PMBus®
Power System Management Protocol Specification
Part IV – Security Commands And Processes

Revision 1.5
25 Aug 2025

www.powerSIG.org

Filename: PMBus_Specification_Part_IV_Rev_1_5_20250825.docx
Last saved: 25 Aug 2025, 20:59

DISCLAIMER

This specification is provided “as is” with no warranties whatsoever, whether express, implied, or statutory, including but not limited to any warranty of merchantability, non-infringement, or fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample.

In no event will any specification co-owner be liable to any other party for any loss of profits, loss of use, incidental, consequential, indirect, or special damages arising out of this specification, whether or not such party had advance notice of the possibility of such damages. Further, no warranty or representation is made or implied relative to freedom from infringement of any third party patents when practicing the specification.

Other product and corporate names may be trademarks of other companies and are used only for explanation and to the owner’s benefit, without intent to infringe.

REVISION HISTORY

REV	DATE	DESCRIPTION	AUTHOR AND EDITOR
1.5	25 Aug 2025	First public release	Author: Robert W. Santucci, Intel Editor: Robert V. White Embedded Power Labs

Table of Contents

Table of Figures

Table of Tables

1. Introduction

This standard defines a hardware interface implementation for the software features defined in the PMBus secure device application profile. Although the standard hardware implementation is not mandated, it is strongly encouraged to maximize compatibility between different PMBus device manufacturers' implementations and minimize software changes between their solutions. The objective of these security features is to prevent cyber-physical attacks, which can expose data or damage systems.

1.1.1 Specification Structure

The PMBus specification is in four parts plus a secure device application profile.

Part I includes the general requirements, defines the transport, electrical interface and timing requirements of hardwired signals for PMBus.

Part II defines the command language used with PMBus.

Part III defines the transport, electrical interface, timing requirements and command language for PMBus 1.x

Part IV, this document, describes the memory structure, security action requests, and related hardware processes that enable security through PMBus and the PMBus Secure Device Application Profile.

The PMBus secure device application profile presents the requirements to achieve different security profile levels, including standardized software APIs.

1.1.2 What Is Included

This specification defines: the security memory map used to transfer the data required for security action requests and processes, the processing of that data through the use of security action requests, and the mapping of the outputs of security action requests back to the memory map to be available for reading.

1.1.3 What Is Not Included In The PMBus Specification

The PMBus specification is not a definition or specification of:

- A particular power conversion device or family of power conversion devices.
- A specification of any individual or family of integrated circuits.

This specification does not address direct unit to unit communication such as analog current sharing, real-time analog or digital voltage tracking, and switching frequency clock signals.

1.2 Specification Changes Since The Last Revision

A summary of the changes between this revision and the previous revision are shown in Appendix II at the end of this document.

1.3 Where To Send Feedback And Comments

Please send all comments by email to: TechQuestions@smiforum.org.

2. Related Documents

2.1 Scope

If the requirements of this specification and any of the reference documents are in conflict, this specification shall have precedence unless otherwise stated.

Referenced documents apply only to the extent that they are referenced.

The latest version and all amendments of the referenced documents at the time the device is released to manufacturing apply.

2.2 Applicable Documents

Applicable documents include information that is, by extension, part of this specification.

[A01] System Management Bus (SMBus) Specification, version 3.1.
http://smbus.org/specs/SMBus_3_1_20180319.pdf

[A02] PMBus™ Power System Management Protocol, Part II, Command Language, Revision 1.5
https://pmbus.org/current-specifications/

[A03] PMBus™ Secure Device Application Profile, Revision XX,
https://pmbus.org/APPLICATION_PROFILES // LINK TBD.

[A04] NIST FIPS 180-4: Secure Hash Standard
http://dx.doi.org/10.6028/NIST.FIPS.180-4

[A05] NIST FIPS PUB 198-1: The Keyed-Hash Message Authentication Code (HMAC)
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf

[A06] NIST FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
http://dx.doi.org/10.6028/NIST.FIPS.202

[A07] NIST FIPS 204: Module-Lattice-Based Digital Signature Standard
https://doi.org/10.6028/NIST.FIPS.204

[A08] NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash.
https://doi.org/10.6028/NIST.SP.800-185

[A09] NIST SP 800-108 Rev 1: Recommendation for Key Derivation Using Pseudorandom Functions
https://doi.org/10.6028/NIST.SP.800-108r1

[A10] NIST SP 800-90A Rev 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators.
https://doi.org/10.6028/NIST.SP.800-90Ar1

[A11] NIST SP800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation.
https://doi.org/10.6028/NIST.SP.800-90B

[A12] NIST SP800-90C: Recommendation for Random Bit Generator (RBG) Constructions (3rd Draft)
https://doi.org/10.6028/NIST.SP.800-90C.3pd

[A13] PMBus Power System Management Protocol, Part I, General Requirements, Transport And Electrical Interface, Revision 1.5

[A14] NIST FIPS 186-5: Digital Signature Standard.
https://doi.org/10.6028/NIST.FIPS.186-5

2.3 Reference Documents

Reference documents have background or supplementary information to this specification. They do not include requirements or specifications that are considered part of this document.

[R01] Cryptographic Algorithm Validation Program
https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/secure-hashing

[R02] Secure Hash Standard Validation System:
https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Algorithm-Validation-Program/documents/shs/SHAVS.pdf

[R03] Secure Hash Algorithm-3 Validation System (SHA3VS):
https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Algorithm-Validation-Program/documents/sha3/sha3vs.pdf

[R04] SP800-90B Entropy Assessment Software:
https://github.com/usnistgov/SP800-90B_EntropyAssessment

[R05] Cryptographic Module Validation Program (CMVP)
https://csrc.nist.gov/Projects/cryptographic-module-validation-program/entropy-validations/announcements-1

[R06] SP800-90B Entropy Source Validation Workshop
https://www.nist.gov/news-events/events/2021/04/sp-800-90b-entropy-source-validation-workshop

[R07] NIST SP800-186: Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters
https://doi.org/10.6028/NIST.SP.800-186

3. Reference Information

3.1 Numerical Formats

All numbers are decimal unless explicitly designated otherwise.

3.1.1 Decimal Numbers

Numbers explicitly identified as decimal are identified with a suffix of “d”.

3.1.2 Floating Point Numbers

Numbers explicitly identified as floating point are identified with a suffix of “f”.

3.1.3 Binary Numbers

Numbers in binary format are indicated by a suffix of “b”.

Unless otherwise indicated, all binary numbers are unsigned. All signed binary numbers are two’s complement.

3.1.4 Hexadecimal Numbers

Numbers in hexadecimal format are indicated by a suffix of “h”.

3.1.5 Examples

255d ⇔ FFh ⇔ 11111111b

175d ⇔ AFh ⇔ 10101111b

1.2f

3.2 Bit And Byte Order

As specified in the SMBus specification

- When data is transmitted, the lowest order byte is sent first and the highest order byte is sent last.
- Within any bytes, the most significant bit (MSB) is sent first and the least significant bit (LSB) is sent last.

3.3 Bit And Field Illustrations

The transmission of bits within a field is illustrated in this section.

In all cases, the least significant bit is indicated as Bit 0, as shown below.

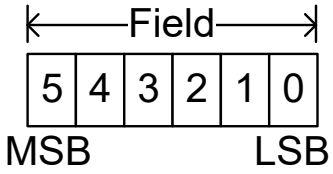


Figure 1. Bit Order Within A Field

This part of the specification describes transactions over PMBus. The symbols used to describe the details of those transactions and protocols are shown in Table 1.

Table 1. Bit And Field Representation For Frame Illustrations

Symbol	Meaning
0	A rectangle with no shading indicates data sent from the host (bus controller) to the target device.
0	A rectangle with a shaded interior indicates data sent from the target device to the host (bus controller).
8 PEC 8 PEC	By default there is a rectangle for each field sent, not for each bit within a field. Typically, the rectangle will have a number over it indicating the number of bits it represents. Shown here is the 8-bit PEC used for verifying the integrity of a transfer. It can be sent by the master or the slave, so it may be found shaded or clear.
S	Start flag. Begins a PMBus transaction.
P	Stop flag Concludes a PMBus transaction.

Symbol	Meaning
Sr	Repeat Start flag Typically used to begin the read portion following the write portion of a PMBus transaction.
1 A 1 A	Acknowledge bit, either from the host (bus controller) to the target, or when shaded from the target back to the host (bus controller). As shown by the number on top, this takes 1 bit time.
1 NA 1 NA	Not Acknowledge bit, either from the host (bus controller) to the target, or when shaded from the target back to the host (bus controller). As shown by the number on top, this takes 1 bit time.

3.4 Abbreviations, Acronyms And Definitions

Table 2. Abbreviations, Acronyms and Definitions Used In This Specification

Term	Definition
ACK	Acknowledge
API	Application Programming Interface
HMAC	Hashed Message Authentication Code
KDF	Key Derivation Function
KMAC	Keccak Message Authentication Code
MAC	Message Authentication Code
NACK	Not Acknowledge
NVM	Non-volatile Memory
OTP	One-time programmable memory (fuses)
PEC	Packet Error Code

Term	Definition
PRoT	Platform Root of Trust
PSK	Pre-shared key

4. General Requirements

4.1 Compliance

The goal of this specification is to define a hardware interface to authenticate that target devices populated on the PMBus are the exact devices with the exact firmware and configuration intended by the platform designer. Additionally, it defines a methodology to help ensure that PMBus targets only accept firmware updates specifically authorized by the platform designer. The goal of this is to prevent PMBus targets from being exploited for the purpose of causing permanent or temporary denial of service or bypassing data protection controls.

Figures demonstrating specific physical implementation, such as a given physical memory size or firmware region arrangement, are made for illustrative purposes only. Internal implementations may vary provided external specification is met.

4.1.1 Commands

To be compliant with the PMBus secure device reference implementation, a PMBus target must implement the commands as described in this specification. If a device accepts a PMBus secure device command, it must execute the associated function as described in PMBus secure device specification (this document).

If a device does not accept a given PMBus secure device command, it must respond per PMBus Part II, Revision 1.5 Section {Invalid or Unsupported Command} [A02].

A device may support non-required Commands or Security Action Requests, including MFR_SPECIFIC commands or request, per the PMBus Part II, Revision 1.5 Specification [A02] unless such commands or requests are prohibited by the supported Application Profile Level.

It is possible to a device to meet PMBus secure device requirements but not utilize the reference-level register implementation defined in this document if and only if provided with the full set of APIs required by the application profile [A03]. This provides high-level software commonality where manufacturer-specific differences in this reference implementation may exist.

4.1.2 Operation

PMBus secure devices must support all required aspects of the profile required for a given security level. For all security levels, the authenticated commands shall be the only method to store to or restore from NVM. When deployed, PMBus STORE and RESTORE commands must be permanently disabled for PMBus secure devices.

At the discretion of the manufacturer and system integrator, devices may be delivered to a system integrator in an “unsecured” state. These devices require the capability of permanently disabling the STORE and RESTORE commands through NVM supported ACCESS_CONTROL functions or firmware updates and is not considered secure until STORE and RESTORE are disabled.

4.2 PMBus Secure Device Application Profile Level

The PMBus Secure Device Application Profile [A03] provides for different levels of Application Profile Support to meet the varying levels of security, flexibility, and cost in the market. High-level descriptions of different security levels profiles are introduced in this section. For full description of each application profile and required functionality, please refer to the application profile document [A03] which will overrule this document in the event of any conflict in required features or API definition.

Security profile levels vary from 0 up to 3, with 0 the least secure and 3 the most secure. Security level 0 provides guidance for password and command access control. Security level 1 enables attestation to validate the measurement of firmware and configuration files active on a controller. It requires a pre-shared key. Security level 2 builds upon level 1 by reusing the attestation mechanism to validate the measurement of an incoming candidate firmware image. Finally, security level 3 uses public key cryptography to validate incoming firmware images and can attest the host sending the request.

4.3 Testing

It is incumbent on PMBus target designers to ensure compliance with this profile. It is incumbent on the system integrators to verify compliance to a profile and the security of their platform root of trust (PRoT) software.

5. Security Actions

As new threats on the supply chain and at runtime emerge, PMBus must play an important role in system security. Running unauthenticated and/or unattested firmware and/or configurations on voltage regulators or other PMBus targets may expose the system to various attacks. By varying voltage supplied to the processor, a malicious voltage regulator (VR) firmware (FW) could jolt the processor out of its normal routine, skipping a security process such as password verification. The platform root-of-trust (PRoT), operating as PMBus host (controller), is leveraged heavily in implementing these security requirements to minimize any un-intended impact to the voltage regulators.

To mitigate against these attacks, this profile recommends employing a 3-part solution:

- **Attestation**: PMBus target device, firmware, and configuration shall be attested using a pre-shared key to the PRoT at every boot cycle. The PRoT can also initiate this attestation request during runtime. The attestation process defined in this profile ensures the firmware and configuration bits of the PMBus target contain the proper non-volatile memory values for this application.
- **NVM Authentication:** This profile provides a mechanism for secure device firmware updates. This mechanism requires firmware to be digitally signed by the system integrator and/or PMBus device manufacturer and verified prior to storing during boot/update cycles by the PRoT. If this method is applied at every boot cycle, this can theoretically enable low-NVM devices. If applied upon request, it can enable PRoT driven updates of PMBus target device firmware or configuration.
- **Access Control:** This is defined between the security application profile and PMBus specification Part II [A02]. Access control can limit writes and reads to potentially problematic PMBus commands, such as those that may alter voltage output or those that may alter device protection thresholds or responses.

The standard defines 3 different profile levels which hit different points of the security vs complexity tradeoff.

In addition to the 3 key security processes, this specification defines additional commands. A major commonality to many of these commands includes attesting, or validating the identity, of the host sending the command. The attestation-of-host capability can enable PMBus commands to have access opened only to specific hosts with knowledge of a pre-shared key, while leaving them closed to an aggressor device which may have gained bus access.

Commands are defined to perform activities associated with firmware updates such as resetting the PMBus target following an update. Maintenance commands such as iterating the secret pre-shared keys based upon hashing a transmitted seed are defined. Finally, status inquiry commands are defined to identify the version of firmware and configuration that is active within the target.

6. Security Memory Architecture

6.1 Communication And Control

PMBus security features are implemented using PMBus commands that read from and write into a primary security memory. Some addresses in that memory serve as a window into larger buffer memories. Other addresses in the primary security memory are used to request and check status of security action requests. The PMBus commands used to interact with the primary memory are:

- SECURITY_BYTE: Reads or writes a byte within PMBus security primary memory
- SECURITY_BLOCK: Reads or writes a block of PMBus security primary memory
- SECURITY_AUTOINCREMENT: Reads or writes a block of buffer memory via a sliding window written through the PMBus security primary memory.

SECURITY_BYTE and SECURITY_BLOCK take as an argument the address at which writing starts within security primary memory space. SECURITY_AUTOINCREMENT always writes starting at security primary memory address 20h, the start of the window into the security buffer memories. The addresses exposed by the window change due to the auto-increment as described in PMBus Part II.

For all three commands, support of PEC instructions is mandatory. The PRoT is required to use PEC when issuing commands.

6.2 Primary Security Memory Map

PMBus security is implemented using two levels of memory. There is a small primary memory that directly interacts with PMBus commands. This PMBus primary memory is used for all external interaction with the PMBus host driving security, often known as the platform root of trust (PRoT). The PMBus secure device primary memory has an 8-bit address space. Single bytes of PMBus secure device primary memory can be written via SMBus WRITE WORD protocol or read via SMBus PROCESS CALL protocol using the SECURITY_BYTE command. Blocks can be written using the SMBus WRITE BLOCK protocol or read via SMBus WRITE-BLOCK READ-BLOCK PROCESS CALL protocol using the SECURITY_BLOCK command. The SECURITY_BYTE and SECURITY_BLOCK commands are specified in the PMBus Part II specification [A02].

Most PMBus targets have firmware and configuration data memories that exceed 256 bytes. The PMBus security specification allows indirect access to that memory via

buffers that can be up to 16MB in size (24-bit address). The indirect access is via a window in the primary memory that can read or write from the buffers. The maximum window length defined in this specification is 32 bytes, but implementations can elect to implement as little as 1-byte maximum window length. The optional PMBus SECURITY_AUTOINCREMENT command can be used to read or write into these buffers more efficiently as it shifts the window location as bytes are written or read.

At least 3 indirectly accessible buffer memory regions must be implemented as shown in Figure 2. These memory buffers may be used to store parameters related to device attestation and authentication or to buffer the firmware contents prior to committing it to NVM or volatile program memory.

In Section 6.2.1 the memory protections that are required to protect against buffer overruns and other similar memory attacks when implementing the architecture of Figure 4 are introduced. PMBus specification Part II [A02] defines commands that implement security memory reads and writes at either the byte level, block level, or block level with auto-incrementing address pointer. Specifically, byte-wide read and write commands to security primary memory are described in SECURITY_BYTE command [A02]. Block read and write commands to security primary memory are described in SECURITY_BLOCK command [A02]. The SECURITY_AUTOINCREMENT command [A02] is used to burst to or from the indirectly accessible buffer memories. Finally, the remaining sections of the chapter indicate how to induce actions based upon the data in primary memory and/or the indirectly accessible buffers.

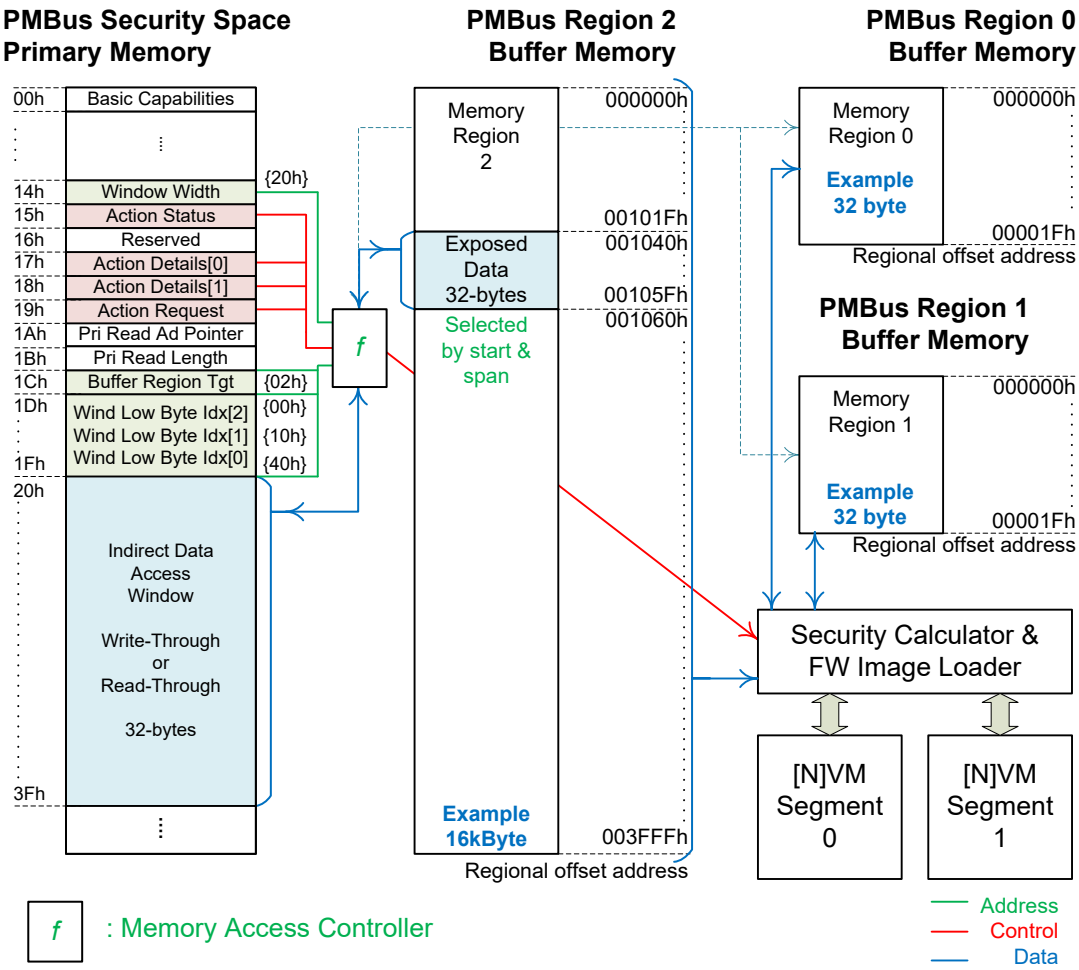


Figure 2: PMBus Security Device Memory Architecture

6.2.1 Security Primary Memory Protection Mechanisms

PMBus security primary memory covers 8-bit addresses starting from address 00h and going up to at least 20h, with optional registers up to 3Fh defined for block transactions to/from the buffer memories and optional registers above D0h reserved for manufacturer usage. If a read or write request transacts with a primary memory address outside of the implemented primary addresses, it is required to perform the following actions shown in Table 3.

