest is programmable through the PA-RoT's communication interface. Auto-detection of Active Components and computation of the PFM maybe considered in future version of the specification.

The PA-RoT uses the manifest to challenge the Active Components and record their measurements. The PA-RoT uses a digest of these measurements for 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 and all Active Components RoT's (AC-RoT) will support Authentication, Confidentiality and Integrity of message challenges. To facilitate this, AC-RoT are required to support certificate authentication. Once a session is established, AES encryption can be used for confidentiality. The Active Component will support a unique 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 for HMAC of critical data.

1.1 Power Control

In the Cerberus motherboard design, power and reset sequencing is orchestrated by the Platform's Active RoT. When voltage is applied to motherboard, it passes through in-rush circuity to a CPLD that performs time sensitive sequencing of power rails to ensure stabilization. Once power good level is established, the platform is considered powered. Upon powering the platform in the Cerberus design, the only active component powered-on is the Active 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 standby power to the PCIe slots and begins Active Component RoT challenge. When the UEFI has been authenticated the platform is held in system reset, the Active RoT will keep the system in reset until Active Component RoT's 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 polices 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 platform Active RoT communicates with the Active Component RoT's over I2C. The protocol supports a challenge for key exchange and measurement channel. The Cerberus Platform Active RoT has the intrinsic ability to generate a secure key pair unique to the microcontroller. The private key is inaccessible outside of SRAM on the Cerberus RoT. Key generation and chaining follows the RIoT specification, described in section: 7.3 DICE and RIoT Keys and Certificates. Derived platform public keys are distributed to the Active Components RoT's. This public key is used by the Active Component RoT's to sign the digest their firmware measurements and establish a symmetric key for message encryption.

2.1 RSA Key Generation

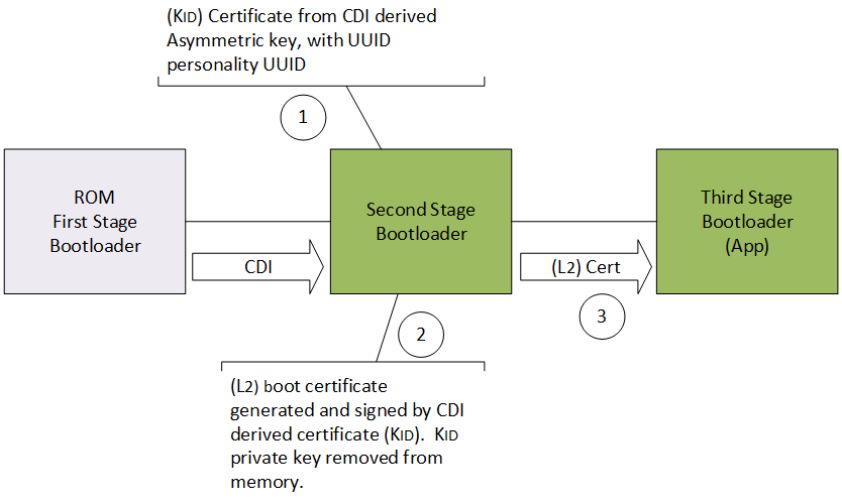
The Cerberus platform Active RoT should support the Device Identifier Composition Engine (DICE) architecture. In DICE, the Composite Device Identifier (CDI) is used as the foundation for device identity and attestation. While keys derived from the device secret that generates the CDI is used for data protection within the microcontroller. The Device Id key is derived cryptographically from the CDI, using the UUID for personality. Furthermore, Cerberus implements RIoT architecture for certificate generation and attestation. For details on key generation on DICE and RIoT review section: 7.3 DICE and RIoT Keys and Certificates

Note: The CDI is a composite key based on the Device Secret (Device Unique), and seconds stage bootloader (mutable). The second stage bootloader is mutable code, but not typically updated with firmware updates. The second stage bootloader may change during device revocation. This will result in a CDI change, which would cause boot time certificate chains to break. A change to the CDI would result in different keys in derivation chain. Key revocation typically requires re-keying of the device.

The CDI key can generate an asymmetry Public/Private key pair with the device UUID and deterministically regenerate the same key pair provided the CDI that derives does not change. The CDI can only change if the mutable code of the second stage bootloader is changed. Proof-of-knowledge of the CDI private key can be used as a building-blocks in a cryptographic protocol to identify the device.

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 can be used to establish TLS style session proving the device and software 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 entities.

Figure 2 Id Key Generation

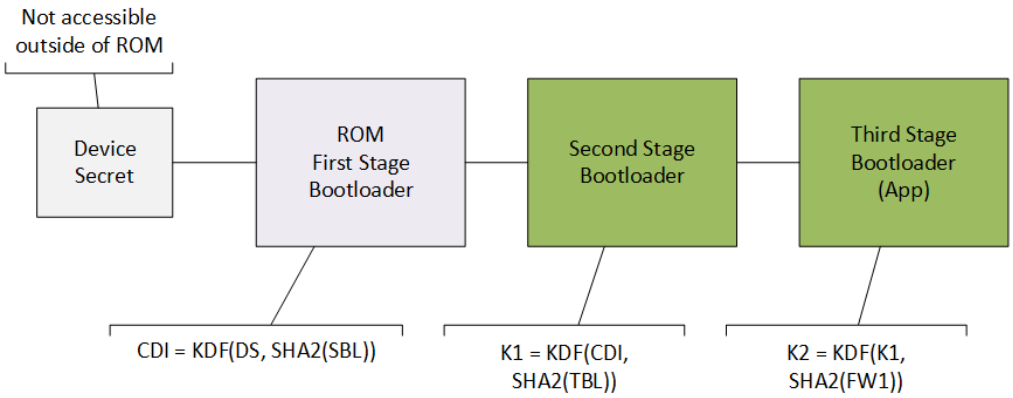


2.2 Chained Measurements

The Cerberus firmware measurements 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, SHA2(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, SHA2(TBL)). The TBL will take additional area measurements of the QSPI flash it protects, both active and inactive areas. At each stage, the measurement captured should be signed by the attestation key for the firmware level calculating the measurement. The measurement and signature should be made available for the measurement challenge.

Figure 3 Measurement Calculation



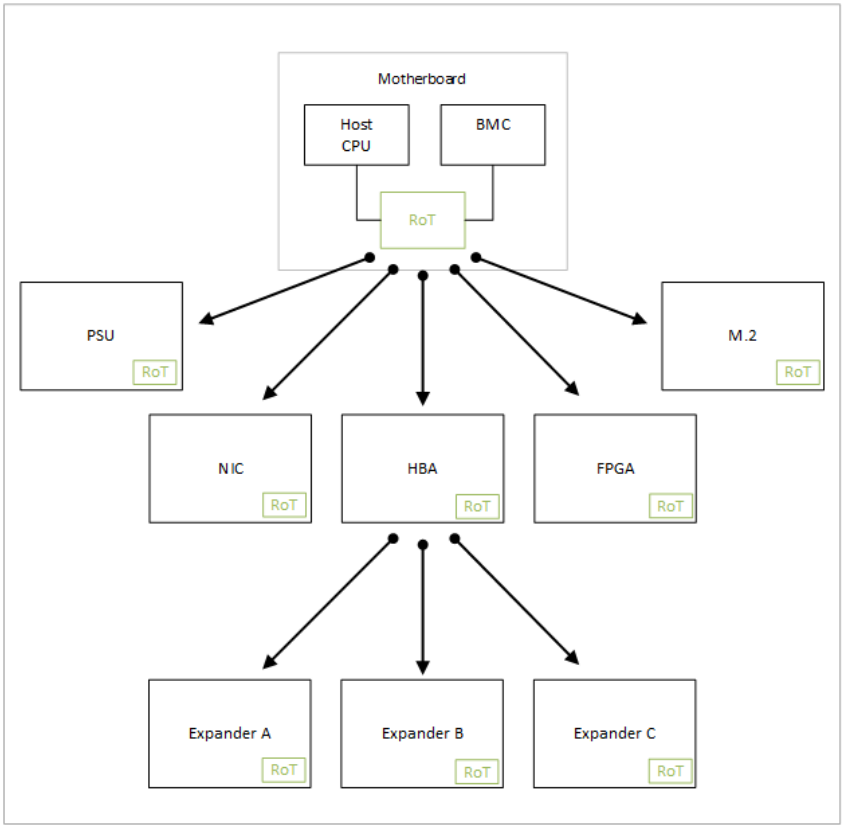
2.1 Protocol and Hierarchy

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 RoT's are required to support the following I2C protocol.

The protocol is derived from the SMBus protocol with similar START and STOP framing conditions. Unlike SMBUS the Active Components are not restricted the 32 byte payload limitation. The Active Component RoT's with I2C FIFO limitations can advertise their maximum buffer limit. The master will query the device capabilities and adhere to the buffer limitations. The Write Block command will increase its offset after every incremental read until the total message length has been read from the register space.

The Platform Cerberus Active RoT is always the I2C master. Active Component RoT's can be configured as Slave, or Slave and Master. There is no requirement for bus arbitration as master and slave definitions are hierarchically established. The only hierarchy whereby the Active Component RoT becomes both Slave and Master is when there is a downstream sub-device, such as the Host Bus Adapter (HBA) depicted in the following block diagram:

Figure 4 Root of Trust Hierarchy



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

The messaging protocol encompasses SMBus "Write Word", "Read Word", "Write Block" and "Read Block" commands, whereby the Active Component RoT is always accessed as slave the Platform's Active RoT as master.

3 Protocol Format

This version of the specification encapsulates the protocol inside SMBus Write/Read Word and Block commands. The interface is register based using similar read and write subroutines of I2C devices. A future specification may consider other already established I2C protocols for transmitting the structures described in this specification. 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.