Table 3. Memory Address Out-of-Bounds Action

• Raise a PMBus Status_CML "Invalid or Unsupported Data Fault" • For writes, discard any data to be written • For reads, return FFh for the unsupported address

In many implementations, the PMBus security primary address space may be located within a larger physical memory. It is required that implementations implement memory address bounding and buffer overrun protections such that an attacker is unable to read or write from regions outside of the primary address space using the PMBus security commands. Furthermore, the PMBus security command shall be the

only commands capable of reading or writing from the PMBus secure device memory. No other commands shall be permitted access.

Figure 3 illustrates one such potential attack, where a PMBus security primary memory region command with address greater than the length of the implemented primary memory region is used to read items from the next higher seated item in physical memory. Attempts to read or write into these out-of-bound addresses must be responded to per Table 3.

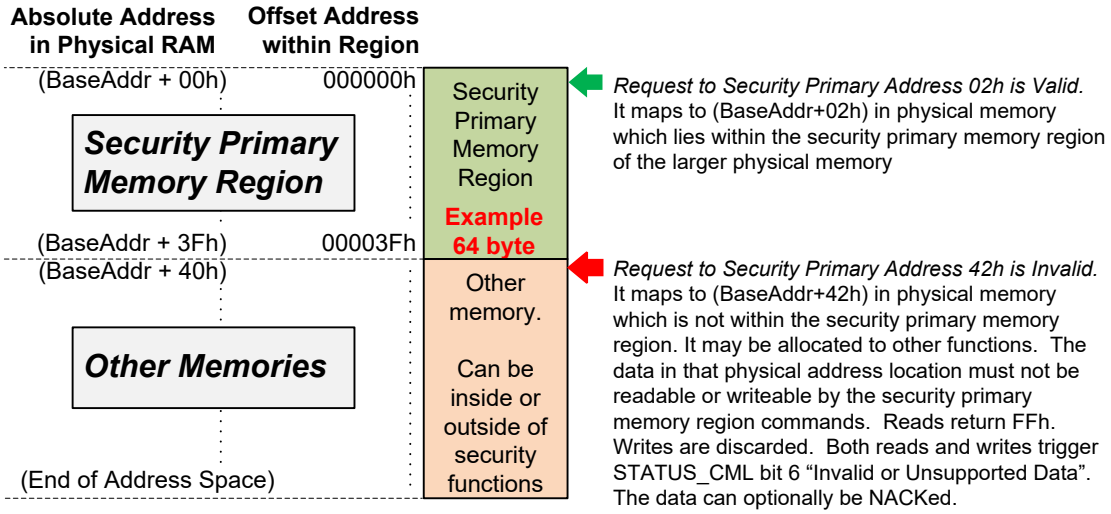


Figure 3. Security Memory Overrun Protection Within Implemented Devices

The PMBus buffer region memory addresses must be similarly bounded. In this case, window low-byte index, and span must point to expected locations in the selected buffer memory region. If the referenced memory address is out-of-bounds from the expected buffer region size, then the write or request shall be denied as described in Table 3. No command except those described in the PMBus secure device specification shall be able to read or write the buffer memories. Checks on the validity of the window low byte index are performed at the time data is written to or read from the “window data” located at 20h to 3Fh in primary memory space. This allows overwriting the buffer region pointer or buffer window low byte index pointer without worrying about creating an invalid address from a combination of prior and next values as byte-by-byte is written.

The valid address checks upon write or read are required to make sure that there is no ability to implement a “beyond array bounds” buffer-overflow attack on items that are in physical address space beyond the end of any PMBus secure device memory space. Similarly, PMBus secure device specification requires protection from other systems sharing the same physical memory.

If a buffer region is selected that does not physically exist within a device, then the PMBus target can elect to handle it per Table 3 or by discarding writes and returning a fixed value for all reads. Operations to non-existent buffer regions shall not overwrite or expose any data. This would be an operation that is not expected in a real application scenario where the attestation and authentication software is provided by the system vendor and PMBus target manufacturer. More information about buffer region address validity is described in Section 6.4.3.

To support PEC operation, all write data to the target must not be committed until the stop-flag has been received to allow verifying the PEC before committing the data to memory. PEC operation for reads is handled by the PMBus host, typically located on the platform root of trust (PRoT).

6.2.2 PMBus Primary Memory Region Map

The PMBus security device primary memory's register map is shown in Table 4.

Table 4. PMBus Secure Device Primary Memory Register Map

Byte Address	Memory Contents	Reg Type	Section
00h	Basic Capabilities	RO	6.2.3
01h	Reserved	N/A	-
02h	Security Profile [0]	RO	12.1
03h	Security Profile [1]	RO	12.1
04h	Firmware Version [0]	RO	12.1.2
05h	Firmware Version [1]	RO	12.1.2
06h	Non-volatile And Alert Capability	RO	12.2.1
07h	PSK Iteration Algorithm Support [0]	RO	8.2.1
08h	PSK Iteration Algorithm Support [1]	RO	8.2.1
09h	PSK Iteration Algorithm Support [2]	RO	8.2.1
0Ah	Attestation Algorithm Support [0]	RO	8.4.1.1
0Bh	Attestation Algorithm Support [1]	RO	8.4.1.1
0Ch	Attestation Algorithm Support [2]	RO	8.4.1.1
0Dh	Reserved	N/A	-
0Eh	Reserved	N/A	-
0Fh	Reserved	N/A	-
10h	PMBus Block Write - Max Length	RO	6.3.2
11h	PMBus Block Write - Min Length	RO	6.3.2
12h	PMBus Block Read - Max Length	RO	6.3.2
13h	PMBus Block Read - Min Length	RO	6.3.2
14h	Buffer Region Window Width (Bytes)	RO	6.3.1
15h	Security Action Status	RO	7.2
16h	Reserved	N/A	-
17h	Security Action Details [0]	RW	7.2
18h	Security Action Details [1]	RW	7.2
19h	Security Action Request	RW	7.2

Byte Address	Memory Contents	Reg Type	Section
1Ah	Primary Memory Read Address Pointer	RW †	6.3.2
1Bh	Read Length	RW †	6.3.3
1Ch	Buffer Region Target	RW	6.3.1
1Dh	Window Low Byte Index **[*2*]**	RW	6.3.1
1Eh	Window Low Byte Index **[*1*]**	RW	6.3.1
1Fh	Window Low Byte Index **[*0*]**	RW	6.3.1
20h	Window Data [0]	RW	6.3.1
21h	Window Data [1]	RW	6.3.1
22h	Window Data [2]	RW	6.3.1
23h	Window Data [3]	RW	6.3.1
24h	Window Data [4]	RW	6.3.1
25h	Window Data [5]	RW	6.3.1
26h	Window Data [6]	RW	6.3.1
27h	Window Data [7]	RW	6.3.1
28h	Window Data [8]	RW	6.3.1
29h	Window Data [9]	RW	6.3.1
2Ah	Window Data [10]	RW	6.3.1
2Bh	Window Data [11]	RW	6.3.1
2Ch	Window Data [12]	RW	6.3.1
2Dh	Window Data [13]	RW	6.3.1
2Eh	Window Data [14]	RW	6.3.1
2Fh	Window Data [15]	RW	6.3.1
30h	Window Data [16]	RW	6.3.1
31h	Window Data [17]	RW	6.3.1
32h	Window Data [18]	RW	6.3.1
33h	Window Data [19]	RW	6.3.1
34h	Window Data [20]	RW	6.3.1
35h	Window Data [21]	RW	6.3.1
36h	Window Data [22]	RW	6.3.1
37h	Window Data [23]	RW	6.3.1
38h	Window Data [24]	RW	6.3.1
39h	Window Data [25]	RW	6.3.1

Byte Address	Memory Contents	Reg Type	Section
3Ah	Window Data [26]	RW	6.3.1
3Bh	Window Data [27]	RW	6.3.1
3Ch	Window Data [28]	RW	6.3.1
3Dh	Window Data [29]	RW	6.3.1
3Eh	Window Data [30]	RW	6.3.1
3Fh	Window Data [31]	RW	6.3.1
40h – CFh	Reserved	-	-
D0h – FFh	Reserved for manufacturer use	-	-

Notes:

- The order of the burst window low-byte-index is reversed compared to other fields. For the low byte, this allows repositing the window by an offset amount without needing to re-write the whole 3-byte address.
- The read length is only required to be stored if a target device implements read/write-block with address auto-increment. Read address pointer need not be retained outside the context of the in-progress read transaction.

6.2.3 PMBus Secure Device Basic Capabilities Register

This register at primary memory address 00h is often the first byte read by a system after booting. It indicates which commands can be used to access the PMBus security primary memory, which commands can be used to access the PMBus security indirectly-accessible buffer memories and provides a starting size for block read and write operations. The bits to indicate these capabilities are shown in Table 5. Knowing that a block size of 4 and that the SECURITY_BLOCK command are both available allows the PRoT to read things such as security profile level and firmware revision with a single block read, speeding initial setup. Similarly, the PMBus block read/write minimum and maximum block sizes can be read in one transaction.

Table 5. PMBus Secure Device Communications Capabilities Register

Primary Memory Address	Bit	Bit description
Primary Address 00h “Basic Capabilities”	[7]	MSB , Reserved, set to 0.
	[6]	Reserved, set to 0.
	[5]	Reserved, set to 0.
	[4]	Reserved, set to 0.

Primary Memory Address	Bit	Bit description
	[3]	If 1, SMBus block size of 4 for read and write transactions is supported.. Enables: • 4 data byte SECURITY_AUTOINCREMENT command writes and reads, • 3 data byte SECURITY_BLOCK command writes, • 4 data byte SECURITY_BLOCK command reads if bit [2] below is also supported If 0, SMBus block size of 4 is not supported.
	[2]	If 1, SMBus Block size of 3 for writes is supported. *Enables:* • 2 data byte SECURITY_BLOCK command block writes – This is important as this is done implicitly as part of the WRITE-BLOCK READ-BLOCK PROCESS-CALL employed for SECURITY_BLOCK command block reads. • 3 data byte SECURITY_AUTOINCREMENT command writes If 0, SMBus block size of 3 is not supported.
	[1]	If 1, then the SECURITY_AUTOINCREMENT command is supported If 0, the SECURITY_AUTOINCREMENT command is not supported.
	[0] LSB	If 1, then the SECURITY_BLOCK command is supported If 0, the SECURITY_BLOCK command is not supported.

6.3 Buffer Regions: Indirect Access to Large Memory

6.3.1 Indirect Memory Access Methods

The PMBus secure device primary memory is limited to 256 bytes due to its 8-bit addressing of 1-byte long data fields. Firmware and configuration sizes often exceed 256 bytes. To allow updating and programming firmware, the primary memory provides a means to indirectly access second-level buffer regions of up to 16 megabytes in size. This indirect read and write scheme is used to transfer larger data items including:

- Security nonces,
- MAC results,
- Firmware Updates.

Within each of these buffer regions there are up to 24-bits of address space available. Although all 24-bits of byte pointer registers must be accessible to read or write commands, the values of unused address bits can be fixed at 0 within the pointer

registers and need not contain actual storage elements. The buffer memory regions themselves need not implement all 24-bits of addressing. These addresses are written or read in blocks of up to 32-bytes through the PMBus secure device primary memory. Example purposes for which the indirectly accessed buffers regions can be used include:

- Staging requests to update firmware or configuration data segments of either NVM or volatile storage,
- Transmitting attestation nonces or PSK-iteration seeds to the PMBus target,
- Transmitting/retrieving expected MAC results during attestation processes,
- Transmitting password and or checksum to the target in security level 1 authentication.

The buffer regions are accessed via a window that can be written or read through via the PMBus security primary memory space. This window must be configured via three attributes in primary memory, shown in Table 6.

Table 6. Buffer Window Configuration Registers

Address Within Primary Memory	Description
Primary Address 1Ch “Buffer Region Select”	Selects the PMBus buffer region into which the PMBus window data registers provide a window.
Primary Addresses {1Dh, 1Eh, 1Fh} “Window Low Byte Index [2:0]”	Specifies the address within the buffer region memory that is accessed via Window Data [0], located at PMBus secure device primary memory address 20h. If the window width is greater than 1, then the next higher address will be at 21h, and so on until the buffer window region width is met. If all address bits are not needed to access available data, it is allowable for those unnecessary bits to be tied to zero rather than containing actual storage elements. Even without physical storage elements, all three of these registers need to respond to read and write commands.
Primary Address 14h “Window Width”	Specifies the data width (length) in bytes of the buffer region that is accessible in the primary memory at one time. The valid window addresses will be from primary address 20h to (20h+windowWidth-1). It writes and reads through to addresses within the buffer window from (WindowLowByteIndex) up to (WindowLowByteIndex + BufferRegionWidowWidth-1). Note: This data is read only and is used to determine the amount of data to be fed via the window.

The PMBus secure device primary memory at addresses from 20h to 3Fh provides the window for writing or reading through the primary register space into the buffer regions. Address validity is checked when these registers are read or written. Reads or writes to invalid registers are handled per Table 3.

Table 7. Primary Write-Through And Read-Through Into Region Buffers

Address In Primary Memory	Register Name	Address Within Buffer Region	Implemented If:
20h	Window Data [0]	Window Low Byte Index + 00h	Always
21h	Window Data [1]	Window Low Byte Index + 01h	Window Width ≥ 02h
22h	Window Data [2]	Window Low Byte Index + 02h	Window Width ≥ 03h
23h	Window Data [3]	Window Low Byte Index + 03h	Window Width ≥ 04h
24h	Window Data [4]	Window Low Byte Index + 04h	Window Width ≥ 05h
25h	Window Data [5]	Window Low Byte Index + 05h	Window Width ≥ 06h
26h	Window Data [6]	Window Low Byte Index + 06h	Window Width ≥ 07h
27h	Window Data [7]	Window Low Byte Index + 07h	Window Width ≥ 08h
28h	Window Data [8]	Window Low Byte Index + 08h	Window Width ≥ 09h
29h	Window Data [9]	Window Low Byte Index + 09h	Window Width ≥ 0Ah
2Ah	Window Data [10]	Window Low Byte Index + 0Ah	Window Width ≥ 0Bh
2Bh	Window Data [11]	Window Low Byte Index + 0Bh	Window Width ≥ 0Ch
2Ch	Window Data [12]	Window Low Byte Index + 0Ch	Window Width ≥ 0Dh
2Dh	Window Data [13]	Window Low Byte Index + 0Dh	Window Width ≥ 0Eh
2Eh	Window Data [14]	Window Low Byte Index + 0Eh	Window Width ≥ 0Fh
2Fh	Window Data [15]	Window Low Byte Index + 0Fh	Window Width ≥ 10h
30h	Window Data [16]	Window Low Byte Index + 10h	Window Width ≥ 11h
31h	Window Data [17]	Window Low Byte Index + 11h	Window Width ≥ 12h
32h	Window Data [18]	Window Low Byte Index + 12h	Window Width ≥ 13h
33h	Window Data [19]	Window Low Byte Index + 13h	Window Width ≥ 14h
34h	Window Data [20]	Window Low Byte Index + 14h	Window Width ≥ 15h
35h	Window Data [21]	Window Low Byte Index + 15h	Window Width ≥ 16h
36h	Window Data [22]	Window Low Byte Index + 16h	Window Width ≥ 17h
37h	Window Data [23]	Window Low Byte Index + 17h	Window Width ≥ 18h
38h	Window Data [24]	Window Low Byte Index + 18h	Window Width ≥ 19h
39h	Window Data [25]	Window Low Byte Index + 19h	Window Width ≥ 1Ah
3Ah	Window Data [26]	Window Low Byte Index + 1Ah	Window Width ≥ 1Bh
3Bh	Window Data [27]	Window Low Byte Index + 1Bh	Window Width ≥ 1Ch
3Ch	Window Data [28]	Window Low Byte Index + 1Ch	Window Width ≥ 1Dh
3Dh	Window Data [29]	Window Low Byte Index + 1Dh	Window Width ≥ 1Eh

Address In Primary Memory	Register Name	Address Within Buffer Region	Implemented If:
3Eh	Window Data [30]	Window Low Byte Index + 1Eh	Window Width ≥ 1Fh
3Fh	Window Data [31]	Window Low Byte Index + 1Fh	Window Width ≥ 20h

An optimized command has been defined to allow block writes and block reads through the window to/from the buffer memory. The goal of this command is to maximize the efficiency of data transfers to and from the buffer memory regions by maximizing the percentage of bytes transferred that contain useful data rather than addressing and flow control.

This block read or write command always commences at PMBus primary security memory "Window Data [0]" located at address 20h. This location contains the lowest addressed data byte from the present window into buffer memory as selected per Table 6 by the buffer select and low-byte index fields located from 1Ch to 1Fh of primary address space. Refer to the PMBus Part II specification [A02] for the operation of the PMBus security buffer read and write commands:

- SECURITY_BYTE
- SECURITY_BLOCK
  SECURITY_AUTOINCREMENT

6.3.2 PMBus Block Transaction Size Support

There are PMBus secure device registers that state the minimum and maximum lengths uniquely for block reads and block writes. The registers specifying these lengths are shown in Table 8. For devices only supporting a single block size, the minimum and maximum lengths would be identical. At the present time Linux PMBus drivers allow a maximum block size of 32 bytes, and this will serve as an additional maximum length constraint on PMBus burst size in most applications. A maximum block size of 0 indicates that block transactions of that type are not supported.

Table 8. PMBus Block Length Bounds

PMBus Secure Device Primary Memory Address	Register Descriptions	Access Type
Primary Address 10h "Block Write - Max Length"	The maximum SMBus write block size permitted by this target device for write-block transactions. A value of zero indicates block writes are not supported. *Note*: For SECURITY_BLOCK command writes, the number of transmitted data bytes is one less than the SMBus block length as one byte is used to transmit the address.	Read-only

PMBus Secure Device Primary Memory Address	Register Descriptions	Access Type
Primary Address 11h “Block Write - Min Length”	The minimum block size permitted by this target device for write-block transactions. A value of zero indicates block writes are not supported *Note*: For SECURITY_BLOCK command writes, the number of transmitted data bytes is one less than the SMBus block length as one byte is used to transmit the address.	Read-only
Primary Address 12h “Block Read - Max Length”	The maximum block size permitted by this target device for the read portion of SMBus write-block, read-block process-call transactions. A value of zero indicates write-block read-block process-calls are not supported	Read-only
Primary Address 13h “Block Read - Min Length”	The minimum block size permitted by this target device for the read portion of SMBus write-block, read-block process-call transactions. A value of zero indicates write-block read-block process-calls are not supported.	Read-only

Protection against out-of-bounds data reads and writes must be implemented as per single-byte commands, with invalid addressing faults reported per Table 3. Memory Address Out-of-Bounds Action.

One special case exists regarding a common block size. If the registers 10h, 11h, 12h, and 13h all contain the same value, then it is allowed for target hardware to require that all block-read or block-write commands start at an integer multiple of that common block size. Because the block read/write with auto-increment commands start at register 20h, it requires that the common block size be an exact integer divisor of 20h (32) to ease implementation by allowing the least significant bits to be directly used as part of the address within a buffer memory region.

Software / Driver Guidance: Drivers intending to support any generic PMBus device need to be aware of this window low-byte index restriction when implementing block read and writes via PMBus.

6.3.3 Block Read Size Support

For PMBus block transactions, the PMBus target must be programmed in advance with a value indicating the number of data bytes from the memory map to be returned in the read portion of a SECURITY_BLOCK or SECURITY_AUTOINCREMENT read.

Table 9. PMBus Secure Device Primary Memory Address 1Bh Contents

PMBus Secure Device Primary Memory Address	Register Descriptions	Access Type
Primary Address 1Bh	Block Size to Read Back for Read Transactions Program the target so it knows how much data to reply within each block read. This value, *N*, must be constrained by: • *N* ≥ Min Block Read Size (reg 12h) • *N* ≤ Max Block Read Size (reg 13h) • *N* ≤ Window Size (reg 14h)	Write

6.4 Error Handling Of Pmbus Security Commands

6.4.1 PEC Error Handling

For write transactions, the PEC is sent from the PMBus controller (PRoT) to the PMBus target devices. The PMBus target validates the PEC data against the received data stream. If the PMBus target detects a mismatch with the transmitted PEC, then it:

1. Sets the STATUS_CML bit [5] within the target and
2. Discards the transaction so it has no effect on the system.

For read transactions, the PEC is sent from the PMBus target device back to the PMBus controller. In this case, the PRoT must perform the calculation to validate the PEC returned by the target. In these read scenarios, it is the responsibility of the PMBus controller (PRoT) to discard the bad read data and resolve any potential issues. This includes resetting the “window low byte index” if it was affected by a read command with auto-increment during the failed-PEC transaction.