3.1 PEC Handling

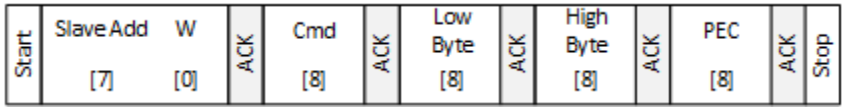
The SMBus protocol implementation leverages the 8bit SMBus Packet Error Check (PEC) for transactional data integrity. The PEC is calculated by both the transmitter and receiver of each packet using the 8-bit cyclic redundancy check (CRC-8) of both read or write bus transaction. The PEC accumulates all 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.

3.2 Write Word Command

The first byte of the Write Word command is the command code, followed by the data to be written. The following diagram demonstrates the master RoT sending the Component RoT a Write Word.

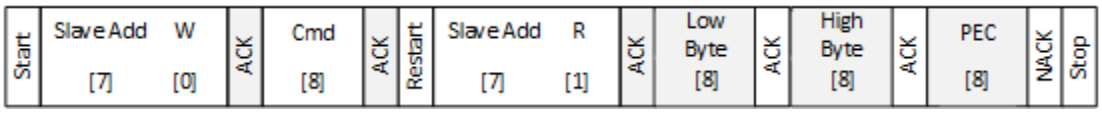
Figure 5 Write Word Command



3.3 Read Word Command

The first byte of the Write Word command is the command code. The following diagram demonstrates the master RoT sending the Component RoT a Write Word.

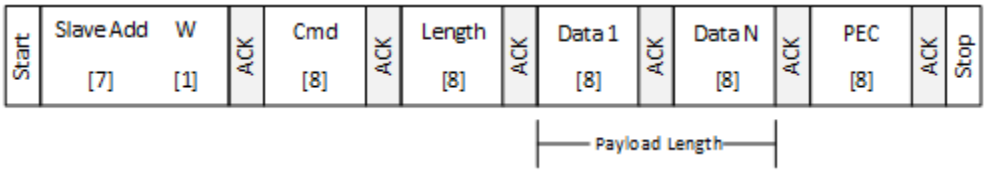
Figure 6 Read Word



3.4 Write Block Command

The Write Block transaction transfers messages up to the N bytes in length. The length field provides the count of data bytes in the payload. The block write begins with a slave address and write condition, after the command code the byte count indicates the length of bytes to follow. The maximum packet length per message determined by the session negotiated packet length.

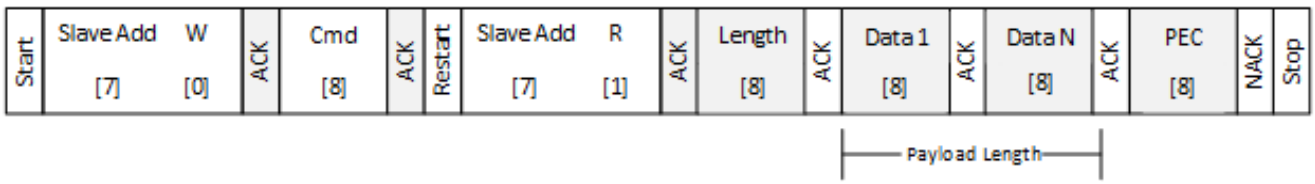
Figure 7 Write Command



3.5 Read Block Command

The read command contains a response to the responder slave address. The length of each packet is determined by packet length negotiated during session setup. The total length of a message can be transferred over multiple packets. The Read Block begins with a slave address and a write condition. After the command a byte count which describes how many more bytes will follow in the message

Figure 8 Block Read Command



3.6 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.

3.7 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.

Figure 9 Command Payload

The data blocks of the Write and Read commands will encapsulate the message payload. The encapsulated payload includes a uint16 register offset, sequence number and command identification byte in addition to the body of the command in the data section.

4 Sessions

A session in the context of this specification is the process of establishment of authentication, integrity and confidentiality between the external data center software and the PA-RoT, or between the PA-RoT and the AC-RoT. Since only a single session is supported on a given I2C channel, there are three types of session supported by the Platform RoT, secured and authenticated, secured and unsecured. The secured authenticated sessions will use certificate authentication.

Although I2C buses are internal to the system, in some circumstances cable routing to external chassis can take these signals off the motherboard. In these instances, the AC-RoT should support secure session adding additional encryption to deter physical the protocol snooping and replay. The secure protocol requires a certificate and challenge between the PA-RoTs and the AC-RoT. This mechanism provides authentication, confidentiality and data integrity.

4.1 Session establishment

The initial session authentication uses certificate authentication that results in a secret being exchanged that forms session keys for a MAC and confidentiality.

To establish a session over I2C, the sequence starts with the PA-RoT issuing a SMBUS Block Read to Session Query Command. The AC-RoT responds with cipher capabilities and random number. The PA-RoT then issues a Activate Session command contain a RN2.

Note: If the device signals, it does not support sessions the following public key exchange does not occur. The command exchanges occur in a session less context, whereby the Platform RoT random forms the session id for ensuring freshness of the measurement KDF.

The PA-RoT will challenge the AC-RoT for a session by issuing a Challenge Certificate block read command. The AC-RoT will respond with a public certificate containing a CA digitally signed signature.

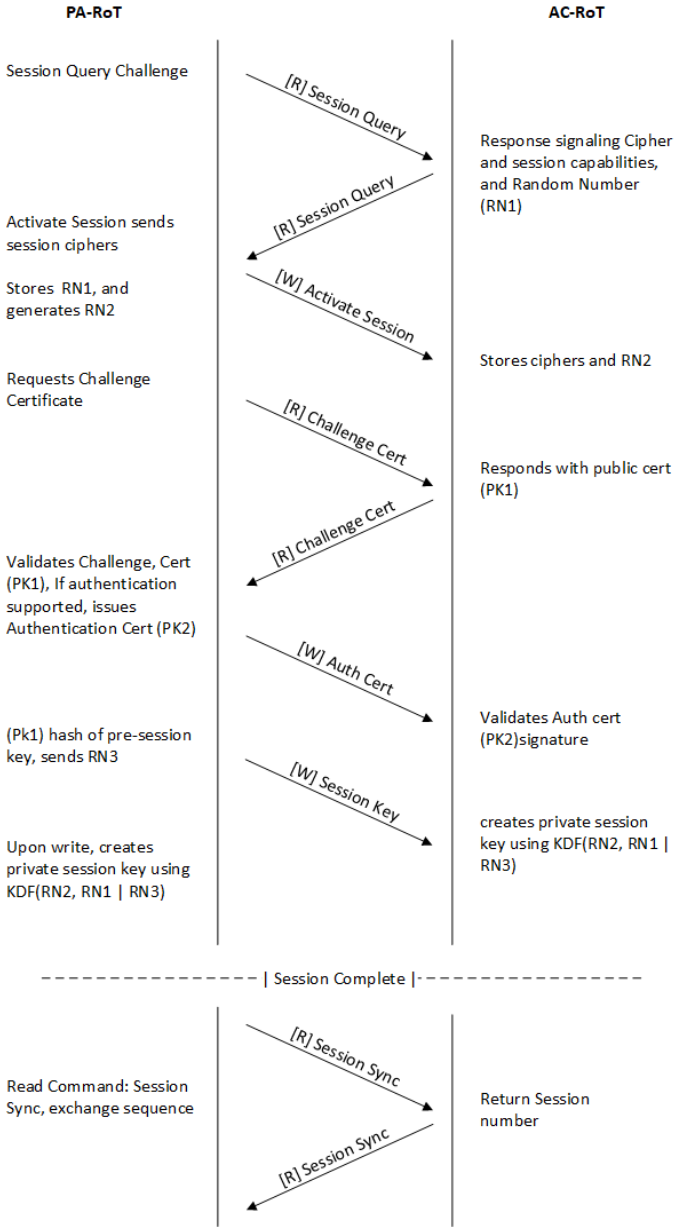
The PA-RoT will verify the signed certificate signature of the AC-RoT, if verification fails or certificate has been revoked, the session challenge will fail. The PA-RoT and AC-RoT can be updated with revocation patches, see firmware update specification for further details.

If the certificate signature is valid, and the AC-RoT supports authentication, the PA-RoT will write its signed public certificate to the AC-RoT. The AC-RoT will verify the integrity of the certificate using the digital signature. If the signature does not match or the certificate has been revoked the challenge will fail.

The PA-RoT will then write a random number (pre-master) hashed with public key of the certificate from the AC-RoT.

The Active RoT and the Active Component will use the random numbers and pre-master key to generate session keys. These will form a symmetric key for encryption for

Figure 10 Session Setup



the remainder of their communication. This sequence is a TLS handshake mechanism.

4.2 Session TLS Setup

1. PA-RoT: Session Query
   - AC-RoT: provides max payload size, ciphers supported, session support
   - AC-RoT: provides random number $^{(RN1)}$
2. PA-RoT: Session Activate, specifying ciphers
   - PA-RoT provides random number $^{(RN2)}$
3. PA-RoT: Challenge Certificate, Public cert $^{(pk1)}$
   - PA-RoT, verifies Cert Signature of PK1
4. PA-RoT: If Session Query $^{(1)}$ indicates Authentication is supported.
   - PA-RoT: Authentication Certificate, Public cert $^{(PK2)}$
   - AC-RoT: verifies Cert Signature of PK2
5. PA-RoT Session Key, pre session key$^{(RN3)}$ hashed with PK1
   - Both sides generate session key, HMAC(RN3, RN2 | RN1)
   - AC-RoT responds with AC.
6. Session Activation transmission AES encrypted
7. Session Sequence incrementally changes on message transactions.
8. Session sync for session state.