6.4.2 Unsupported Block Length Selection

Writes of block lengths not supported by the target shall trigger STATUS_CML bit [6] invalid data. A well implemented PRoT shall not issue any of these transactions because this specification defines means for the PRoT to determine supported block sizes.

6.4.3 Writes To Invalid Buffer Region Addresses Locations

Writes to locations within an implemented buffer region but where the window low-byte index is not valid must trigger STATUS_CML bit [6], invalid data. If the buffer region is not valid, then the manufacturer can choose to either trigger a STATUS_CML or just discard the data for writes and return a fixed value for all reads.

If the window low-byte index is a valid memory address within a given buffer region, but the block length would write past the end of the memory, then (1) the write is connected without triggering a STATUS_CML and (2) writes to valid addresses are committed, and (3) writes to addresses past the highest valid address in the memory region are discarded without notice. It is important that these writes past the end be discarded without overwriting the contents of anything that may be implemented in a physically higher memory address that is not part of the targeted memory region. A write past end of the targeted memory buffer region that is committed to RAM would be a buffer overflow attack and must be impossible by design implementation.

7. Security Action Request Engine

7.1 Introduction

The buffer regions defined in Section 6.3 are utilized by the PMBus target for three primary functions with respect to security:

- Attestation: Proving the target is the claimed device with the expected firmware
- Authentication: Proving that an updated firmware or configuration is current and expected.
- Conveyance: Facilitating the transfer of new firmware or configuration data.

When considering these purposes, authentication and conveyance are intrinsically paired. A new candidate firmware once uploaded to the target must be authenticated before being applied. Attestation intrinsically cannot be done simultaneously to authentication for a given region, attesting to a current firmware while it is being overwritten is an unexpected and unsupported case.

Minor functions exist that also utilize these PMBus buffer regions. These minor functions, including pre-shared-key updates and locks, shall also be used exclusively for attestation and authentication functions.

This mutual exclusion of different functions allows the buffer regions themselves to be re-used, saving implementation resources on the PMBus target. Table 10 shows PMBus security actions and corresponding opcodes, all of which must be started and run to completion exclusive of other PMBus security commands. It also shows a high-level description of what data is written to or read from buffer regions for each opcode. Full details of the data requirements are given in the indicated sections. Illustrated in Figure 4, the usage of buffer regions allows reutilizing memory for different functions that do not run simultaneously.

Upon a write to any byte of buffer region outside of the security action process outlined in Section 7.2, the “security action status” register is cleared to “Idle” with value indicated in Table 12. This indicates that the results for any process attempting to read a result from the security memory buffer regions may be reading results altered by something other than the target itself.

If a security action request is submitted with an unknown command, then “security action status” register at PMBus secure device primary memory address 15h to “Failed, Operation not supported” result per Table 12. Command completion can either be detected by polling the security action status register or, if enabled via security action request command 0Bh, the SMBALERT status bit as per Section 12.2.

Table 10. PMBus Secure Device Security Actions And Buffer Region Usage[1, 2]

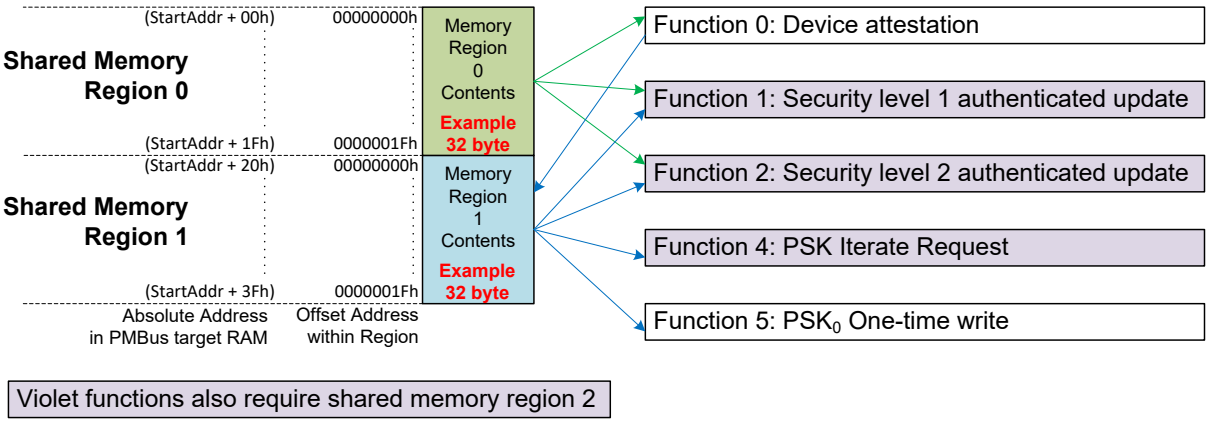
Opcode	Function	Section	Region 0	Mode	Region 1	Mode	Region 2	Mode
00h	Device attestation	8.4	MAC result	Read	Nonce	Write	-	-
01h	Sec Level 1 authenticated update	10.2	Password	Write	Checksum	Write	Fw/Cfg	Write
02h	Sec level 2 authenticated update	10.2	Exp MAC	Write	Nonce	Write	Fw/Cfg	Write

Opcode	Function	Section	Region 0	Mode	Region 1	Mode	Region 2	Mode
03h	Sec level 3 authenticated update	10.2	-	-	-	-	Fw/Cfg	Write
04h	PSK iterate request[3] Nonce can be PRoT or target generated.	8.2.2	Exp MAC	Write	Nonce	Write or Hold	Seed	Write
05h	PSK_0 (Initial PSK) one-time write	8.1.3	-	-	PSK_0 value	Write	-	-
06h	PSK lock with target-generated nonce[4, 5]	8.2.4	Exp MAC	Write	Nonce	Hold	-	-
07h	Power-On Reset while output off	10.4.2	-	-	-	-	-	-
08h	Firmware Fetch[4]	10.5.1	-	-	-	-	Fw/Cfg	Read
09h	Firmware Fetch Lockout [4, 5]	10.5.2	Exp MAC	Write	Nonce	Hold	-	-
0Ah	Power-On Reset Lockout [4, 5]	10.4.1	Exp MAC	Write	Nonce	Hold	-	-
0Bh	PMBus Status alert triggering [4, 5]	12.2	Exp MAC	Write	Nonce	Hold	Config	Write
0Ch	Power-On Reset (Anytime) [4, 5]	10.4.3	Exp MAC	Write	Nonce	Hold	-	-
0Dh	PAGE_PLUS_READ/ PAGE_PLUS_WRITE with host attested [4, 5]	12.4	MAC	R/W	Nonce	R/W	Cmd	Write
0Eh	Secured Access Control Action [4, 5]	11.1	Exp MAC	Write	Nonce	Hold	Config	R/W
0Fh	Fetch Nonce[6]	9.2	-	-	Nonce	Read	-	-
10h	PSK lock forever (NVM) with PRoT generated nonce[7]	8.2.3	Exp MAC	Write	Nonce	Write	-	-
11h-5Fh	Reserved	-	-	-	-	-	-	-
60h-7Eh	Reserved for Manufacturer	-	-	-	-	-	-	-
7Fh	Reserved	-	-	-	-	-	-	-
≥80h: Unavailable. The most significant being set is used to indicate the start of sending a security action request for the operation specified by the lower 7-bits.								

Notes on Table 10:

1. All writeable buffer regions are readable to allow validating memory access.
2. Only one of the three commands (1-3) shall be active on a product based on the targeted security level.
3. These commands scan using either a PRoT or target generated nonce.

4. These commands are optional and driven by the needs of a particular product.
5. These commands require attestation of the controller identity. They must be executed following a “Fetch Nonce” command 0Fh. After their execution, the target must invalidate its generated nonce after each attempt of these commands. This is done using the steps of Section 9.
6. The fetch target nonce command itself as mentioned in Note 5. Typically used at security level 3.
7. Optional if coverage can be provided in NVM / firmware authenticated level 2 or 3 update, if firmware always locks out PSK iterate once shipped, or if PSK cannot be iterated. Otherwise, this feature is required.



Memory region sizes are examples. Requirements vary with the attestation & authentication algorithms utilized.

Figure 4. PMBus Security Buffer Region Sharing By Security Actions 0 To 5

7.2 Security Action Request Process And Status Check

Section 7.1 and Table 10 lists the security actions provided for by the PMBus secure device including PMBus target device attestation, authenticated updates, secured access control action, and related functions. Table 10 indicates which shared buffer regions must be preloaded with specific data for each function prior to the invocation of this command. In this section, the mechanism used to set action options or details, request the action, and to determine successful failure or completion is described.

To call a specific function, the PRoT must first declare that it is starting a security action request. It does this by writing the security action request register with the bitwise-OR of the request opcode from Table 10 and 80h. If an operation is already being executed due to a prior security action request being submitted (at step A8 of Figure 5), the write to the security action request register would trigger a STATUS_CML.Invalid_Data error. Declaring the action request being transmitted, when combined with the strict upload ordering requirements listed in this section, may result in the microcontroller being able to start processing some portions of the request speculatively as data becomes available rather than waiting for all required data to be written.

Next, it must write the “security action details” registers, two bytes which parameterize the behavior taken by the action request. Security action details registers can never be skipped, they must be written with details byte [0] written before details byte [1].

The PRoT must then load its prerequisite data into the three buffer regions using the buffer window described in Section 6.3 in the specific order of region 2, then region 1, then region 0. Each region is written starting at address 0 and going up until all required bytes are written. If writing a buffer region is not required for a given request, then the buffer region may be skipped. An overview of the prerequisite data is in the section with the full specification of each function as indicated by Table 10. Ordering is enforced only for writes to the security action details, security action request, and buffer regions. Writes to unrelated registers or reads are not involved in the order checking process.

The process is summarized in Figure 5.

7.3 Security Action Request Exception Handling

If a new security action request with the most significant bit asserted indicating the start of a new action request is received prior to the reception of the security action start request (step A8 of Figure 5), then the not-yet-started security action request is aborted and a new action request sequence starts.

If there is an implementation risk that a requested security action may fault and freeze prior to completion, the PMBus target shall implement a watchdog timer at the designer's choice duration to allow subsequent security actions to be resumed

After a security action request has been issued, but prior to the completion of the request, the target will disallow writes to the buffer regions, to security action details, and security action request registers. Those write attempts will be met with a STATUS_CML invalid data error.

7.4 Security Action Request Behavior Required By PMBus Secure Device Host

- *The PMBus host must properly obey SMBus arbitration.*

  The host must observe its SDA line as it transmits data on the open-drain bus. If it transmits a "1" but observes "0" it shall surrender bus control to the device pulling the bus to "0". After the bus is free, it must retry the aborted transaction.
- Any packet aborted due to arbitration must be retried without change.

  If an attacker managed to alter the data and PEC of a transmitted packet in such a manner that the packet passed PEC checks, the target would recognize the unexpected second valid write to the same data location during the security action request would be unexpected and indicate the presence of a possible attack.
- The PMBus host must utilize PEC. If a PEC error occurs and PEC state not checked by PRoT every transaction, the process must begin again at step 1. If PEC is checked after every transaction, then the most recent write can be repeated.

  Provides some protection against corruption by introducing a quick transient seen only at the target device. Note: Full period bit swaps from 1 to 0 should be seen at the target as bit swaps.
- Security Action Requests must follow the specific sequence shown in Table 11.

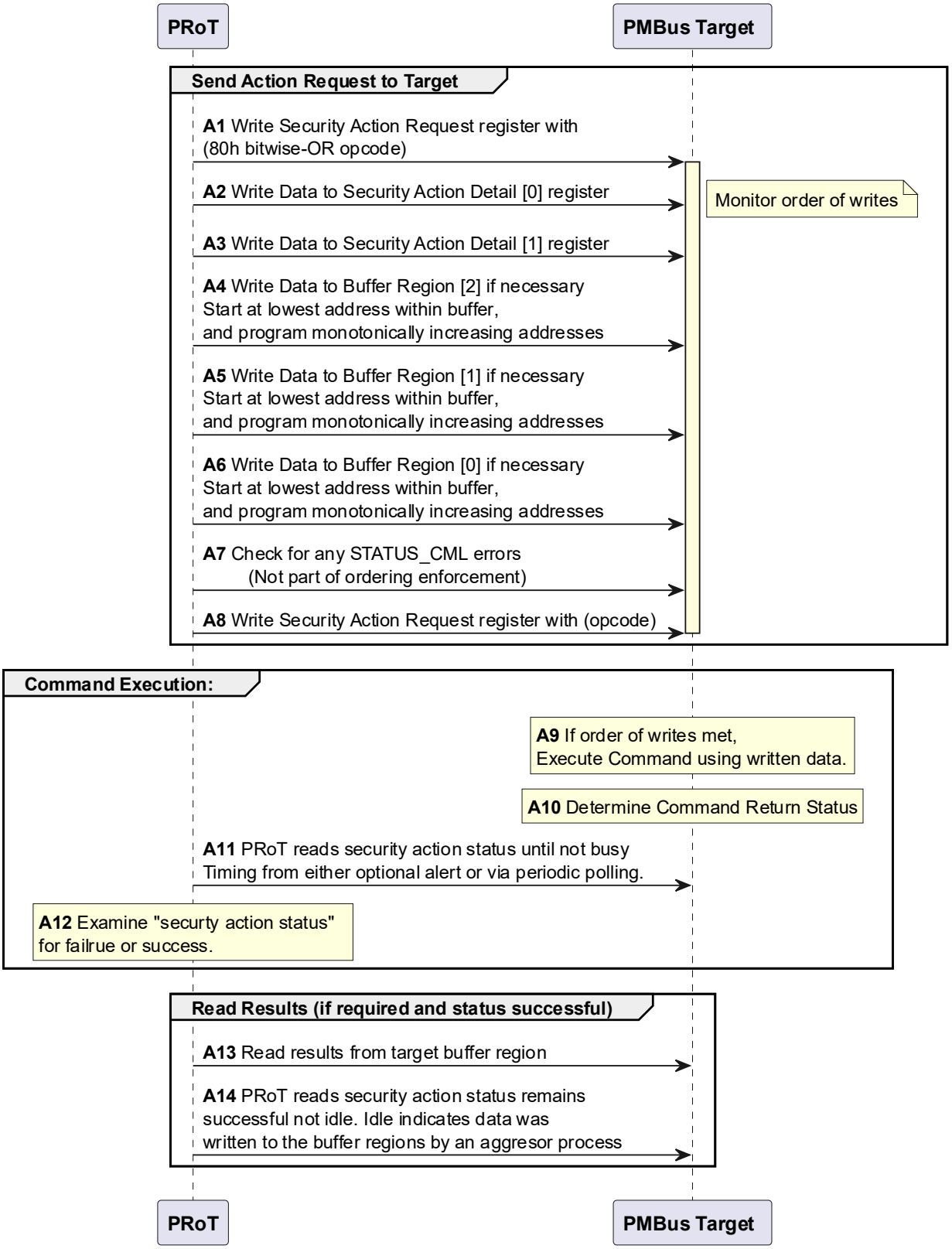


Figure 5. PMBus Security Action Request

Table 11. Security Action Request Process

Step	Instruction
1	Clear the STATUS_CML PEC error bit.
2	The PMBus host must write the security action request register to inform the target that a request is being sent, shown in Figure 5 as step A1. The value written is the opcode for the action desired bitwise-OR'ed with 80h. The assertion of the most-significant bit tells the target this is the beginning of an action request sequence, and the opcode informs the action that will be requested once all data is present. This serves two purposes: It prevents an attacker from issuing a different security action request opcode after the authorized host has staged the buffer regions and security action details attributes with parameters intended for the intended opcode. Knowing the action request that is looming may allow the target to begin required calculations to begin pre-emptively on the first arriving data while other data is still being send.
3	Writes PMBus primary address 17h security action details [0] register, shown in Figure 5 as step A2. Required even if value is reserved to clear permissions for future usage.
4	Writes PMBus primary address 18h security action details [1] register, shown in Figure 5 as step A3. Required even if value is reserved to clear permissions for future usage.
5	If the requested security action requires writes to PMBus secure device buffer region [2], then write to that data buffer with monotonically increasing address order. If no data is required, this can be skipped. Shown in Figure 5 as step A4.
6	If the requested security action requires writes to PMBus secure device buffer region [1], then write to that data buffer with monotonically increasing address order. If no data is required, this can be skipped. Shown in Figure 5 as step A5.
7	If the requested security action requires writes to PMBus secure device buffer region [0], then write to that data buffer with monotonically increasing address order. If no data is required, this can be skipped. Shown in Figure 5 as step A6. Buffer region [0] is the last to be written as it contains the expected MAC used for PRoT attestation. By forcing the expected MAC to be the last item written, a a sophisticated attacker cannot modify other command or buffer fields to produce the desired MAC result.
8	Reads the STATUS_CML to ensure no PEC error has occurred. If a PEC error has occurred, this process must be restarted at step 1 prior to issuing the command. This PEC error check could be done after each command via more frequent polling or using the SMBLERT signal. If done after every command then the last command can be reissued instead of starting from step #1. Shown in Figure 5 as step A7.

Step	Instruction
9	Writes the security action opcode to the PMBus primary address 19h. This starts the target processing the security action. The security action will not allow writes until the operation has completed. If the security action requested does not match the lower 7-bits from step 2 but with the leading bit de-asserted, then the command shall be rejected. This step corresponds to Figure 5, step A8.
10	Reads the security action status register to determine if the operation has completed successfully, failed, or remains in progress. The time at which this read is issued can either be by periodic polling or using an interrupt mechanism. Shown in Figure 5 as step A9.
10	If the “security action status” register read indicates a successfully completed transaction and any data is returned to the buffer regions for that opcode. Shown in Figure 5 as step A12.
11	If “security action status” is successful, go read that yielded data from the appropriate buffer region(s). Shown in Figure 5 as step A13.
12	After the read, re-read the “security action status” register to check that it remains at “successful” and not at “idle”. If “idle” is indicated, it is possible that a process overwrote the data read out from the buffer region during the readback and that the data may not be the authentic output of the security action request. Shown in Figure 5 as step A14.

7.5 Security Action Request Behavior Required By The Target

The PMBus target must support the following behaviors:

- The target must support PEC and reporting PEC failures via the STATUS_CML register bit [5].
- The target must enforce the strict security action request sequencing of Section 7.4. An example pseudo-code of such an engine is shown in **Error! Reference source not found.**.
- Upon any write to any PMBus secure device buffer region that contains data to be read resulting from a security action request, then the “security action status” register must be cleared to “idle” per Table 12. This allows a PRoT to ensure that any data it has read is the output resulting from a security action request and not a buffer write by an aggressor process. The PRoT ensures this by reading the “security action status” register and making sure it remains indicating a successful outcome.
- PMBus security operations invoked by these registers cannot be pipelined. One action must be completed with either success or failure before another can be requested. After a new “security action request” opcode action is made, the target will respond to another action request with STATUS_CML bit “Invalid Data” until the security operation status indicates a completion, with either a success or failure result yielded per Table 12.
- The target must maintain the integrity of all security action request, security action details, and security buffer region data throughout the execution of a security action. This includes from the commencement of the command execution, induced by the write of security action status with the most-significant-bit equal to 0, to the completion of action execution, indicated by the security action status update. A

target may do this by preventing writes to those data fields during command execution or by copying the values to another memory for execution.

Pseudo-code for the enforcement of the ordering of a Security Action Request is shown in Appendix I.

7.6 Security Action Status Register

The security action status register reflects if an operation was successful or not. When the execution of a new security action request is started, the PMBus operation updates the security action status register. That register indicates the state of the action being requested. It is polled by the bus controller to determine when the requested operation has completed. Alternately, an interrupt mechanism has been defined in Section 12.2 that allows notifying the bus controller that a status is available. This register is in PMBus secure-device primary memory at address 15h.

When the security action is requested by the bus controller per step A8 of Figure 5, the security action status will transition to “In Progress” value as indicated in Table 12.