4.3 Session-less Setup

For Active Component RoT's that do not support sessions, a derived session id is still required for measurement freshness. The session handshake follows a subset of the exchanges for RoT's supporting sessions. The following differences occur:

1. Get Session, containing random number
   - Response provides max payload size, *ciphers supported, *session support
   - AC-RoT provides a random number$^{(RN1)}$
2. Set Session, specifying ciphers none.
   - PA-RoT provides random number $^{(RN2)}$
   - Both sides generate session key, HMAC(RN2, RN1)
3. Session Sequence incrementally changes on message transactions.
4. Session sync for session state.

Note: When session support signals the RoT does not support sessions, the RN1 is used as the key for HMAC functions with algorithms defined in Session Challenge.

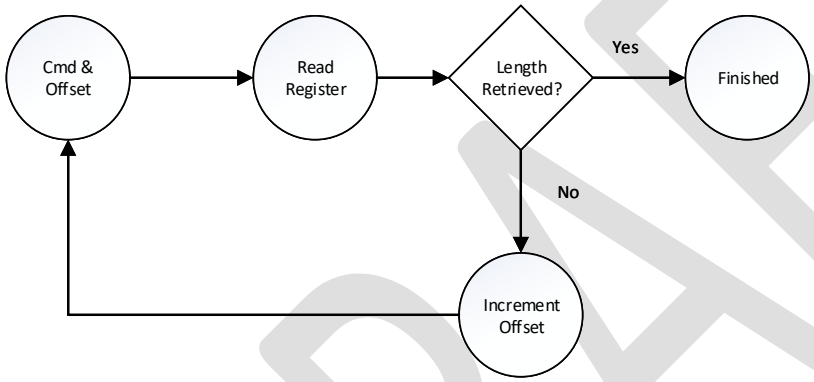
5 Register Format

The following section describes the register format to support the challenge protocol. 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.

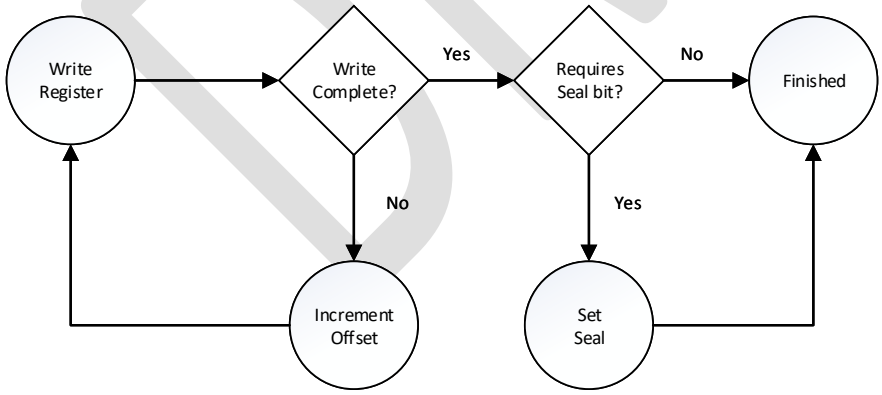
Certain registers may contain partial or temporary data while the register is being written across multiple commands. The completion or sealing of register writes can be performed by writing the seal register to the zero offset.

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

Figure 11 Register Read Flow



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



5.1 Type Code

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

Table 1 Command Types

Type	Description
1	Accepted inside or outside session. Typically pre-session commands
2	Session setup commands
3	Session required commands, obfuscated by session encryption or HMAC, register content is normally scrambled.

The protocol follows the typical I2C register based interface whereby given register offsets represent write and read registers. The reserved register space should be sufficient to accommodate the data described for the register.

5.2 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 2 Register

Register Name	Type	Command	Length	R/W	Description
Firmware Version	01h	32h	16	R/W	Retrieve firmware version information
Device Id	01h	33h	8	R	Retrieves Device Id
Capabilities	01h	34h	9	R	Retrieves Device Capabilities
Session Query	02h	3C	7	R	PA-RoT retrieves session information
Session Activate	02h	3D	7	W	PA-RoT sets session variables based on Session Query
Challenge Certificate	02h	3Eh		R	PA-RoT retrieves and verifies AC-RoT certificate
Authentication Certificate	02h	3Fh		W	PA-RoT sends session certificate
Session Key	02h	40h		W	Exchange pre-master session keys
Get Session Sync	02h	25h		R	Retrieve session state from RoT
Set Policy	03h	4Ah	5	W	Set Firmware and recovery failure policy
Get Policy	03h	4Bh	5	R	Get Firmware and recovery failure policy
Get Debug Log	03h	50h		R	Retrieve debug log
Clear Debug Log	03h	51h		W	Clear debug log
Get Tamper Log	03h	52h		R	Get FW change log
Flash Descriptors	03h	5Ah		R	Retrieve session challenge
Signature Key	03h	5Bh		W	Updates
Signature Refresh	03h	5Ch		W	Refreshes firmware measurement, calculated with signature key.

Retrieve Signature	03h	5Eh		R	Reads firmware measurement, calculated with Signature key.
Recovery Firmware	03h	5Dh		W	Restore Firmware Index using backup.
Flash Checksum	03h	5Fh		R	Get Flash area checksum

5.3 Register Structures

The following section describes the structures of registers. When written or read, the register content exchange is encapsulated in the "data" field described in 3.7 Payload Format. The register offsets are described in section: 5

5.4 Firmware Version

This command write command sets the target firmware block to retrieve the version.

Table 3 Firmware Version Write Request

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

The subsequent read to the Firmware Version registers, retrieves the version associated with the previously written area index. Sixteen contiguous bytes should be provisioned in the register space for firmware version. The data should be ASCII formatted and null terminated.

Table 4 Get Firmware Version Response

Register	Description
1:15	Firmware Version Number ASCII Formatted

5.5 Capabilities

Eight bytes are reserved in the register space for the response.

Table 5 Device Capabilities

Payload	Description
1	Payload Size:
2	Mode: [7:6] 00 = AC-RoT 01 = PA-RoT [5:4] Master/Slave 00 = Unknown 01 = Master 10 = Slave [3] Reserved [2:0] Security 0000 = None 0001 = Hash/KDF 0010 = TLS 0100 = Auth

5.6 Device Id

Eight bytes are reserved in the register space for the response.

Table 6 Get Device Id Read Response

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

5.7 Session Query

This register provides the RoT session capabilities. Seven bytes are reserved in the register space for reading the response. The data is always unencrypted/unscrambled. The Random Number is randomized on every read.

Table 7 Get Session Write Request

Payload	Description
1	Bit [7] Session Supported: 01b = Session Supported 00b = Session not Supported Bit [6] reserved Bit [5:3] Session Type 000 = unsecured 001 = secured 010 = secured and Authenticated Bit [2:0] confidentiality ciphers supported 000 = None 001 = 128 AES CBC 010 = 256 AES CBC
2	Bit [7:5] Algorithm 000 = None 001 = SHA1 010 = SHA2 100 = SHA3 Bit [3:0] Hashing algorithms supported 000 = None 001 = SHA-128 010 = SHA-256 100 = SHA-512
3	Bit [7:4] Reserved RSA Signature 0000 = None 0001 = 512 0010 = 1024 0100 = 2048 1000 = 4098
4:7	32-bit random number (RN1). LSB First.

5.8 Session Activate

Writing to this register initiates a session; the register can be written at any time. If a session was previously established it will renegotiate/override the session. As devices only support a single session over a given I2C channel, this effectively closes and restarts a new session on that channel. Registers written on a given channel pertain to that channel only.

Table 8 Session Activate

Payload	Description
1	Bit [7] Session Supported: 01b = Session Supported 00b = Session not Supported Bit [6] reserved Bit [5:3] Session Type 000 = unsecured 001 = secured 010 = secured and Authenticated Bit [2:0] confidentiality ciphers supported 000 = None 001 = 128 AES CBC 010 = 256 AES CBC
2	Bit [7:5] Algorithm 000 = None 001 = SHA1 010 = SHA2 100 = SHA3 Bit [3:0] Hashing algorithms supported 000 = None 001 = SHA-128 010 = SHA-256 100 = SHA-512
3:6	32-bit random number (RN2). LSB First.

5.9 Challenge Certificate

This register space contains the public attestation certificate for the AC-RoT. If the Session Query command byte 1, bit [7] indicates sessions are support the PA-RoT read this register space for the AC-RoT's public certificate.

Table 9 Get Session Certificate Register Space

Register	Description
1:2	Certificate Length
3:N	Certificate data (PK1)

The Challenge Certificate command/register will a zero offset will read the first N number of bytes. Where N is the number of bytes to read. The first two bytes in the register contain the certificate length. The PA-RoT will issue a Write Word or Block Write to offset into the register space.

Example:

Master: Block Read: Cmd: = 21h

Slave: 2 bytes Certificate Length + 30 bytes of Certificate data.

Write Word: Cmd 21h, Offset: 2000h

5.10 Authentication Certificate

If the Session Query register indicates authentication is supported/required, the PA-RoT will write its public certificate on the AC-RoT. If secure and authenticated session are supported, and the AC-RoT certificate verifies successfully, then the PA-RoT will write this challenge certificate.

Table 10 Session Certificate Authenticate

Register	Description
1	Register Seal: 00: Write Complete 01: Write in Progress
2:3	Length
4:N	Certificate data (PK2)

The AC-RoT should verify the authenticity of the Authentication Certificate when the command is complete.

5.11 Session Key

After verifying the Challenge Certificate authenticity, the PA-RoT will write a public key-hashed pre-session key to the AC-RoT, using the PA-RoT's public key.