After submitting a security action request, a PMBus host controller can query the PMBus target “security action status” register to tell if a security action is request has been completed. It is legal for writes to the PMBus action request register to report STATUS_CML or NACK while the firmware that handles its commands is being updated. In all other cases, it is expected that the “security action status” register will report “In Progress” value from Table 12 while an operation is in progress.

When the operation completes, either successfully or with failure, it will transition again to a new value indicating the success or failure and some details about that failure. Unique return codes are allocated to communicate status that a device user may need to know, whereas errors only likely encountered by the system vendor or manufacturer developing drivers are bundled into simpler error codes.

For successful actions that store resultant data to the buffer regions for readback by the PMBus controller, the action status must transition back to “Idle” per Table 12 once that returned data has been overwritten. If no returned data is overwritten, then the security action code will remain until the next action request is submitted per step A8 of Figure 5.

Table 12. PMBus Security Action Status Error Codes

Result	Meaning
00h	Idle
01h	Succeeded, operation complete.
02h-5Fh	Reserved for future codes
60h-7Fh	Succeeded, Reserved for manufacturer defined codes
80h-9Fh	Failed, manufacturer defined reason
A0h-EFh	Failed, reserved reason
F0h	Failed, cause unspecified
F1h-F5h	Failed, reserved reason.
F6h	Failed, Operation not supported

Result	Meaning
F7h	Failed, Unsupported security action details
F8h	Failed, Unable to attest PMBus host (PRoT). Target generated nonce is invalidated or not present.
F9h	Failed, Unable to attest PMBus host (PRoT): MAC failure - Calculation result mismatch.
FAh	Failed, Update to invalidated version prohibited
FBh	Failed, Not enough free NVM space
FCh	Failed, Watchdog timeout
FDh	Failed, Ordering requirements not met
FEh	Failed, Operation is blocked. Operation can be blocked by many causes, including Authentication Failure, PSK not set, Invalid Location, Operation Locked out, Volatile updates not allowed for selected firmware segment. Errors that fall into this category are those that would be encountered during development of firmware update software, but not likely encountered by the end-user.
FFh	Operation in progress.

8. VR Attestation

The PMBus security application profile requires that the PMBus controller be able to validate that PMBus targets are the exact devices, with the same firmware, configuration, and pre-shared key as expected by the system integrator. The PRoT is the trusted verification agent to drive this attestation request. This attestation is to be executed prior to the VR output being enabled and intermittently during device operation. The operations described in this section are required for all security profile definitions.

The required solution uses only one pair of messages between the PMBus target and PRoT for attestation.

The attestation protocol accomplishes the following goals:

- During the boot phase, the PRoT authenticates the PMBus target firmware and configuration. If the target is pre-loaded, the mechanism validates that the on-board firmware and configuration are a match to the latest updated version available in the PRoT. If it is not pre-loaded, it validates that the PMBus is the right hardware to accept the firmware/configuration from the PRoT.
- Establish a secure channel between the PMBus target and the PRoT with mutual authentication using a pre-shared key (PSK) sized per the attestation algorithm requirements.

The process requires the following hardware capabilities to support attestation:

- The PMBus target is required to support a hash function, such as SHA-384 or SHA3-384 of its firmware and configuration ROM.
- The PRoT is required to support a digital random bit generator (DRBG) and a keyed hash function (HMAC or KMAC).

- A secure memory in both the PMBus target and the PRoT that cannot be accessed from off-chip to store the secret PSK.
- In the event a device supports multiple configurations the manufacturer must include the active configuration file and firmware in the hash calculation. It is optional to include inactive configuration files in the hash calculation. If the inactive configuration files are included as part of the hash calculation, the hash must incorporate data including which file is active, so the attestation can detect if the active configuration file choice has been changed. This attestation shall be computed over the firmware, NVM, and configuration data and need not be dynamically updated per any volatile commands applied by the PRoT after power-on. It must be recomputed after the push of an authenticated firmware or configuration update to either active volatile or non-volatile memory. The attestation shall not be affected by "transient" operations such as PMBus VOUT_COMMAND that do not alter the firmware or default configuration settings.
- PSK that is unique to each motherboard instance and recommended to be unique per each VR.

The process used to perform device attestation is summarized in Figure 6. Any key values used or calculated during the attestation or authentication processes should be retained in memory for the shortest amount of time possible in both the PRoT and the target device.

The keyed hashes discussed in this specification require random number generation. The random number generator used in nonce generation during attestation should be as strongly random as the strength of the hashing algorithm. Guidelines for random number generation and entropy sources are given in [A10], [A11], and [A12]. Validation collateral and tools are given in [R04], [R05], and [R06].

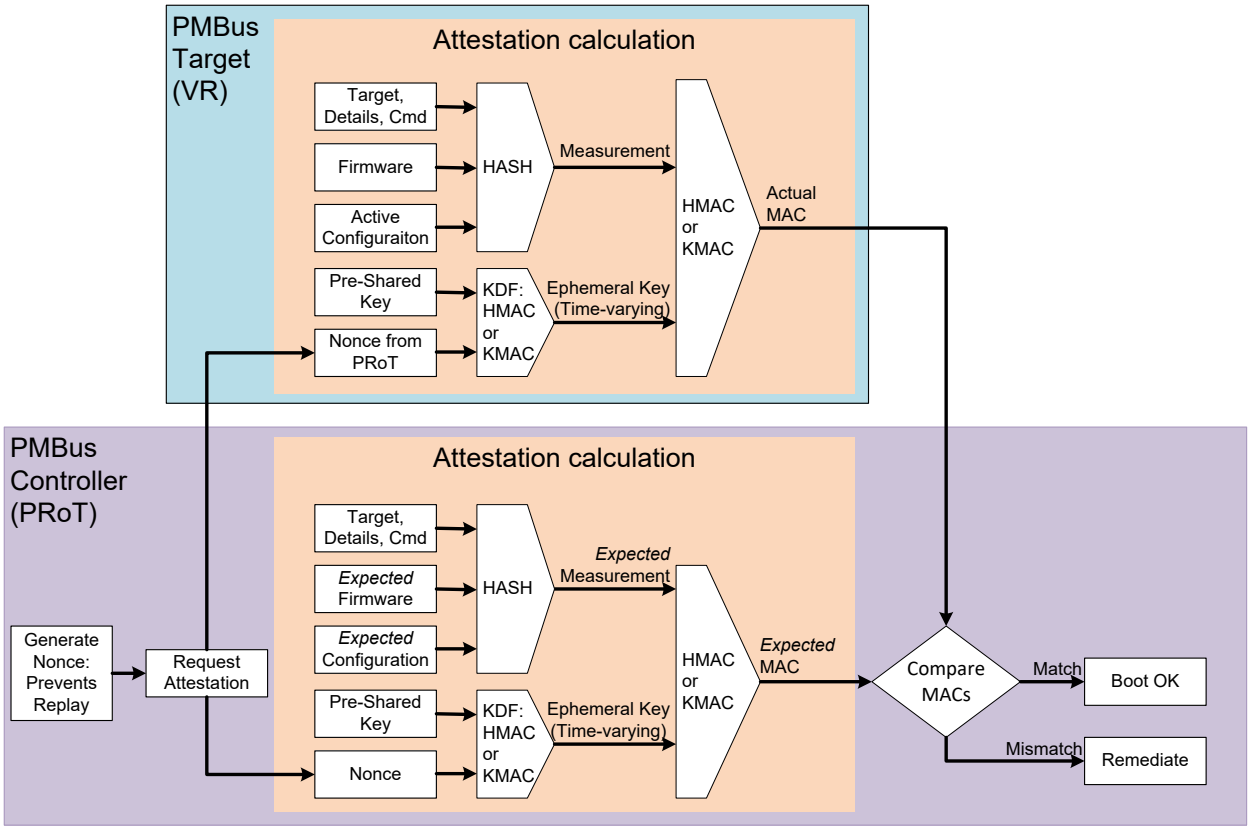


Figure 6. Attestation Flow with APIs from [A03]

8.1 Pre-Shared Key Provisioning

The series of steps in provisioning this PSK is described in the remainder of this section. In this flow, a PSK for every unit is generated using a DRBG. The PMBus target and the PRoT store the PSK in secure non-volatile memory (NVM), such as fuses, respectively. PSK storage requires confidentiality protection, it shall not be directly readable from outside the PRoT or VR. The PSK shall be statistically unique per unit, only the VR and the PRoT on the specific unit know the PSK.

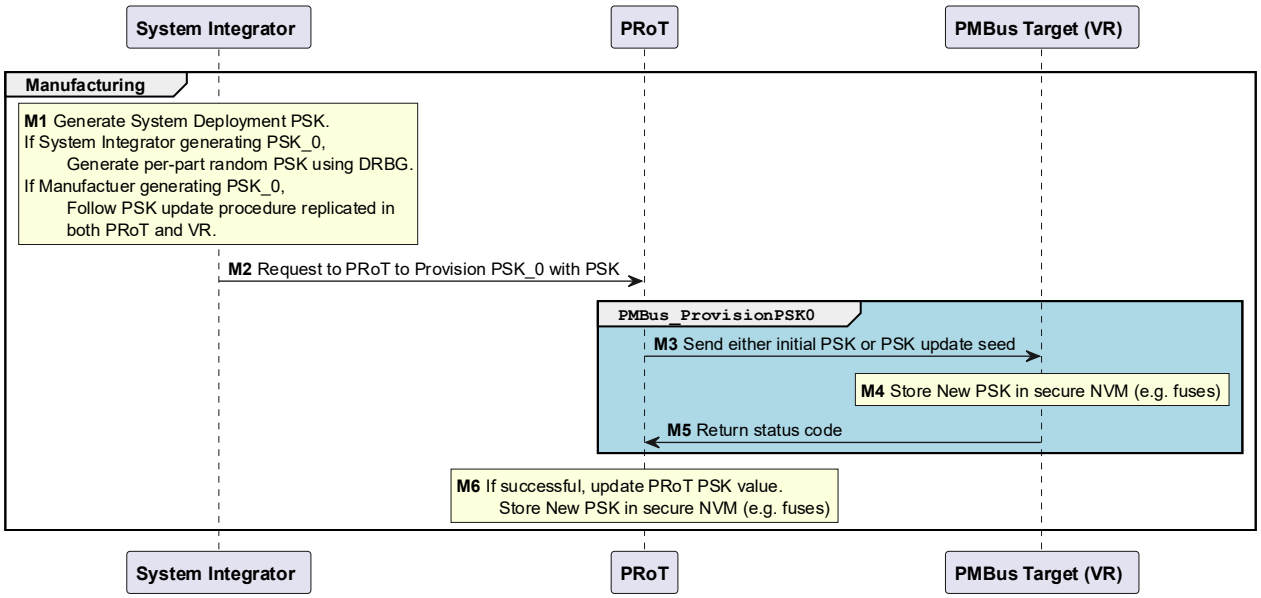


Figure 7. PSK Provisioning Process

8.1.1 Initial PSK Requirements

Refer to the secure device application profile for initial PSK requirements.

8.1.2 Initial PSK Tracking Requirements

If PSK_0 is fused by the manufacturer, it is mandatory to provide the system integrators with a secure method to determine the PSK on each device. This method **must be** irreversibly fused off by the system integrator after assembly to ensure system security. It is recommended that methods be utilized to track the PSK to the individual VR that have zero opportunity for accidental disclosure.

8.1.3 Provisioning Of Initial PSK: Standard Implementation

The command used to program the PSK_0 is “security action” 05h.

The flow of the command used to program the PSK_0 within the secure environment is shown in Table 13.

Table 13. PSK_0 Programming Request Activity

Step from Figure 5	Relevant Bit Fields	Contents
A1	Write Primary Address 19h “security action request”	85h: Start PSK_0 setting command sequence
A2	Write Primary Address 17h “security action details [0]”	Reserved, program to 00h.
A3	Write Primary Address 18h “security action details [1]”	Reserved, program to 00h

Step from Figure 5	Relevant Bit Fields	Contents
A6	Write Buffer region [1]	Program PSK_0 value to use
A8	Write Primary Address 19h “security action request”	05h: PSK_0 setting command.

In response to the “security action request” with value 05h, the PMBus target will check if the PSK_0 has been previously programmed and thus locked out forever. If the PSK_0 has been previously programmed “security action status” will indicate a blocked operation and the PSK_0 is not modified. If the PSK_0 has not been previously programmed, the data in PMBus shared buffer region [1] will be loaded into the PSK_0 storage within secure NVM and “security action status” will indicate success.

Throughout the PSK_0 programming request, the PMBus target updates the “security action status” register of primary memory space per Section 7.6.

8.2 Pre-Shared Key Iteration Process

If the PMBus target can iterate PSKs, then a new PSK is derived from the current PSK and a seed specified by the PRoT. Once the PMBus target calculates a new PSK_N and programs PSK_N into secure NVM, the previous PSK_{N-1} is deactivated. The PRoT must have the following data to iterate the PSK: the prior PSK, the iteration algorithm, and the seed. To provide security for the iteration process, a nonce and expected MAC is required to prove to the target that the requestor knows the existing PSK before iterating the PSK. The nonce can be either generated by the PRoT or requested from the target. This MAC is calculated per the attestation process of 8.3 and 8.4, but in this case the input data message is as shown in Table 14 if generated by the PRoT and as shown in Table 30 if generated by the target.

Table 14. Data Input Message To Be Hashed For PSK Iteration Measurement

Measurement Hash	Input Data Stream
All Hashing Algorithms	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details [1] || security action request || Seed from buffer region [2] } Where || is the concatenation operator and << is the left-shift operator. When left shifting the 7-bit device address bit 0 is filled with a 0 to create an 8-bit value.

Upon successful completion of the PSK iterate command, the PMBus target and the PRoT both will have independently derived the new PSK from the current PSK and the seed using an algorithm from Table 15 based upon the selected via the “security action details” registers. The PMBus host (PRoT) and target will both store the new PSK into their secure NVM, non-outside-readable NVM.

The process for establishing which PSK iteration algorithm is supported by a device is presented in Section 8.2.1. The process for requesting a PSK iteration is described in Section 8.2.2. A process for permanently locking the PSK is described in Section 8.2.3.

Table 15. Permitted PSK Iteration Algorithms

PSK Iteration Algorithm	Resultant PSK_N	KDF Operation
0	PSK_N (256-bit)	**KDF in counter mode per [A09] Section 4.1 with: Keyed-Hashed Message Authentication Code SHA-256 per [A04]** $PSK_N = K_{OUT} = PRF(K_{IN}, X)$ $= HMAC_{SHA\text{-}256}(K = PSK_{N-1}, text = X)$ Where: PRF = HMAC with h = SHA-256(SHA2) from [A04] K_{IN} = PSK_{N-1} $X = 0x0001 \mid\mid Label \mid\mid 0x00 \mid\mid Context \mid\mid [L]_2$ L = length(PSK_N) = 256 bits h = 256 bits (SHA-256) n = $[L/h] = 1$ $Label$ = ASCII of “PSK” $Context$ = Seed from Buffer Region [2] $[L]_2$ = 0x0100
1	PSK_N (128-bit)	**KDF Using KMAC mode per [A09] Section 4.4 with: KMAC128(K, X, L, S) based upon SHA3 from [A08].** $PSK_N = K_{OUT}$ = KMAC128(K, X, L, S) Where: $K = K_{IN}$ = PSK_{N-1}(128 bits) $X = Context$ = Seed from Buffer Region [2] L = length(PSK_N) = 128 bits $S = Label$ = ASCII of “PSK”
2	-	**Reserved**

PSK Iteration Algorithm	Resultant PSK_N	KDF Operation
3	PSK_N (256-bit)	**KDF Using KMAC mode per [A09] Section 4.4 with: KMAC256(K, X, L, S) based upon SHA3 from [A08].** $PSK_N = K_{OUT}$ = KMAC256(K, X, L, S) Where: $K = K_{IN}$ = PSK_{N-1}(256 bits) X = $Context$ = Seed from Buffer Region [2] L = length(PSK_N) = 256 bits S = $Label$ = ASCII of “PSK”

8.2.1 PSK Iteration Algorithm Support

The three bytes of the PMBus secure device primary memory shown in Table 16 are used to indicate which PSK update algorithms are supported to generate the next PSK from the existing PSK and the buffered seed. Each bit within the range of the first three bytes indicates support for the corresponding PSK update algorithm listed in Table 15. An additional byte, described in Section 12.3 is available that indicates the quantity of NVM remaining, with one field specifically indicating the amount of free PSK storage.

Table 16. PSK Iteration Algorithm And Remaining NVM Support Bytes

PMBus security memory map register and address	Bit	Description
Primary Address 07h “PSK Iteration Algorithm Support [0]”	[0] (LSB)	PSK iteration algorithm 0 from Table 15
	[1]	PSK iteration algorithm 1 from Table 15
	[2]	PSK iteration algorithm 2 from Table 15
	[3]	PSK iteration algorithm 3 from Table 15
	[4]	Reserved for future algorithms, set to 0.
	[5]	Reserved for future algorithms, set to 0.
	[6]	Reserved for future algorithms, set to 0.
	[7] (MSB)	Reserved for future algorithms, set to 0.
Primary Address 08h “PSK Iteration Algorithm Support [1]”	[7:0]	Reserved for future algorithms. Set to 00h
Primary Address 09h “PSK Iteration Algorithm Support [2]”	[0 LSB:3]	Reserved, set to 0h
	[4]	Reserved for manufacturer algorithm [0]
	[5]	Reserved for manufacturer algorithm [1]

PMBus security memory map register and address	Bit	Description
	[6]	Reserved for manufacturer algorithm [2]
	[7] (MSB)	Reserved for manufacturer algorithm [3]

8.2.2 Pre-shared key Iteration: Request Process

The process of requesting a PSK iteration is described in Table 17. To initiate the PSK iteration request, three items must be available to the PMBus security shared memory regions. This is done using the “indirect access to large memory buffer” technique outlined in Section 6.3

The three items needed to iterate the PSK include the nonce, the expected MAC resulting from that nonce, and the seed used to iterate the PSK. This command allows either for the nonce to be generated by the PRoT or generated by the target, based upon the security action details.

This request process utilizes attestation to verify that the PRoT is the agent making this iteration request knows the original PSK and that the request data is not modified. This helps prevent a malicious actor from iterating the PMBus target key without knowledge of the PRoT. This act would cause the PRoT to be unable to attest the PMBus target.

If being run with a target generated nonce, then the PRoT will request a nonce for a given attestation algorithm from the target per Section 9. If the PSK is being generated on a nonce, then two criteria must be met: (1) the PSK utilized is different on each unit of a board (PSK are not re-used across all boards), and (2) there are no scenarios where the PSK would fail to update due to any other reason than failed MAC verification, unsupported command, or insufficient space to perform the upload.

Figure 8 and Table 17 present the steps required to update the PSK.

Table 17. Process Requesting PSK Iteration

Step In Figure 8	Action	Description
I1	Select Command Attestation Algorithm	This algorithm validates the device requesting the PSK iteration knows the existing PSK. This process uses the same attestation flow used to validate the firmware and configuration, but the measurement is formed from a hash of the security action command instead. Attestation algorithms chosen from Table 21.

Step In Figure 8	Action	Description
I2	**If using a target generated nonce**, request a nonce using the steps of Section 9. **If using a PRoT generated nonce**, generate the nonce using the PRoT DRBG. **For both cases**, generate the seed for iteration on the PRoT using the DRBG.	The nonce is used to protect the attestation calculations from replay attacks. Because the PSK will change if the command is successfully committed and security action request and target address are parts of the data being hashed, replay attack effects should be minimal. The seed is used as the data to feed calculation of the next PSK.
I3	Determine E[meas]	Determine expected measurement as a hash of the security action command request with as described in Table 14.
I4-I5	Determine the ephemeral key for the given nonce	Per process described in Section 8.4.3
I6	Calculate Expected MAC	This data must equal the result obtained by the target for the same data and PSK for the command to be accepted. Computed per Section 8.4.4.
I7	Write Primary Address 19h “security action request”	Tell the target that PSK iteration sequence is beginning. 84h: Begin “PSK iteration” request sequence
I8	Write Primary Address 17h “security action details [0]”	Write PSK Iteration Algorithm Selection 00h: Use PSK iteration algorithm 0 from Table 15 01h: Use PSK iteration algorithm 1 from Table 15 02h: Use PSK iteration algorithm 2 from Table 15 03h: Use PSK iteration algorithm 3 from Table 15 14h: Use PSK iteration algorithm 20 from Table 15 15h: Use PSK iteration algorithm 21 from Table 15 16h: Use PSK iteration algorithm 22 from Table 15 17h: Use PSK iteration algorithm 23 from Table 15 Other values: reserved for future algorithms Specific implementations may neglect bits that are always 0 for all available implemented algorithms.

Step In Figure 8	Action	Description
I9	**If Target generated nonce:** Set to 00h. **If PRoT generated nonce:** Write. Primary Address 18h "security action details [1]"	Command Attestation Algorithm Selection **If using target generated nonce,** set to 00h. Security action command attestation algorithm was determined during the nonce request process. **If using PRoT generated nonce,** set as described below so the target knows the attestation algorithm used to verify the actual MAC against the expected value. Bits [7:5], set to 100b Bits [4:0], set per attestation algorithm below: • 00h: Use attestation algorithm 0 from Table 21 • 01h: Use attestation algorithm 1 from Table 21 • 02h: Use attestation algorithm 2 from Table 21 • 03h: Use attestation algorithm 3 from Table 21 • 04h: Use attestation algorithm 4 from Table 21 • 05h: Use attestation algorithm 5 from Table 21 • 06h: Use attestation algorithm 6 from Table 21 • 07h: Use attestation algorithm 7 from Table 21 • 06h to 13h: Reserved for future algorithms • 14h: Use attestation algorithm 20 from Table 21 • 15h: Use attestation algorithm 21 from Table 21 • 16h: Use attestation algorithm 22 from Table 21 • 17h: Use attestation algorithm 23 from Table 21 • Others: Reserved.
I10	Write Buffer Region [2]	Write PSK iteration seed generated by PRoT This is the value used to randomize the PSK iteration algorithm using the hashing calculation Byte ordering within the memory region is to be given in the device literature.
I11	**If using target generated nonce,** skip. **If using PRoT generated nonce,** write Buffer Region [1]	Nonce generated by PRoT. **If using target generated nonce,** this value shall not be written as it already holds the target generated nonce. If this area is written, the target generated nonce must be invalidated yielding request failure. **If using PRoT generated nonce,** this value must be programmed with the nonce to be used during the attestation algorithm. Byte ordering within the memory region is to be given in the device literature.

Step In Figure 8	Action	Description
I12	Write Buffer Region [0]	Write Expected MAC used for PRoT attestation Write MAC result that target should match when replicating the calculation using target-determined nonce and original PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.
I14	Write Primary Address 19h “security action request”	Initiate the PSK iteration request Write 04h: PSK iteration request start. Target indicates “In Progress” via “security action status” register.
I15-I23	Target updates PSK	Target attempts to attest PRoT. If it passes attestation, then the target: (1) calculates the new PSK using the selected iteration algorithm and seed, (2) it stores the new PSK in its NVM and deactivates the prior PSK. Target updates “security action status” register based on operation success per Section 7.6. Requested nonce in buffer region [1] is deactivated regardless of operation success as it has now been used.
I24	Read Primary address 15h “Security Action Status”	PRoT determines if the operation was successful.
I25-I26	PRoT updates PSK.	If PSK iteration was successful, PRoT updates its pre-shared key in line with the target’s update. Prior PSK is deactivated.

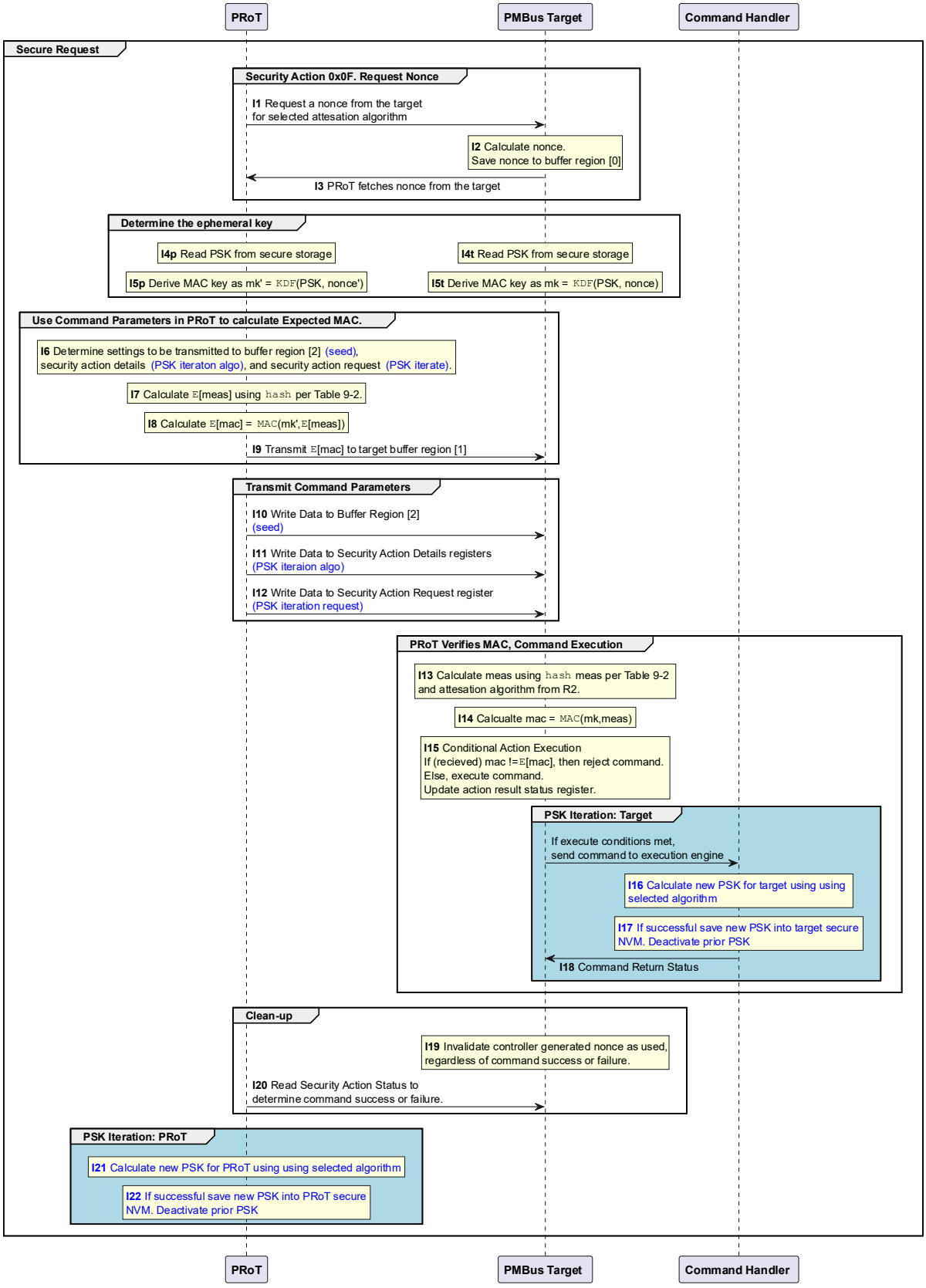


Figure 8. PSK Iteration Flowchart

8.2.3 Pre-shared Key Lock: Forever (Non-volatile) with PRoT generated nonce

A command exists to write-protect the PSK. Once locked permanently, the lock bit can only be released by a security level 2 or security level 3 authenticated update that re-writes the entirety of device NVM. While the lock bit can be transmitted as part of a firmware update, the PSK itself shall not be directly transmitted, including during any authenticated firmware update. Transmitting the PSK directly during an authenticated update could leave the PSK free to be sniffed by an aggressor.

If a device has the capability to handle PSK lock via a security level 2 authenticated update, this method is an acceptable alternative to this command provided that the alternate method is documented in the product literature.

To ensure that the person requesting the PSK lock is authorized, the command follows the attestation process of 8.3 and 8.4. The data to be measured for the attestation is not the firmware and configuration, but instead the command parameters shown in Table 18. Because the lock bit is permanent, and the command data hashed includes the requested security action, then retry attacks are not meaningful as the lock will already be set.

The sequence of commands required to perform the PSK lock operation is shown in Table 19.

Table 18. Data Input Message To Be Hashed For PSK Lock Forever With Attestation Of Host

Measurement Hash	Input Data Stream
All Hashing Algorithms	{(PMBus target 7-bit address << 1) || security action details[0] || security action details [1] || security action request } Where || is the concatenation operator and << is the left-shift operator. When left shifting the 7-bit device address bit 0 is filled with a 0 to create an 8-bit value

Table 19. PSK Iterate Lock Forever Command Byte Write Sequence

Step in Figure 9	Action	Description
L1	Select attestation algorithm. Generate a nonce for that attestation algorithm.	Select an algorithm from Table 21 to attest the command issuer knows PSK. Generate a nonce using DRBG that when combined with the PSK will form an ephemeral key.
L2	Determine E[meas]	Determine the expected measurement as a hash of the security action command request with as described in Table 18.
L3-L4	Determine the ephemeral key for the given nonce	Per process described in Section 8.4.3

Step in Figure 9	Action	Description
L5	Compute the E[mac]	Compute the expected MAC using the hashing algorithms of Section 8.4.4.
L6	Write Primary Address 19h “security action request”	Program to 90h: Start PSK lock forever command sequence
L7	Write Primary Address 17h “security action details [0]”	Reserved set to 00h.
L8	Write. Primary Address 18h “security action details [1]”	Command Attestation Algorithm Selection Set as described below so the target knows the attestation algorithm used to verify the actual MAC against the expected value. Bits [7:5], set to 100 binary. Bits [4:0], set per attestation algorithm below • 00h: Use attestation algorithm 0 from Table 21 • 01h: Use attestation algorithm 1 from Table 21 • 02h: Use attestation algorithm 2 from Table 21 • 03h: Use attestation algorithm 3 from Table 21 • 04h: Use attestation algorithm 4 from Table 21 • 05h: Use attestation algorithm 5 from Table 21 • 06h: Use attestation algorithm 6 from Table 21 • 07h: Use attestation algorithm 7 from Table 21 • 08h: Use attestation algorithm 8 from Table 21 • 09h: Use attestation algorithm 9 from Table 21 • 0Ah: Use attestation algorithm 10 from Table 21 • 0Bh: Use attestation algorithm 11 from Table 21 • 0Ch to 13h: Reserved for future algorithms • 14h: Use attestation algorithm 20 from Table 21 • 15h: Use attestation algorithm 21 from Table 21 • 16h: Use attestation algorithm 22 from Table 21 • 17h: Use attestation algorithm 23 from Table 21 • Others: Reserved.
-	Skip Buffer Region [2]	No additional parameter data to write for this command
L9	Write Buffer Region [1]	Nonce generated from target for attesting seed and command, establishing shared PSK. Byte ordering within the memory region is to be given in the device literature.

Step in Figure 9	Action	Description
L10	Write Buffer Region [0]	Write Expected MAC value used to attest command data of Table 18. Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.
L12	Write Primary Address 19h “security action request”	Execute the PSK lock forever command action. Program to 10h: PSK lock forever command go Target will place lock as requested if PRoT attestation and ordering requirements are met and a valid lock type is presented in security action details [0].
L20	Primary Address 15h “security action status”	Read for PSK lock request operation status. Values are set per Section 7.6. If PSK lock in NVM is not supported, an “Operation Blocked” status will be returned. If PSK lock is not supported via security action request 06h or 10h and iterations are supported, then PSK shall be lockable via an authenticated firmware update at security level 2 or 3.

8.2.4 Pre-Shared Key Lock With Target Generated Nonce (Volatile Or Non-Volatile)

A command exists to write-protect the PSK for either the remainder of the power-on-reset cycle or forever. If locked for a power-on-reset cycle, it cannot be unlocked for the remainder of that power cycle. If locked in non-volatile memory, then the PSK remain unchangeable unless a security level 2 or security level 3 authenticated firmware update reprograms the lock bit NVM state.

To ensure that the person requesting the PSK lock is authorized, the command follows the attestation of controller process presented in Section 9. This process requires a target generated nonce. The sequence of commands used to perform this remainder of power-sequence lock is described in Table 20.

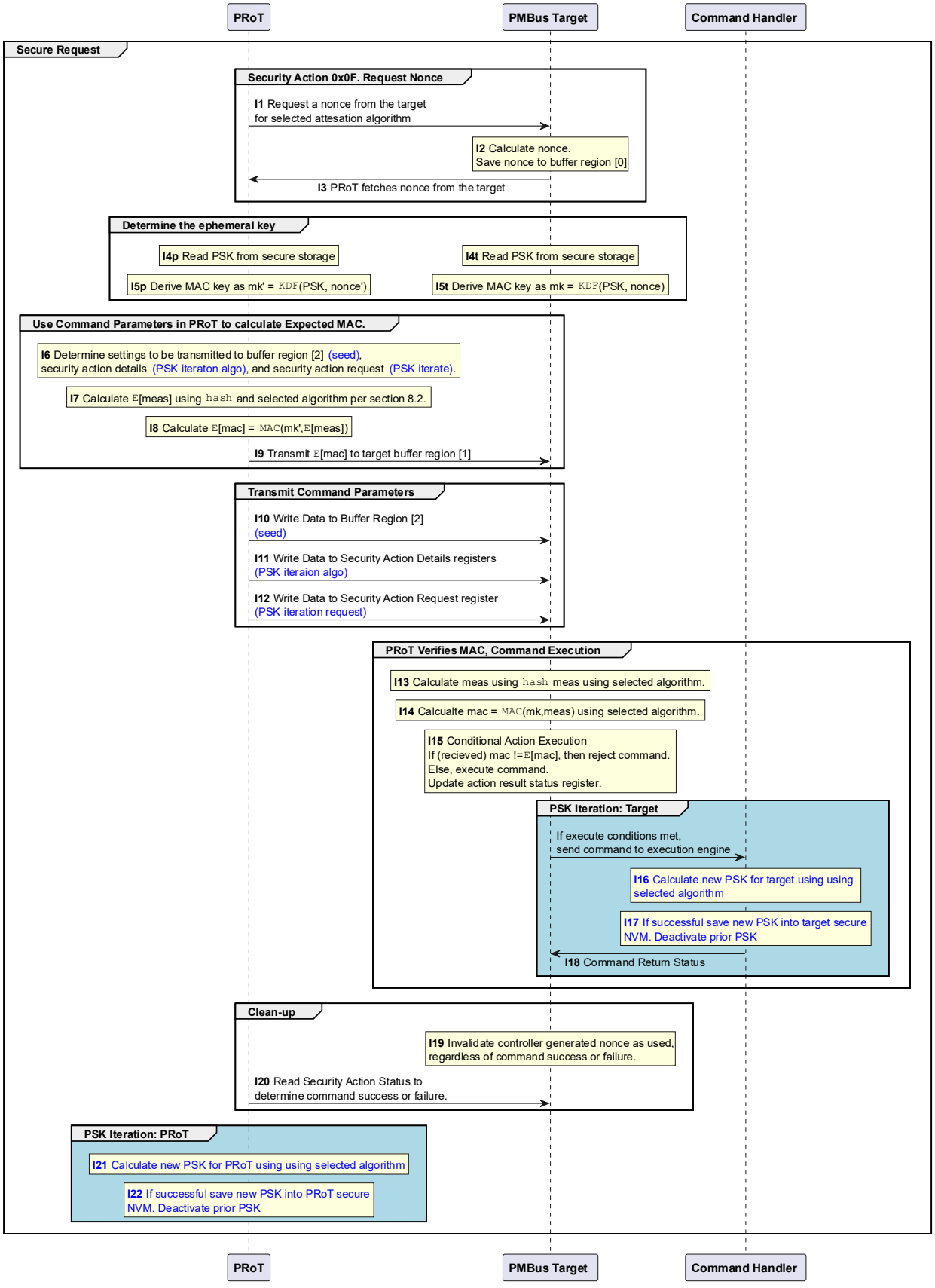


Figure 9. Pre-shared Key Lock Forever Flow Diagram

Table 20. PSK Iterate Lockout For Power-Cycle Command Byte Write Sequence

Step In Figure 11	Action	Description
R1-R8	Perform PRoT attestation calculations as required per Section 9.	Request a random nonce and a PRoT attestation algorithm from the target using the process described in Section 9.2. Read the nonce from buffer region [1]. Use the PSK lock type to calculate the expected MAC per Table 30. This expected MAC will be pushed to the target later in this process to help attest that the PRoT is the issuer of the PSK lock request.
R9	Write Primary Address 19h “security action request”	Program to 86h: Start PSK lock command sequence
R10	Write Primary Address 17h “security action details [0]”	Program the lock duration. 00h: Set this value for PSK lock in NVM, with the locking applied to the remainder of this controller power-on-reset cycle and all subsequent power-on reset cycles. 01h: Set this value for PSK locking in volatile memory, with the locking applied to the remainder of this controller power-on-reset cycle. 02h to FFh: Reserved for future applications.
R11	Write Primary Address 18h “security action details [1]”	Reserved set to 00h.
-	Skip Buffer Region [2]	No additional parameter data to write for this command
-	Skip Buffer Region [1]	Nonce generated from Target Nonce Request. Do not write this location or it will be invalidated.
R13	Write Buffer Region [0]	Write Expected MAC value used to attest PRoT Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. MAC is calculated per Table 30. Byte ordering within the memory region is to be given in the device literature.

Step In Figure 11	Action	Description
R15	Write Primary Address 19h “security action request”	Execute the PSK lock command action. Program to 06h: PSK lock command go Target will place lock as requested if PRoT attestation and ordering requirements are met and a valid lock type is presented in security action details [0].
R19	Primary Address 15h “security action status”	Read for PSK lock request operation status. Values are set per Section 7.6. If either the command is not supported or the specific lock type being requested is not supported, then an “Operation Blocked” status will be returned. If PSK lock is not supported via security action request 06h or 10h and iterations are supported, then PSK shall be lockable via an authenticated firmware update at security level 2 or 3.

8.3 Attestation Protocol Summary

During the boot, and at any point later in time, the PRoT can submit a request for attestation to the PMBus secure device target. This mechanism is used to validate the PMBus target, its firmware, its configuration, and its PSK are a match to the PRoT’s anticipated values. The protocol described below is a means to accomplish this attestation while providing protection against attacks as described in [A03] “Threat Matrix” section. The protocol is illustrated in Figure 10 and described in the subsequent text.

To trigger the attestation flow, the PRoT generates a random nonce (step A1 of Figure 10). It sends the target a security action request informing the target that it is beginning an attestation request (step A2). It communicates the attestation algorithm (steps A3, A4) and sends the nonce to the target’s buffer region [1] (step A5). It checks that everything was transmitted without issue (step A6) and then informs the target that all data is in place and to start the attestation calculation (step A7).

In response to the attestation request, the PMBus target calculates hash of the current FW and configuration (step A8v). The PRoT is preloaded with a value corresponding to the hash that would result from the desired target FW and configuration. The expected measurement value in the PRoT and the PMBus target should match. The algorithm used to calculate this hash is selected from Table 21.

Using their pre-shared key entry and the shared nonce, the PRoT and the PMBus target each calculate an ephemeral (time-varying) key. To calculate this, the PRoT (step A8p) and the PMBus target (step A8v) read the active PSK entry from their own internal secure storages. The PRoT (step A9p) and the PMBus target (step A9v) each derive an ephemeral MAC key “mk” using the key derivation function (KDF) from the PSK and the nonce. The KDF process is described in 8.4.3 and uses an algorithm from Table 21. If the two PSK match, these two ephemeral keys should also be equal.

To complete the proof to the PRoT, the VR calculates a “mac” using the measurement (step A8v) as data and “mk” of step A10v as the ephemeral key in step A11v using an

algorithm selected from Table 21. While the PMBus target is doing its calculations, the PRoT also calculates an expected MAC value E[mac] (step A11p) from the measurement of step A8p and “mk’” of step A10p. When both calculations are complete, the PRoT will retrieve it from the PMBus target in the clear via the PMBus (step A16).

Finally, the PRoT compares received “mac” with locally calculated expected “mac” E[mac] (step A18). If they are not equal, then either the PMBus target is not running the PRoT-authenticated FW and/or configuration, or there was alternation on PSK and/or messages exchanged between the PRoT and the target. In these scenarios, the PRoT should assume that an attack had happened and perform remedial actions such as reprogramming the VR and/or reporting an error.

Algorithm pair sets available for the secure hash function and for the keyed-hash message authentication are chosen from the list of algorithms in from Table 21.