Table 11 Key Exchange Write

Register	Description
1	Register Seal: 00: Write Complete 01: Write in Progress
2:3	Length
4:N	(PK1)Hash of pre-session key (RN3)

5.12 Set Policy

Set the policy to use for slave attestation or firmware recovery failures. The policy actions occur in series, for triggering multiple actions

Table 12 Set Policy Request

Register	Description
1	Device Id: 00 for AC-RoT, component index for PA-RoT
2	Area Index: Offset to firmware region
3	Policy Id
4	[7:5] Policy Type: 01h = Boot Firmware recovery failure 02h = Runtime Firmware recovery failure 03h = Boot Attestation failure 04h = Runtime Attestation failure [4] Threshold Counter, 00b = Enabled 01b = Disabled [3:0] Threshold Count
5	[7:4] First Action: 00h: Warning Log 01h: Error Log 02h: Reset 03h: Power off Device 04h: Restore active image 05h: Restore default image [3:0] Second Action: 00h: Warning Log 01h: Error Log 02h: ACPI Shutdown/Reset 03h: Power off Device 04h: Restore active image 05h: Restore default image

5.13 Get Policy

Get the current policy for handling firmware recovery or attestation failures.

Table 13 Get Policy Response

Register	Description
1	Device Id: 00 for AC-RoT, component index for PA-RoT
2	Area Index: Offset to firmware region
2	Policy Id
4	[7:5] Policy Type: 01h = Boot Firmware recovery failure 02h = Runtime Firmware recovery failure 03h = Boot Attestation failure 04h = Runtime Attestation failure [4] Threshold Counter 00b = Enabled 01b = Disabled [3:0] Threshold Count
5	[7:4] First Action: 00h: Warning Log 01h: Error Log 02h: Reset 03h: Power off Device 04h: Restore active image 05h: Restore default image [3:0] Second Action: 00h: Warning Log 01h: Error Log 02h: Reset 03h: Power off Device 04h: Restore active image 05h: Restore default image

5.14 Get Debug Log

Get the internal log for the RoT.

Table 14 Get Debug Log Response

Register Offset	Description
1:2	Length in bytes
3:N	The contents of the log

5.15 Clear Debug Log

Clear the log in the RoT.

Table 15 Clear Debug Log Request

Register Offset	Description

5.16 Get Tamper Log

Get the change history of critical RoT components.

Table 16 Get Tamper Log Register Space

Payload	Description
1:2	Length in bytes
3:N	The contents of the log

5.17 Flash Descriptors

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

Payload	Description
1	Area Count. Each following 2 byte increments describe an Area in indexed order:
2:N	Byte 1: Index Number [7] Reserved [6:0] Index Number Byte 2: Identifier [7] 00: Read-Only 01: Writable [6] 00: Primary 01: Backup [5:4] Area: 00: Signature 01: Metadata 02: Boot Loader 03: Application Firmware 3:0 Instance Id

5.18 Signature Key

This register sets a key for the HMAC of the public key signed FW image. The Key is used in recalculation of the firmware digest. Writing to the register t*riggers the Active Component to re-calculate signatures*

Table 17 Signature Key

Payload	Description
1	Nonce [7] 00b No KDF, ignore nonce in bytes 3-6 01b KDF with nonce in bytes 3-6 [6] Index Number
2:6	Nonce for signature KDF

5.19 Signature Refresh

Triggers the Active Component to re-calculate signatures against the attestation freshness seed. The command is not intended to reload the component and calculate the hash chain that forms the measurement, it is intended to ready the measurement for exchange performing a KDF with the freshness signature key This command may be send ahead of the Read signature challenges.

Table 18 Signature Refresh

Payload	Description

5.20 Retrieve Signature

The Signature registers contain the corresponding signature written in the 5.17 Flash Descriptors

Table 19 Signature Register

Payload	Description
1:2	Length in bytes
3:N	Chained digest of firmware region, with KDF of the signature using nonce in 5.18 Signature Key

5.21 Recover Firmware

Start the firmware recovery process for the device. Not all devices will support all types of recovery.

Table 20 Recover Firmware Request

Register Offset	Description
1	Firmware image to use for recovery: 0: Index

5.22 Flash Checksum

This command returns a CRC-32 checksum of the flash block.

Table 21 Write Flash Checksum offsets

Register	Description
1:3	Start Address LSB first.
4:7	End Address LSB First.

Table 22 Read Flash Checksum

Register	Description
1	[3:0] Status: 0. Complete 1. In-progress 2. Error [7:4] Sub Status
2:3	Time in seconds to complete
4:7	CRC-32 Checksum of flash block

5.23 Set Hash Key (PA-RoT)

This register is only used on the PA-RoT, it sets a key for the HMAC of the public key signed FW image, used in recalculation of the firmware digest.