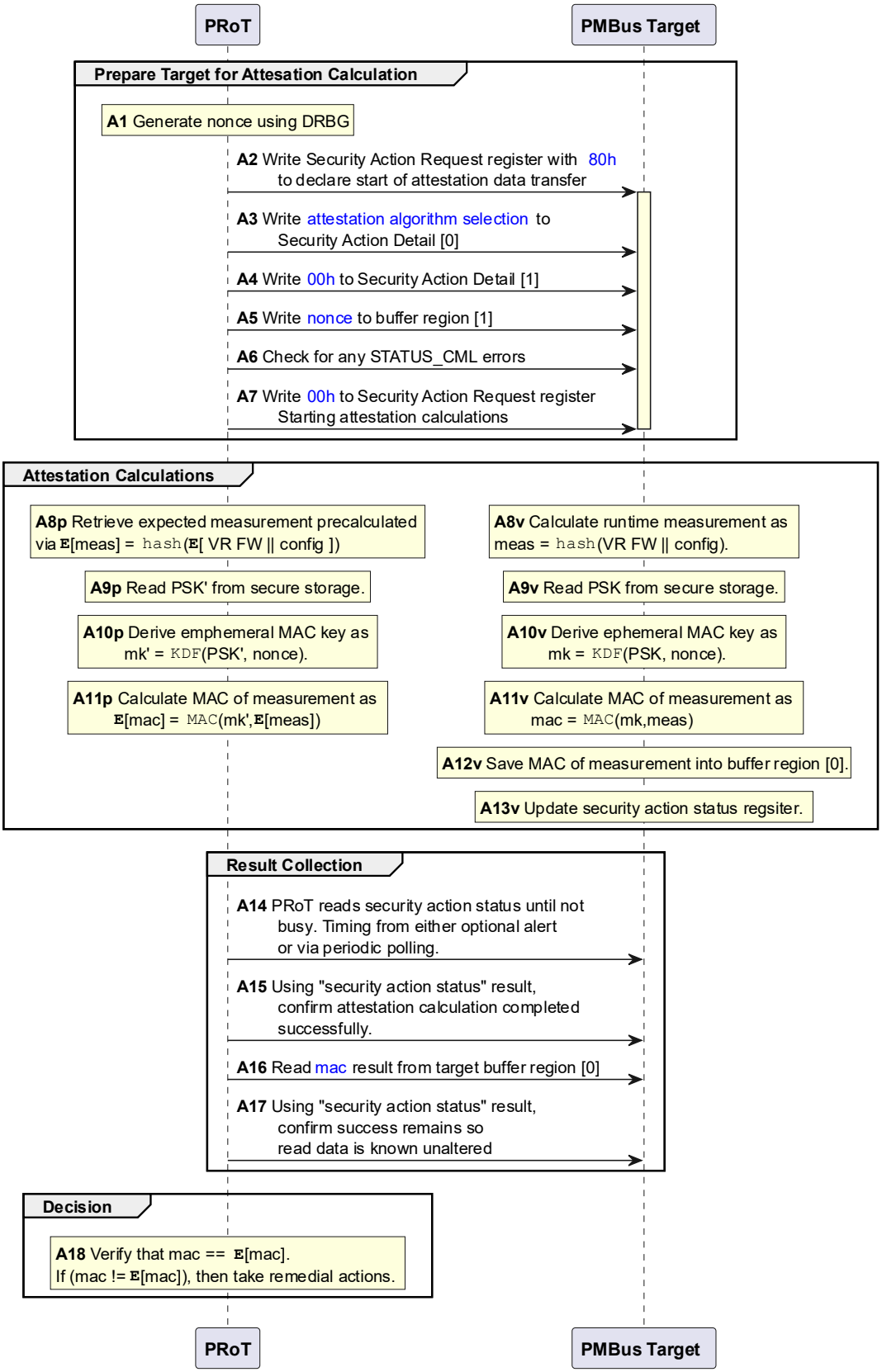


Figure 10. Attestation Process Flow

Table 21. Attestation Algorithm Options

Attestation Algorithm Set	Hashing Algorithm Used To Develop Measurement Of Firmware And Configuration	Keyed Hash Function Used To Determine Ephemeral Key And Resulting Hash Message Authentication Code
0	Hash A from Table 22	Keyed Hash A from Table 23 and Table 24
1	Hash A from Table 22	Keyed Hash B from Table 23 and Table 24
2	Hash A from Table 22	Keyed Hash C from Table 23 and Table 24
3	Hash A from Table 22	Keyed Hash D from Table 23 and Table 24
4	Hash B from Table 22	Keyed Hash A from Table 23 and Table 24
5	Hash B from Table 22	Keyed Hash B from Table 23 and Table 24
6	Hash B from Table 22	Keyed Hash C from Table 23 and Table 24
7	Hash B from Table 22	Keyed Hash D from Table 23 and Table 24
8	Hash C from Table 8-10	Keyed Hash A from Table 23 and Table 24
9	Hash C from Table 8-10	Keyed Hash B from Table 23 and Table 24
10	Hash C from Table 8-10	Keyed Hash C from Table 23 and Table 24
11	Hash C from Table 8-10	Keyed Hash D from Table 23 and Table 24
12-19	Reserved	Reserved
20	Manufacturer Defined Algo 0	Manufacturer Defined Algo 0
21	Manufacturer Defined Algo 1	Manufacturer Defined Algo 1
22	Manufacturer Defined Algo 2	Manufacturer Defined Algo 2
23	Manufacturer Defined Algo 3	Manufacturer Defined Algo 3

Table 22. Attestation Measurement Hashing Algorithm

Hashing Algorithm	Message Digest	Resultant Measurement Length	Operation	Data Input Message (M)
Hash A	meas	384-bits	**SHA-384** from SHA2 [A04], Block Size = 1024	*For device attestation command:* see Table 27. *For PSK iteration command*: see Table 14. *For PSK lock forever command:* see Table 18
Hash B not recommended beyond 2030	meas	256-bits	**SHA3-256** from [A06]	
Hash C	meas	384-bits	**SHA3-384** from	

Table 23. Keyed Hash Options for Ephemeral Key Determination (KDF)

Keyed Hash Algorithm Selection	Ephemeral Key (K_{OUT})	KDF Operation
Keyed Hash A	mk (256-bit)	**KDF in counter mode per [A09] Section 4.1 with: Keyed-Hashed Message Authentication Code SHA-256 per [A04]** $mk = K_{OUT} = PRF(K_{IN}, X)$ $= HMAC_{\text{SHA-256}}(K = PSK_N, text = X)$ Where: PRF = HMAC with h = SHA-256 (SHA2) from [A04] $K_{IN} = \text{PSK}_N$ $X = 0x\ 0001 \parallel Label \parallel 0x00 \parallel Context \parallel [L]_2$ $Label$ = ASCII of “VR security protocol” $Context$ = Nonce (256 bits) $[L]_2$ = 0x0100 L = length(PSK_N) = 256 bits h = 256 bits (SHA-256) $n = [L/h\ ]$ Byte ordering within multibyte variables, such as *Label, Context,* and *PSK* shall be given in the device literature.
Keyed Hash B Not recommended beyond 2030	mk (128-bit)	**KDF Using KMAC mode per [A09] Section 4.4 with: *KMAC128(K, X, L, S)* based upon SHA3 from [A08].** $mk = K_{OUT} = KMAC128(K, X, L, S)$ Where: $K = K_{IN} = \text{PSK}_N$(128 bits) $X = Context$ = Nonce(256 bits) L = length(PSK_N) = 128 bits $S = Label$ = ASCII of “VR security protocol” Byte ordering within multibyte variables, such as Label, Context, and PSK shall be given in the device literature.

Keyed Hash Algorithm Selection	**Ephemeral Key (K_{OUT})**	**KDF Operation**
Keyed Hash C	mk (256-bit)	**KDF Using KMAC mode per [A09] Section 4.4 with: *KMAC128(K, X, L,S)* based upon SHA3 from [A08].** $mk = K_{OUT} = KMAC128(K, X, L, S)$ Where: $K = K_{IN}$ = PSK_N(256 bits) $X = Context$ = Nonce(256 bits) L = length(PSK_N) = 256 bits $S = Label$ = ASCII of “VR security protocol” Byte ordering within multibyte variables, such as *Label, Context, and PSK* shall be given in the device literature.
Keyed Hash D	mk (256-bit)	**KDF Using KMAC mode per [A09] Section 4.4 with: KMAC256(K, X,L, S) based upon SHA3 from [A08].** $mk = K_{OUT}$ = KMAC256(K, X, L, S) Where: $K = K_{IN}$ = PSK_N(256 bits) $X = Context$ = Nonce(256 bits) L = length(PSK_N) = 256 bits $S = Label$ = ASCII of “VR security protocol” Byte ordering within multibyte variables, such as *Label, Context, and PSK* shall be given in the device literature.

Table 24. Hashed Message Authentication Code Selection (MAC)

Keyed Hash Algorithm Selection	**Resultant HMAC (K_{OUT})**	**Keyed Hash Message Authentication Code**
Keyed Hash A	mac (256-bit)	**mac = $HMAC_{SHA\text{-}256}(H, K, X)$ from [A05] using SHA2 [A04]** Where: H = SHA-256(SHA2) [A04] K = mk(256 bit) of Table 23 X= Measurement of Table 22 with $length \geq length(mk)$ Byte ordering within the multibyte variables H, K, X shall be given in the product literature.
Keyed Hash B Not recommended beyond 2030	mac (128-bit)	**mac = KMAC128(K, X, L, S) from [A09] using SHA3 [A08]** Where: K = mk (≥128-bit) of Table 23 X = Measurement of Table 22 with l$ength \geq length(mk)$ L = 128-bits, S = “” Byte ordering within the multibyte variables $K, X,$ *and* L shall be given in the product literature.
Keyed Hash C	mac (256-bit)	**mac = KMAC128(K, X, L, S) from [A09] using SHA3 [A08]** Where: K = mk(≥256-bit) of Table 23 X = Measurement of Table 22 with length ≥ length(mk) L = 256-bits S = “” Byte ordering within the multibyte variables $K, X,$ *and* L shall be given in the product literature.

Keyed Hash Algorithm Selection	Resultant HMAC (K_{OUT})	Keyed Hash Message Authentication Code
Keyed Hash D	mac (256-bit)	**mac = KMAC256(K, X, L, S) from [A09] using SHA3 [A08]** Where: K = mk(≥256-bit) of Table 23 X = Measurement of Table 22 with length ≥ length(mk) L = 256-bits S = “” Byte ordering within the multibyte variables K, X, *and* L shall be given in the product literature.

8.4 Attestation Protocol Details And Step-By-Step Command Definition

There are four steps of interaction between the target and host for attestation. The interactions are summarized in the following list, each corresponding to a subsection within section 8.4.

- A method to determine the attestation algorithm(s) supported. Note this may be bypassed when the driver already knows the supported algorithm.
- A method to transfer a segment of nonce bits and to request the target to calculate the attestation MAC.
- A method to validate that attestation calculation was successfully completed.
- A method to retrieve the MAC result from the PMBus target back to the ProT.

8.4.1 Attestation Request Process

8.4.1.1 Attestation Algorithm Support

The following three bytes of the PMBus security memory map is used to indicate which attestation update algorithms are supported by that PMBus target to calculate a MAC from the PSK, the randomized nonce, and the firmware/configuration image. Each bit within the range of four bytes indicates support for the corresponding attestation algorithm listed in Table 21.

Table 25. Host Attestation Algorithm Support Bytes

Relevant Bit Fields	Bit	Description
Primary Address 0Ah Attestation Algorithm Support [0]”	[0] (LSB)	Attestation algorithm set 0 from Table 21.
	[1]	Attestation algorithm set 1 from Table 21.
	[2]	Attestation algorithm set 2 from Table 21.
	[3]	Attestation algorithm set 3 from Table 21.
	[4]	Attestation algorithm set 4 from Table 21.
	[5]	Attestation algorithm set 5 from Table 21.
	[6]	Attestation algorithm set 6 from Table 21.

Relevant Bit Fields	Bit	Description
	[7]	Attestation algorithm set 7 from Table 21.
Primary Address 0Bh “Attestation Algorithm Support [1]”	[7:0]	Reserved for future attestation algorithms, set to 00h
Primary Address 0Ch “Attestation Algorithm Support [2]”	[0 (LSB) :3]	Reserved for future algorithms, set to 0h
	[4]	Attestation algorithm 20 from Table 21.
	[5]	Attestation algorithm 21 from Table 21.
	[6]	Attestation algorithm 22 from Table 21.
	[7] (MSB)	Attestation algorithm 23 from Table 21.

8.4.1.2 Attestation: Nonce Transfer and Initialize Calculation.

A prerequisite to having the PMBus target complete the attestation request, the seed must first be uploaded to shared memory region 0. This is done using the “indirect access to large memory buffer” technique outlined in Section 6.3. Once the nonce is fully loaded into shared memory region 1, then the PRoT will program the “security action details” byte within the PMBus security register map to select the algorithm to be used for the calculation. After selecting the attestation algorithm, the PRoT will write the “security action request” register to initiate the attestation calculation to begin. The steps are summarized in Section 8.3.

Table 26. Data Items To Request Attestation

Step from Figure 10	Data Fields	Contents to Write
A2	Write Primary Address 19h “security action request”	80h: Attestation Request, start data transfer
A3	Write Primary Address 17h “security action details [0]”	Attestation Algorithm Selection 00h: Use attestation algorithm 0 from Table 21 01h: Use attestation algorithm 1 from Table 21 02h: Use attestation algorithm 2 from Table 21 03h: Use attestation algorithm 3 from Table 21 04h: Use attestation algorithm 4 from Table 21 05h: Use attestation algorithm 5 from Table 21 06h: Use attestation algorithm 6 from Table 21 07h: Use attestation algorithm 7 from Table 21 06h to 13h: Reserved for future algorithms 14h: Use attestation algorithm 20 from Table 21 15h: Use attestation algorithm 21 from Table 21 16h: Use attestation algorithm 22 from Table 21 17h: Use attestation algorithm 23 from Table 21 Others: Reserved.
A4	Write Primary Address 18h “security action details [1]”	Reserved. Set to 00h.

Step from Figure 10	Data Fields	Contents to Write
A5	Write Buffer Region [1]	Attestation Nonce Byte ordering within the memory region is to be given in the device literature.
A7	Write Primary Address 19h “security action request”	00h: Attestation Request, start computation

8.4.1.3 Attestation Calculation: Completion Status

While the PMBus is calculating an attestation MAC result in response to the attestation request, the PMBus “security action status” will read “In Progress” per Section 7.6. When the attestation calculation is done, “security action status” will update to a failure or success code per Section 7.6.

8.4.1.4 Attestation: Read MAC Result

The PMBus host (PRoT) will read the MAC result from buffer region [0]. If the target device contains the expected firmware, configuration, and PSK as is expected by the PRoT, then the MAC result from the target will match the expected value and the PRoT may authorize the system to enable the target’s output, presumably and output voltage source. If the target MAC does not match expected value, the target can take remedial action such as trying to apply a firmware update to make the target match the expected configuration.

8.4.2 Calculate Hash Measurement Of VR Firmware And Configuration

The following sequence is used to calculate a measurement by hashing the PMBus target firmware and configuration file. Expected hash function options may include SHA2-384, as described in NIST FIPS 180-4 [A04], or SHA3-256 or SHA3-384 as described in [A06], and other future algorithms. This is step A8v of the verification sequence.

A corresponding expected value is stored within the PRoT as part of its software image. This expected value is formed by calculating the same hashing function across the values expected to be stored in its memory. The result of the hash calculation is not transmitted by the PMBus target, instead it feeds an HMAC function. The HMAC value is transmitted.

Table 27. Hash Measurement Of VR Firmware And Configuration

Result	Operation	SHA data input	Computed By
meas Measurement	Per selection from Table 21 and Table 22	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request || VR_FW || active configuration }	PMBus Target

		Where || is the concatenation operator and << is the left-shift operator. When left shifting the 7-bit device address bit 0 is filled with a 0 to create an 8-bit value	
E[meas] Expected Measurement	Per selection from Table 21 and Table 22	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request || expected VR_FW || expected active configuration } Where || is the concatenation operator and << is the left-shift operator. When left shifting the 7-bit device address bit 0 is filled with a 0 to create an 8-bit value	Pre-computed, stored in PRoT

8.4.3 Calculation Of MAC Key "mk"

The key derivation function (KDF) utilized can include either HMAC-SHA256 (SHA2), as described in [A05], the KMAC as described in [A08], or other future MAC algorithms. The following operation is performed internally within both the PMBus target and the PRoT to convert the nonce and PSK into an ephemeral key for applying HMAC to the target FW/Config measurement hash. This corresponds to steps A10v and A10p of Figure 10.

Table 28. Ephemeral Key Calculation

Result	Operation	HMAC Key	HMAC data input	Computed
mk (32-byte)	Per selection from Table 21 and Table 23	PSK (from step A9v)	Per Table 23	By PMBus target
mk' (32-byte)	Per selection from Table 21 and Table 23	PSK (from step A9p)	Per Table 23	By PRoT

8.4.4 Calculation Of MAC

The MAC key calculated in Section 8.4.3 is used to scramble the hash representing VR firmware and configuration. It is computed by SHA-256 (SHA2) [A05] or KMAC [A08], or a future MAC algorithm. The calculation uses the ephemeral key calculated from Section 8.4.3 and the MAC data input "meas" from Section 8.4.2.

Table 29. MAC Result Calculation

Result	Operation	HMAC Key	HMAC data input	Computed
mac	Per selection from Table 21 and Table 24	mk (from step A10v)	meas (from step A8v)	By PMBus target

Result	Operation	HMAC Key	HMAC data input	Computed
E[mac]	Per selection from Table 21 and Table 24	mk' (from step A10p)	E[meas] (from step A8p)	By PRoT

8.4.5 Retrieval Of MAC Result From PMBus Target

Once the MAC has been calculated, the PRoT must gather the result from the PMBus target. Since alert/interrupt upon security action completion is optional, so the PRoT may need to rely on polling the "security action status" byte. Alert upon completion is described in Section 12.2. As indicated in Table 12, a status value of FFh indicates that the calculation is still in progress. A status of 01h indicates the MAC result was successfully computed and is ready for retrieval by the PRoT.

If the MAC value was successfully calculated, it is stored in shared memory region 0. The contents of the shared memory region 0 are read back following the indirect large memory region access described in Section 6.3.

8.4.6 Comparison And Resulting Action

If the PMBus target returned MAC matches that calculated by the SOC, then the PRoT can consider the PMBus target to have successfully attested to its model, configuration, firmware, and pre-shared key. The boot process can continue. If unable to attest successfully, the system can take remedial action including either attempting to reprogram the VRs or halting the boot process.

9. PMBus Host Attestation

9.1 Process Overview

The process described in this section attests that the authorized host (PRoT) is sending PMBus commands to the target device. This procedure uses the attestation process described in Section 8 used to attest the PMBus target with some modification to instead attest the PRoT. The sequence is illustrated in Figure 11.

In this approach, a DRBG in the PMBus target generates a nonce upon receiving a request from the PRoT per Section 9.2. The process of requesting a nonce also communicates the attestation algorithm that will be utilized and thus the nonce requirements for that algorithm. When the request nonce security action request completes successfully, the PRoT then reads this nonce from the target. Both target and PRoT will use this shared nonce and their PSK to generate their ephemeral key. Both target and PRoT will compute a measurement of the data to be validated via attestation. They both use their ephemeral key and measurement to compute a MAC. If these MAC do not match, then either the secret PSK differs, or the data being attested has been altered. Both differences would be recognized as failures and command requiring PRoT attestation will not be executed.

The measurement is based upon a hash of all data transmitted as part of the request as presented in Table 30. For the PRoT attested command, the security action details [0] and [1] registers are written, followed by any required parameters sent to buffer region [2]. Buffer region [1] is not altered (hold state) as it contains the previously requested target-generated nonce. The expected MAC is then sent out to the target via buffer region [0], followed by the final security action request.

The VR target will similarly calculate a MAC from this data using its ephemeral key and the attestation algorithm for which the nonce was requested. If the calculated MAC matches the value transmitted for expected MAC by the PRoT, then the target will execute the command. If not, an error will be reported to the “security action status” register.