Table 23 Set Firmware Challenge

Payload	Description
1	Device Id
2	Area Index, offset to firmware region
3:6	HMAC Key

5.24 Firmware Challenge (PA-RoT)

This register returns HMAC of the public key signed FW image digest with the HMAC salt.

Table 24 Firmware Challenge

Payload	Description
1:2	Length in bytes
3:N	HMAC(Key, $SGN^{(pk)}$(FW))

6 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.

6.1 Platform Firmware Manifest (PFM)

The PA-RoT contains a Platform Firmware Manifest (PFM) that describes the firmware permitted on Active Component in the system. Note: The PFM is different from the boot key manifest described in the Processor Secure Boot Requirements specification. The PFM describes firmware permitted to run in the system across all Active Components. A complement to the PFM is generated by the PA-RoT for the measurement comparison of components in the system. This complement is as the Reported Firmware Manifest (RFM). 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 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 25 PFM Attributes

Attribute	Description
Device UUID	Globally Unique Id for the AC-RoT
Device Address	I2C address and bus
Device Part#	Device Part Number
Device Type	Underlying Device Type of AC-RoT
Device ID	CDI derived MFG Certificate
Device Support	Device Session Support
Remediation Policy	Policy(s) defining default remediation actions for integrity failure.
Firmware Version	List of firmware versions
Firmware Approved	List of approved versions
Flash Areas	List of offset and CRCs, used and unused
Public Key	Public keys in the key manifest
Approved Public Key	List of approved keys

Signature	Firmware signature(s)
Digest Algorithm	Algorithm used to calculate
Firmware Digest	Digest of firmware image
Tamper logs	GMC of tamper log

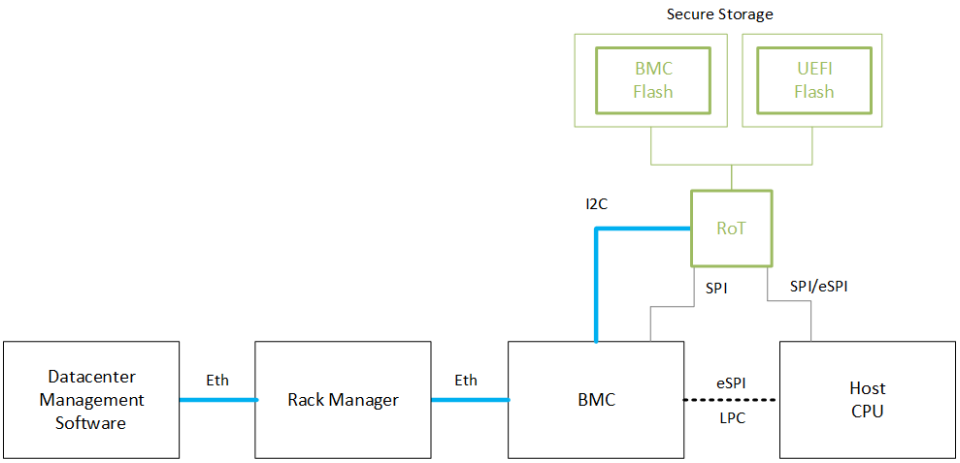
The PA-RoT maintains the original and copy of the PFM date. When the PA-RoT collects data in from the AC-RoTs it compares values in the PRM, while constructing the RFM. A digest of the original PFM is generated and compared to a digest of the RFM. Should a mismatch occur, the PA-RoT would raise an event log and invoke the policy action defined for the AC-RoT. A variety of actions can be automated for a PFM challenge failure. The actions are defined by the Active Component and Platform policies.

Note: The PA-RoT and AC-RoT enforce secure boot and only permit the download of digitally signed and unrevoked firmware. A PFM mismatch can only occur if updates are not performed correctly, whereby the PFM is notified of legitimate firmware change.

6.2 RoT External Communication interface

The PA-RoT connects to the platform through, either SPI, eSPI or 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



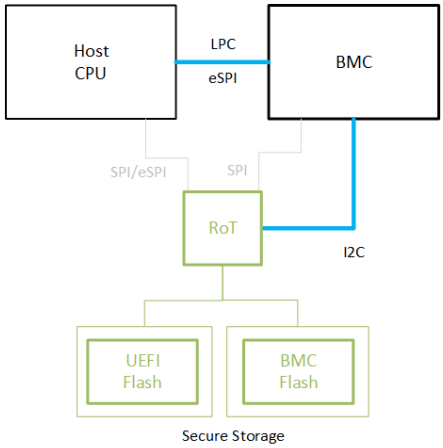
The Datacenter Management (DCM) software can communicate with the PA-RoT Out-Of-Band (OOB) through the Rack Manager. The Rack Manager allows tunneling through to the Baseboard Management Controller, which connects to the PA-RoT over I2C. This channel is assumed insecure, which is why all communicates 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.

6.3 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



6.4 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.

7 References

7.1 DICE Architecture

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

7.2 RIoT

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

7.3 DICE and RIoT Keys and Certificates

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