Table 30. Data Considered For PRoT Attestation Measurment Hash By Action

Opcode	Function	Section	Message Data used for Hash HMAC calculation
04h	PSK Iteration	8.2.2	{ (PMBus target 7-bit address << 1) || security action details[0] || security action details [1] || security action request || Seed from buffer region [2] }
06h	PSK Lock with Target Generated Nonce	8.2.4	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request }
09h	Firmware Fetch Lockout	10.5.2	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request }
0Ah	Power-On Reset Lockout	10.4.1	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request }
0Bh	PMBus Status alert triggering	12.2	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request }

Opcode	Function	Section	Message Data used for Hash HMAC calculation
0Ch	Anytime Power-On Reset	10.4.3	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request }
0Dh	PMBus PAGE_PLUS_WRITE and PAGE_PLUS_READ with Host Attestation	12.4	For the write portion: { (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request } Note: as described in Section 12.4.2, for read portion protecting returned data: { security action details [0] || security action details[1] || security action request || buffer region [2] }
0Eh	Secured Access Control Action	11.1	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request || buffer region [2] contents used to transmit command and access pairs }
10h	PSK lock forever	8.2.3	{ (PMBus target 7-bit address << 1) || security action details [0] || security action details[1] || security action request }

Notes:

- Byte ordering within concatenated fields is to be given in the device literature
- || is the concatenation operator
- << is the left shift operator

- When left shifting the 7-bit device address bit 0 is filled with a 0 to create an 8-bit value.

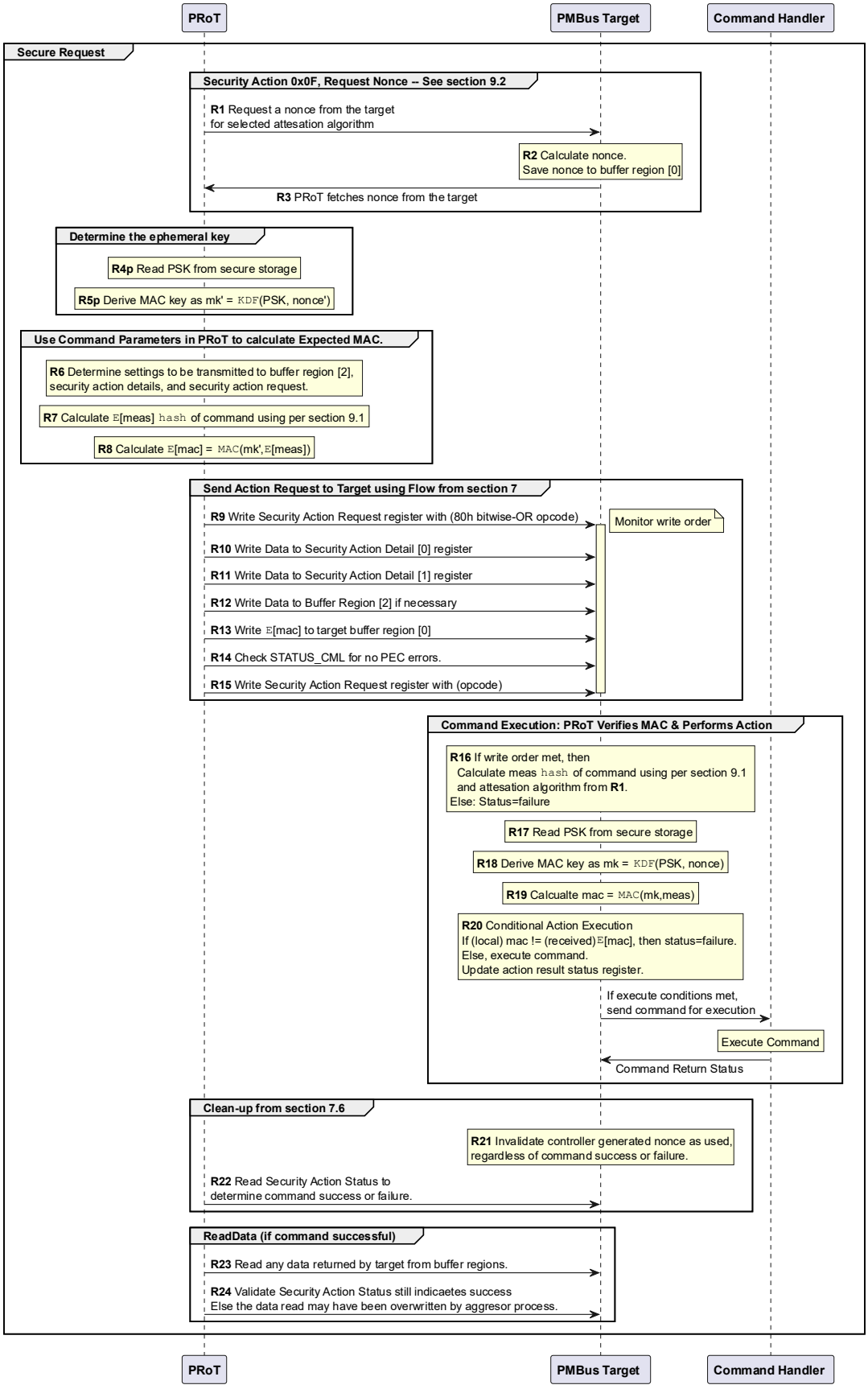


Figure 11. PMBus Host (PRoT) Attested Command to Target

9.2 PMBus Target-Generated Nonce Request

The command sequence to request the target to generate a nonce is shown in Figure 12. This command stores the nonce in buffer region 1. This nonce will be utilized only for the next submitted security action command regardless of if that security action request passes or fails. If any writes are done to security region 1 while it contains the nonce, then the nonce must be invalidated and re-generated before issuing the next attested response. It is recommended that the PRoT read the “security action status” register upon completing the nonce read to ensure the nonce was not altered by another process. If it has not been altered, it will remain at a “Success” value in Section 7.6.

The target must be capable of generating a random number with sufficient entropy for the nonce. The entropy requirement is set per the of the chosen attestation standard to which the nonce will be applied. The nonce-request command must return failure status until sufficient entropy has been developed in the pseudorandom number generator. The random number generator is seeded with a number calculated from the nonce presented during initial device attestation per “security action command” 0 from Table 10. Any attempt by the controller to request a nonce prior to this initial device attestation will be replied to by the target as failed due to insufficient entropy.

Table 31. Process, Registers, And Commands For Attesting Controller Itdentity

Step of Figure 12	Action	Description
N1	Write Primary Address 19h “security action request”	Program to 8Fh: Initiate Fetch Nonce sequence

Step of Figure 12	Action	Description
N2	Write Primary Address 17h “security action details [0]”	This field defines the attestation algorithm to which the nonce will be applied when applying the desired command. This will determine the appropriate nonce and entropy requirements. *Attestation algorithm for which nonce is to be constructed* 00h: Nonce for attestation algorithm 0 from Table 21 01h: Nonce for attestation algorithm 1 from Table 21 02h: Nonce for attestation algorithm 2 from Table 21 03h: Nonce for attestation algorithm 3 from Table 21 04h: Nonce for attestation algorithm 4 from Table 21 05h: Nonce for attestation algorithm 5 from Table 21 06h: Nonce for attestation algorithm 6 from Table 21 07h: Nonce for attestation algorithm 7 from Table 21 08h to 13h: Reserved for future algorithms 14h: Nonce for attestation algorithm 20 from Table 21 15h: Nonce for attestation algorithm 21 from Table 21 16h: Nonce for attestation algorithm 22 from Table 21 17h: Nonce for attestation algorithm 23 from Table 21 Others: Reserved. Note: The top 3-bits of this of this 8-bit field are expected to always be 0. Hardware can treat them as reserved, set to 000.
N3	Write Primary Address 18h “security action details [1]”	Reserved. Set to 00h.
N5	Write Primary Address 19h “security action request”	Program to 0Fh: Take “Fetch Nonce” action
N8	Read Primary Address 15h “security action status”	Read for target nonce request operation status. Values are set per Section 7.6.

Step of Figure 12	Action	Description
N10	Read Buffer Region 1	Target-Generated Nonce Note: that security action status will be cleared to 00h once any byte is written to security buffer region 00h following this command. This ensures that the data in this region is valid and unaltered. Any attempt to perform the "next security action command" will not be prohibited until a new nonce is obtained. Byte ordering within the memory region is to be given in the device literature.
N11	Read Primary Address 15h "security action status"	Ensure the target nonce that was read remains valid by checking the operation code remains successful and not idle. If idle, then it is possible the results read were tampered with by an aggressor's write to the buffer region. Values are set per Section 7.6.
Section 9.1	Next Security Action Command	The next security action request will utilize the nonce contained within this response in combination with random data and its command to determine the expected MAC. Target IC will verify this expected MAC before carrying out the request.

10. NVM / Firmware Authentication And Update

Any firmware or configuration data written to PMBus, hereafter collectively referred to as firmware, must be signature verified within the PRoT prior to committing to either the volatile or non-volatile firmware or configuration memory within the PMBus target. The firmware image to be loaded onto the PMBus target shall be signed by the system integrator with their private key. The PRoT carries and securely stores the public key corresponding to that private key. The integrity of the public key must be protected by the PRoT.

If either (a) the PMBus target is not pre-loaded with a full NVM image or (b) a PMBus target firmware update is requested, then the PRoT must verify the new firmware's signature using the public key stored securely within the PRoT. If the firmware does not pass authentication, the update flow is blocked, and the existing firmware shall continue to remain active. It must not be possible to generate a hardware "denial of service" attack via an improperly authenticated firmware update. New firmware must be validated by the PRoT prior to overwriting the existing firmware. To preserve the security assurances provided by this authentication flow, in normal operation no adjust-single-NVM-byte (PMBus STORE/RESTORE) commands are allowed as they would bypass the signature.

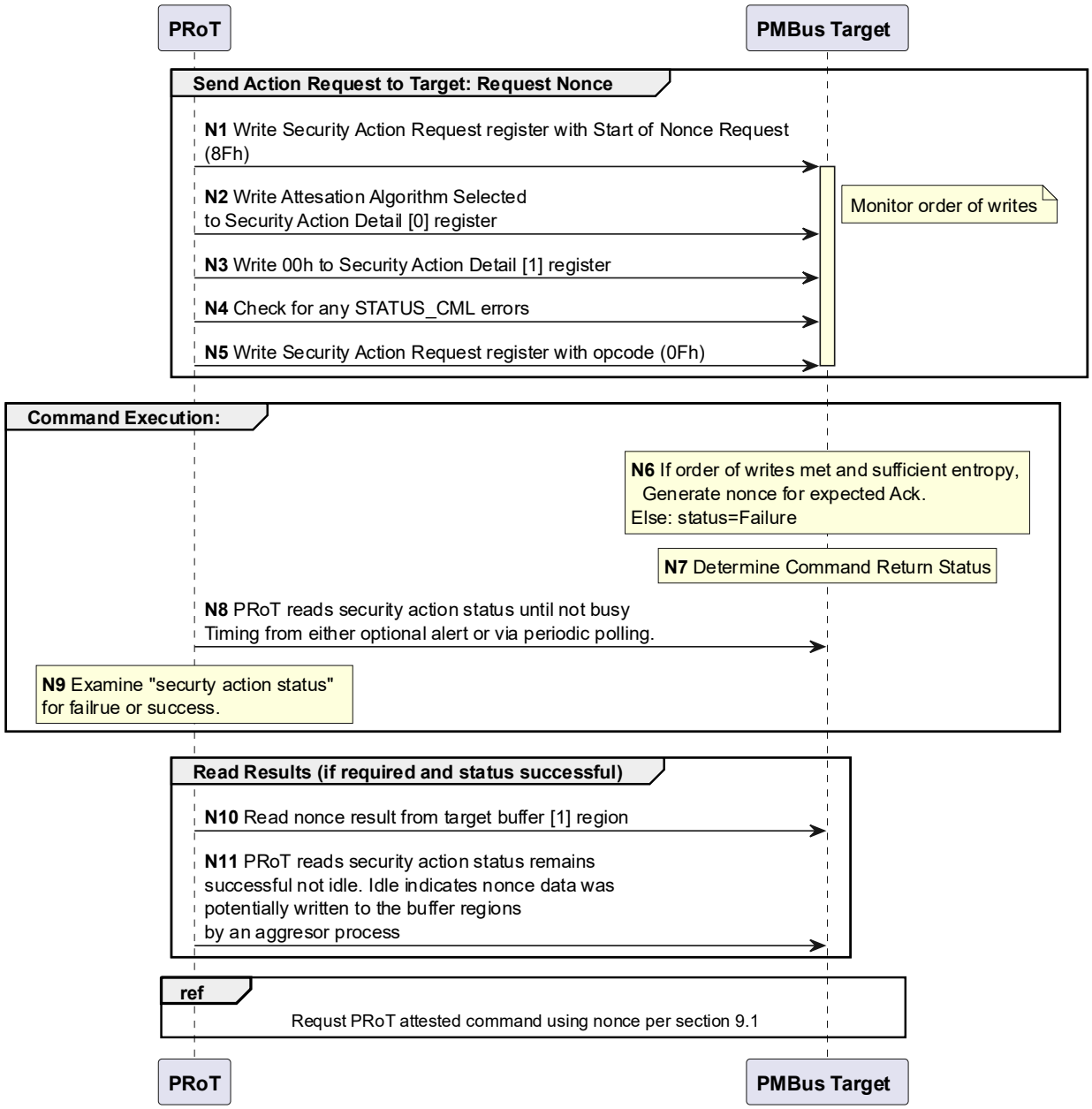


Figure 12. Nonce Request For PRoT Attested Command

This firmware signature verification process ensures the firmware to be loaded onto the PMBus target is a properly signed firmware image from the system integrator. After uploading the new firmware image to the PMBus target, the attestation process of Section 8 is utilized to ensure that the signed firmware is what was loaded into the target and that no modifications were made during the transfer from PRoT to PMBus target. If a firmware update request does not pass the attestation, the system shall not command the target voltage regulator to enable the boot flow until the proper firmware update can be authenticated and attested.

The process imposes the following hardware requirements on the PRoT:

- Secure storage with integrity protection within the PRoT for the public key corresponding to the private key used for signing the NVM.

- Attestation algorithm support from Table 21.
- Secure storage and update in the PRoT and PMBus target for pre-shared keys.
- A strong private/public key signature verification algorithm in the PRoT for all levels. An example of such an algorithm is module-lattice-based digital signature standard (ML-DSA signature), as defined in FIPS Pub 204 [A07] for all levels. Digital signatures such as those in FIPS publication 186-5 [A14] or other algorithms may be acceptable for specific situations. The selected algorithm, its standard, and its strength shall be clearly stated in device datasheets.

Having this secure upload capability enables the potential to have low-NVM PMBus targets in future systems where the firmware is updated into volatile memory by the PRoT at each boot and attested to by the target prior to output enabling.

For security profile level 1 in addition to this PRoT-computed signature verification done, it is required for the PMBus target to have rudimentary password protection and checksum to provide some coverage against bit errors during transmission of firmware over PMBus to the target.

For security profile level 2 in addition to the PRoT-computed signature verification done, it is required to run the attestation flow in the PMBus target against the candidate firmware. This verifies that the image was not manipulated in transmission but will not provide protection against a compromised PSK. In this approach, the MAC calculated by the PRoT must be transmitted to the PMBus target to be compared with the target-calculated value prior to committing the candidate image to NVM, enabling the target to make an accept or reject decision about the firmware.

For security profile level 3, the PMBus target itself also required to verify the PMBus target firmware and configuration file image using the same signature-verification algorithm utilized in the PRoT.

There are two classes of firmware and/or configuration updates. One class contains items that must not be updated while a critical function, such as voltage regulator output, is active. For a voltage regulator, this would contain items that affect loop dynamics such as filter bandwidths or internal gain settings within the controller. Such authenticated update requests that could create physical instability of the voltage regulator during the update shall be blocked by the VR itself while the output is active. The second class contains items which can be updated anytime. This includes items such as the software to implement certain PMBus functions implemented on an embedded microcontroller, such as the security functions. The second class of updates is permitted to be applied immediately regardless of if device output is active or not. For each class, the updates may be applied in either a volatile manner or to the non-volatile memory.

These two different classes imply the need for segmented memory updates, where updates are applied to portions of the volatile or non-volatile program memory. Segmented memory updates may also be needed to allow updates with limited RAM available on the VR to temporarily store candidate firmware until it is authenticated.

If the PMBus target authenticates updates using the whole firmware/memory image instead of just the segment to be updated, then non-volatile updates shall be applied only immediately following a PMBus target power-on or reset cycle and must be prior to any volatile-only updates. This requirement avoids any ambiguity about whether to use the volatile-only or non-volatile firmware contents to use when performing authentication calculations. An exception is allowed where parts can commit all previously committed

volatile segments along with a new non-volatile segment to NVM. It may be optional, but permissible, to allow simultaneous update to volatile and non-volatile memory.

If the PMBus target has regions that cannot be updated while the VR output is on, then any attempt to request an update to such regions while the output is on are to be rejected.

There is a register, described in Section 12.3.1, that indicates how many authenticated updates remain available. When this register reaches 0, there are no more updates to non-volatile memory remaining. The meaning of non-zero values from 1 to 6 is left to the manufacturer to determine. Value 7 means that the non-volatile firmware updates are effectively unlimited.

If a firmware function is attempted while its contents are being updated, it is permissible to reject the command using STATUS_CML “Other Communication Fault Error”.

10.1 NVM Firmware Authentication And Update Protocol Summary

The process for updating the NVM firmware is illustrated in Figure 13.

During a firmware update, the PRoT must read the firmware image to be pushed to the target (step F1). The PRoT must validate the signature associated with the firmware image utilizing the public key stored in its secure storage (step F2). Should that signature pass, the PRoT then will push the firmware image to be updated out to the PMBus target. The process by which the new firmware image is pushed to the PMBus target changes depending on the security profile.

When the firmware image is large relative to available memory, this process can be carried out iteratively over multiple smaller segments. The PRoT informs the target that it will be performing an authenticated update (step F8) and writes parameters specific to that authentication including what firmware segment is to be updated and if that update is written to volatile or non-volatile memory (step F9-F10).

For all security profile levels, the PRoT will send the firmware update (F11) to the PMBus target. Note: This upload only includes the new or modified segment if updating iteratively.

If the PMBus is security level 2, the PRoT generates a nonce using the DRBG (step F3). The PRoT performs the attestation calculations of section using the nonce, the shared PSK, the candidate firmware update, and attestation algorithm as selected from Table 21. Note: if performing update in segments, the MAC calculation is performed using (a) the new firmware for segments that are either already updated or that are being updated in this pass and (b) the old firmware for segments that have not yet been updated (step F3-F6). In this case, the PMBus target is the device that permits the update if the expected MAC delivered by the PRoT matches its own calculation. The PMBus target must know the nonce and MAC calculation result from the PRoT. These are transmitted to the PMBus target (step F12-F13).

If the PMBus is security profile level 1, the PRoT transmits the password and checksum to the target (step F12-F13).

After the programming time allotment expires (step F17), the PRoT will query the PMBus target to determine if programming was successful (step F18). If the programming was successful, the PRoT updates the attestation calculations (step F19).

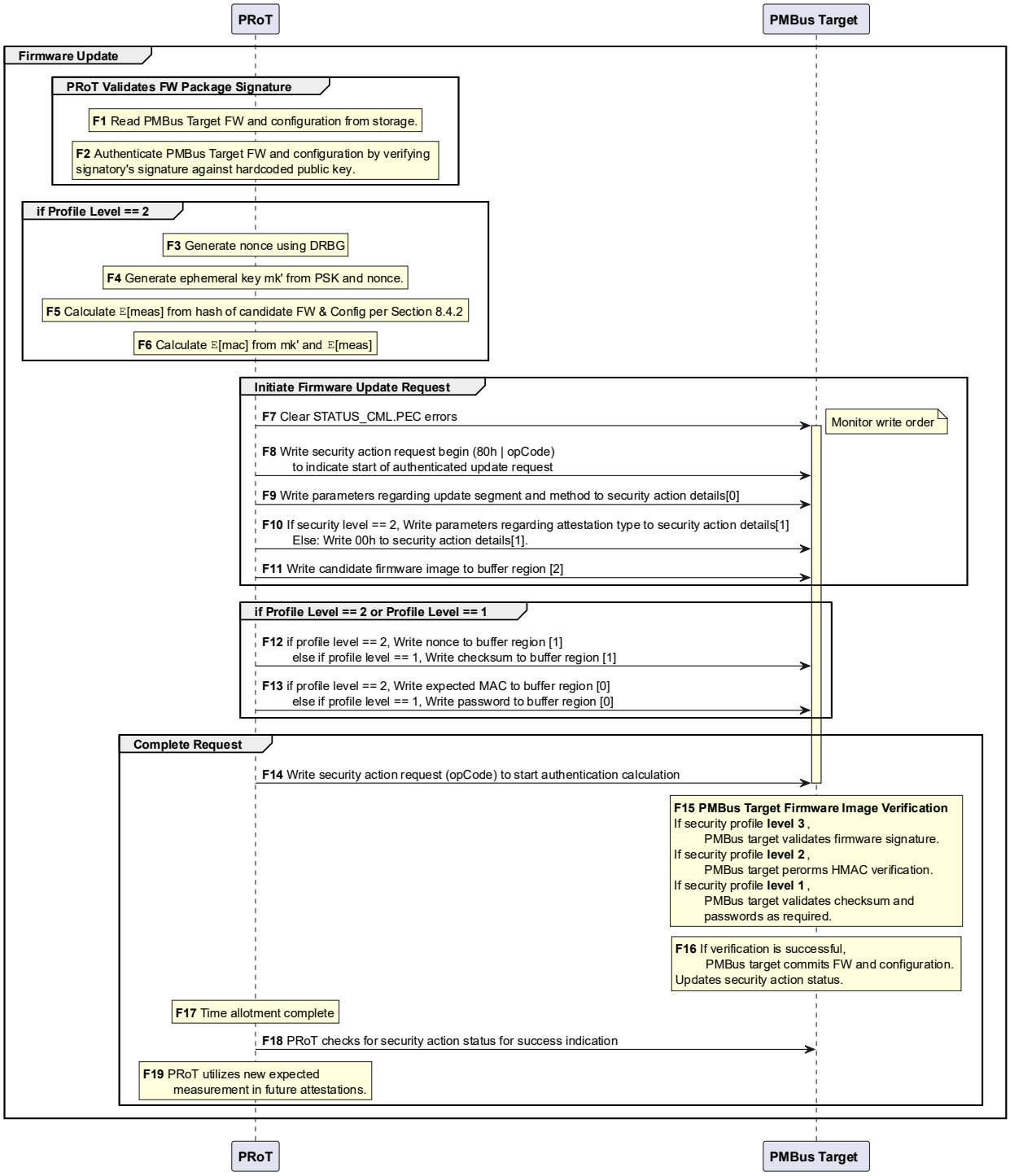


Figure 13. Authenticated Firmware Update Flow

10.2 Firmware Authenticated Update Request

This section defines a procedure for sending to the VR target: candidate firmware updates, additional data required for authentication, the memory segment to be updated, and whether that update is to be applied to volatile memory, non-volatile memory, or both.

The structure can individually write up to 64 different segments of memory. Each segment has a size limit of a theoretical 16 megabytes (2^{24} = 16,777,216 bytes). It should be noted that there must be a buffer memory equal to the size of the memory segment to allow the data to be authenticated before it is committed, so in many situations using segments reduces the memory required to stage and authenticate the new firmware image prior to committing it. Another reason for splitting up firmware into segments is that some segments may have different programming constraints. To use a voltage regulator as an example, it may be possible to update the PMBus transaction memory in one segment while the regulator output is on. Updating of the voltage regulator control loop dynamics, such as filter bandwidth or gain settings, may produce an unknown transfer function as the controller poles and zeros migrate. This segment of memory should only be updated to NVM only when the VR output is deactivated.

The first step the PRoT must perform when requesting an authenticated firmware update is to perform signature verification upon the incoming candidate firmware image. The signing mechanism used to authenticate the image coming into the PRoT shall be at least as strong as the valid signing mechanisms as approved for security profile level 3 on the VR.

The second step when updating a PMBus firmware segment, is to load the candidate firmware segment contents into the buffer region 2 using the large memory region access techniques outlined in Section 6.3. Once the candidate firmware image is uploaded to the PMBus target, the next thing that must be transferred is any data required for authentication. The authentication data must be placed into the buffer regions as shown in Table 10.

After both the candidate firmware segment and required authentication items are loaded into the PMBus target's shared security memory buffers, the PRoT can request the firmware update begin. The first part of requesting the firmware authentication update is to specify the details of the update request. Those details include the memory region to be updated and, if using security level 2, the method to use for the attestation calculation.

Devices must be able to limit the number of failed Authenticated Updates allowed per Power Cycle to reduce sensitivity to brute-force attacks. It is recommended that the limit of consecutive failed attacks be set to 8, with the counter cleared to 0 upon a successful update prior to the limit value. If devices limit the number of failed Authenticated Updates, the counter shall only reset on Power Cycle and shall not reset on RESTORE or on Power-On-Reset. If an Authenticated Update action is requested after the failed attempt limit has been reached, the device shall respond as: "Failed, Unable to attest PMBus host (PRoT): MAC failure - Calculation result mismatch" per Table 12.

Table 32. Commands To Perform Security Update Requests

Step in Figure 13	Relevant Bit Fields	Bits	Description
F8	Write Primary Address 19h “security action request”	7:0	Inform the target of the start of data transfer for an authenticated update at a given security level: • 80h: Security Level 0 • 81h: Security Level 1 • 82h: Security Level 2 • 83h: Security Level 3
F9	Write Primary Address 17h “security action details [0]”	MSB [7]	0: Update volatile firmware memory only 1: Update non-volatile firmware memory Targets may elect to make non-volatile firmware update available only when VR output is not enabled. Those requests would respond with operation blocked action status). Security action details [1] bit [6] determines if volatile memory is also updated.
		[6]	0: Authentication calculations are computed using the full NVM or firmware image within the chip 1: Authentication calculations are computed using only the segment to be updated.
		[5:0] LSB	00h to 3Fh: Firmware segment to update.
F10	Write Primary Address 18h “security action details [1]”	MSB [7]	0: Has no effects on prior volatile commits 1: If applied with a non-volatile firmware commit (security action details [0] bit [7] == 1) this extends that commit to push previously authenticated volatile-only updates into their non-volatile storage. This runs only if the existing commit action passes authentication.
		[6]	If applied with a non-volatile firmware commit (security action details [0] bit [7] == 1): 0: NVM update does not also induce a commit to volatile memory of the new firmware image 1: NVM update also induces a commit to the corresponding volatile segment. If applied with a volatile firmware commit (security action details [0] bit [7] == 0): Reserved, set to 0.
		[5]	Reserved, set to 0.

Step in Figure 13	Relevant Bit Fields	Bits	Description
		[4:0] LSB	If requesting security profile level 2 update: *Attestation algorithm used to authenticate candidate firmware image:* 00h: Use attestation algorithm 0 from Table 21 01h: Use attestation algorithm 1 from Table 21 02h: Use attestation algorithm 2 from Table 21 03h: Use attestation algorithm 3 from Table 21 04h: Use attestation algorithm 4 from Table 21 05h: Use attestation algorithm 5 from Table 21 06h: Use attestation algorithm 6 from Table 21 07h: Use attestation algorithm 7 from Table 21 08h to 13h: Reserved for future algorithms 14h: Use attestation algorithm 20 from Table 21 15h: Use attestation algorithm 21 from Table 21 16h: Use attestation algorithm 22 from Table 21 17h: Use attestation algorithm 23 from Table 21 Others: Reserved. Else: Reserved, set to 00h.
F11	Write Buffer region [2]	-	Write firmware update package for selected buffer region into buffer region [2] Byte ordering within the memory region is to be given in the device literature.
F12	Write Buffer region [1]	-	If security level 1 update: Write checksum to buffer region [1] If security level 2 update Write nonce to buffer region [1] If security level 3 update: Do not write buffer region [1] Byte ordering within the memory region is to be given in the device literature.
F13	Write Buffer region [0]	-	If security level 1 update: Write password to buffer region [0] If security level 2 update Write expected MAC to buffer region [0] If security level 3 update: Do not write buffer region [0] Byte ordering within the memory region is to be given in the device literature.

Step in Figure 13	Relevant Bit Fields	Bits	Description
F14	Write Primary Address 19h “security action request”	[7:0]	Request the authenticated update to a given security level: • 01h: Security Level 1 • 02h: Security Level 2 • 03h: Security Level 3

10.3 Data Commit Status

While the PMBus is updating firmware in response to an authenticated firmware request, the PMBus “security action status” will read “Operation in Progress” per Table 12 to indicate the update remains in progress. When the authenticated firmware update is done, the “security action status” byte changes value to indicate if the update completed successfully or failed per Section 7.6. It is allowable to NACK or assert STATUS_CML invalid command as the firmware is being updated if the firmware of the PMBus interface engine itself is what is being updated.

10.4 Reboot Cycle

These commands were added to allow power-on-resetting a PMBus target while its output is turned off. The first command is optional and can be used to block a reset command until all update portions are completed. Once set, it remains throughout the duration of the power-on-reset cycle.

10.4.1 Lockout Power-On-Reset Function

The primary purpose of this optional function is to prevent a reboot instruction from being called with a partially updated NVM image. This command ensures that all requested firmware segments have successfully completed authenticated updates before a reboot is allowed. It is particularly useful for scenarios where an aggressor calls a reboot between NVM firmware segments could leave a device in an unusable state.

To make sure this reboot command modifying command is applied by an attested host, it is issued with a nonce and an attestation MAC. The data to be loaded to the target prior to calling this function is in Table 33, and the flow follows the procedure provided for requests where the PRoT must be attested from Figure 11.

In response to receiving the security action request in step R15, the PMBus target will update the security action status register as per Section 7.6.

Table 33. PMBus Reboot Prevention Data Preparation

Step from Figure 11	Action	Description
Not Shown	Select power-on-reset gate mode	Select whether power-on-reset is gated by the VR output being on.
Not Shown	Select firmware segments that gate power-on reset	Determine which firmware segments must be updated before permitting a power-on-reset to apply an update.

Step from Figure 11	Action	Description
R1-R8	Perform PRoT attestation calculations as required per Section 9.	Request a random nonce and a PRoT attestation algorithm from the target using the process described in Section 9.2. Read the nonce from buffer region [1]. Use the reboot prevention power-on-reset gate option and required firmware segments to calculate the expected MAC per Table 30. This expected MAC will be pushed to the target later in this process to help attest that the PRoT is the issuer of the power-on-reset gate request (reboot lockout).
R9	Write Primary Address 19h “security action request”	8Ah: Request reboot lockout command, start command sequence
R10	Write Primary Address 17h “security action details [0]”	Configures when power-on-reset requests may be issued. 00h: Power-on reset request allowed when the firmware update conditions are met, and the VR output is off. 01h: Power-on-reset request is allowed when firmware update conditions are met, even if the VR output is on. Else: Reserved
R11	Write Primary Address 18h “security action details [1]”	Reserved, program to 00h.
R12	Write Buffer Region [2]	Each segment with a 1 must have received a successful authenticated firmware update before a reboot is permitted. If there is to be no requirements on a subset of segments being completed, then set all bits to 0. • Byte 0 bit 0: FW segment 0 NVM must be updated • Byte 0 bit 1: FW segment 1 NVM must be updated • Byte 0 bit 6: FW segment 6 NVM must be updated • Byte 0 bit 7: FW segment 7 NVM must be updated • Byte 1 bit 0: FW segment 8 NVM must be updated *Repeated up to the maximum firmware segment index of 63. There can be fewer if fewer segments are implemented.*
-	Skip Buffer Region [1]	Nonce generated from Target Nonce Request. Do not write this location or it will be invalidated.

Step from Figure 11	Action	Description
R13	Write Buffer Region [0]	Expected MAC: Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.
R15	Write Primary Address 19h “security action request”	0Ah: Request reboot lockout command, apply command.

10.4.2 Output-Off Power-On-Reset Function

The primary purpose of this instruction is to trigger a re-load from the new NVM if required following an NVM authenticated firmware update. This is analogous to restarting a desktop computer operating system after applying an update. If the optional command described in Section 10.4.1 is implemented, this command is blocked if not all required NVM segments are updated.

This reboot can operate without attesting the PMBus controller, but it is only available when the target’s output rails are off. Any attempt to call output-off power-on-reset function while output is enabled will fail. The command is requested per the sequence of Table 34. The timing of the power off and subsequent output-enable sequence is dictated by the PMBus initialized state following the reset. In response to the output-off power-on-reset request, the status register is updated per Section 7.6.

The security actions status will return to Idle “00h” state after the reboot has completed. For this reason, the command sequence of Table 34 indicates writing the security action details registers to FFh. When the reset is complete, these registers will return to 00h giving another indication that reset was successfully completed.

Power-On-Reset Security Action Request will not reset the failed attempts per Power Cycle counters for PASSKEY or Authenticated Updates, if supported.

Table 34. Output-Off Power-On-Reset Request Sequence

Step from Figure 5	Memory Location	Data
A1	Write Primary Address 19h “security action request	87h: Begin Output-off Power-on-Reset request sequence
A2	Write Primary Address 17h “security action details [0]”,	Reserved, write to FFh. This value will be set back to 00h after completion of reset so it provides a means to check when reset has completed outside of the status for which idle and reset both return 00h.

Step from Figure 5	Memory Location	Data
A3	Write Primary Address 18h “security action details [1]”	Reserved, write to FFh. This value will be set back to 00h after completion of reset so it provides a means to check when reset has completed outside of the status for which idle and reset both return 00h.
-	Skip Buffer Region [2]	Data not needed for this action request
-	Skip Buffer Region [1]	Data not needed for this action request
-	Skip Buffer Region [0]	Data not needed for this action request
A8	Write Primary Address 19h “security action request”	07h: Output-off Power-On-Reset

10.4.3 Anytime Power-On-Reset Request Function

The primary purpose of this optional function is to provide a secured means to allow an authorized PMBus agent to request a PMBus target to reboot, even while its output is turned on. Once a reboot is requested, it is possible that it may be blocked by either an all-out lockout or a firmware-update gated lockout. If a lockout is present, the reboot request will fail and indicate via “security action status” that it is blocked.

To ensure the power-on-reset command is requested by a permitted agent, the command attests the controller identity per Section 9. If the attestation fails, then the command fails, and the failure cause is reflected in the security action status register. If the attestation passes and no blocking conditions are present, the PMBus device will initiate the power-on reset.

If the output was active when the anytime power-on reset was called, then the power-on-reset first executes the ramp down and wait sequence described in the rest of this paragraph. First, it waits for the time specified by PMBus field TOFF_DELAY prior to starting the ramp down of the rail. This time window allows for sending power-on-reset requests to other PMBus targets as part of the shutdown process. After TOFF_DELAY elapses, the voltage regulator output ramps down per the PMBus TOFF_FALL value. After the TOFF_FALL ends, the output will remain off for the TON_DELAY time.

If either (1) the power-on-reset was called during output-off, or (2) the power-on-reset successfully completed the ramp-down sequence in the prior paragraph, then the PMBus will reset the registers and execute the reset. It activates per the combination of PMBus settings that govern initial power on, that may include the value of key signals such as TON_DELAY, TON_RISE, and if applicable, the VR’s enable pin.

The anytime power-on reset request’s effect on an example VR is shown in Figure 14. This command sequence is listed in Table 35. In response to the output-off power-on-reset request, the status register is updated per Section 7.6.

Power-On-Reset Security Action Request will not reset the failed attempts per Power Cycle counters for PASSKEY or Authenticated Updates, if supported.

Table 35. Preparing Data For Attested PRoT Anytime Power-On-Reset Request

Step from Figure 11	Action	Description
R1-R8	Perform PRoT attestation calculations as required per Section 9.	Request a random nonce and a PRoT attestation algorithm from the target using the process described in Section 9.2. Read the nonce from buffer region [1]. Calculate the expected MAC per Table 30. This expected MAC will be pushed to the target later in this process to help attest that the PRoT is the issuer of the anytime power-on-reset request.
R9	Write Primary Address 19h “security action request”	8Ch: Start “Anytime power-on reboot command” request sequence
R10	Write Primary Address 17h “security action details [0]”	Reserved, write to FFh. This value will be set back to 00h after completion of reset so it provides a means to check when reset has completed outside of the status for which idle and reset both return 00h.
R11	Write Primary Address 18h “security action details [1]”	Reserved, write to FFh. This value will be set back to 00h after completion of reset so it provides a means to check when reset has completed outside of the status for which idle and reset both return 00h.
-	Skip Buffer Region [2]	Do not write. Not used by this security action.
-	Skip Buffer Region [1]	Contains Nonce generated from Target Nonce Request. Do not write this location or it will be invalidated.
R13	Write Buffer Region [0]	Expected MAC Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.
R15	Write Primary Address 19h “security action request”	0Ch: Anytime power-on reboot command

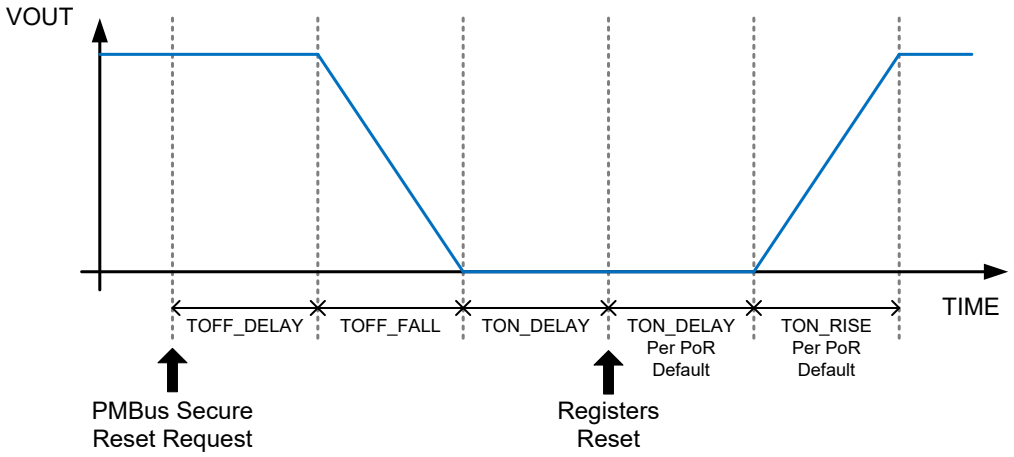


Figure 14. Always-On Power-On Reset Profile With Output On At Time Of Request

10.5 Firmware Fetch Commands

These optional commands facilitate the ability to read out the firmware contents and to lock out the firmware contents from being readable. These commands are meant for device verification. In a deployed system, attestation will validate the firmware image loaded into the target.

10.5.1 Firmware Fetch

This optional command causes the PMBus to fetch the firmware contents from a given memory location and load it into the PMBus security memory buffer region 2. This can then be read via the read-write commands defined in Section 6.3. The purpose of this is to allow reading back to ensure the firmware image was properly loaded and to validate the memory locations for test. The memory test could be alternately covered via attestation. This may be used primarily for device testing and would likely be disabled via fuse for shipped parts. It must be impossible by design to read back the pre-shared key using this command.

Table 36. Firmware Fetch Action Request And Status Codes

Step from Figure 5	PMBus security memory map register and address	Bit(s)	Description
A1	Write Primary Address 19h “security action request”	7:0	88h: Start firmware fetch request sequence
A2	Write Primary Address 17h “security action details [0]”	7:6	Reserved, set to 0.
		5:0	Memory Segment to copy into the PMBus secure device buffer region 2 memory. Can be from 0 to 63 (3Fh)
A3	Write Primary Address 18h “security action details [1]”	[7:0]	Reserved. Set to 00h.

Step from Figure 5	PMBus security memory map register and address	Bit(s)	Description
-	Skip Buffer Region [2]	-	Data not needed for this action request
-	Skip Buffer Region [1]	-	Data not needed for this action request
-	Skip Buffer Region [0]	-	Data not needed for this action request
A8	Write Primary Address 19h “security action request”	[7:0]	08h: Firmware Fetch Request.

Note: In response to a firmware fetch request, the security action status register is updated per Section 7.6.

10.5.2 Firmware Fetch Lockout Command

An optional command exists to permanently lock-out the firmware fetch. Once it is locked, the lock bit can only be released by a security level 2 or security level 3 authenticated update that overwrites its memory region. All calls for this action must attest the PMBus controller making the request. The controller attestation operation works as described in Section 9. The attestation process consumes security buffer memory regions [0] and [1].

Table 37. Buffer Data To Engage Firmware Fetch Lockout

Step from Figure 11	Action	Description
Not Shown	Select Fetch Lockout Duration	Choose value to be loaded into security action details [0] as it will be part of the expected MAC calculation in next step.
R1-R8	Perform PRoT attestation calculations as required per Section 9.	Request a random nonce and a PRoT attestation algorithm from the target using the process described in Section 9.2. Read the nonce from buffer region [1]. Calculate the expected MAC per Table 30. This expected MAC will be pushed to the target later in this process to help attest that the PRoT is the issuer of the anytime power-on-reset request.
R9	Write Primary Address 19h “security action request”	Program to 89h: Start Firmware fetch lockout command sequence

Step from Figure 11	Action	Description
R10	Write Primary Address 17h “security action details [0]”	Program the value: 00h: Set update to this value to lockout fetch operation for the remainder of this controller power-on reset cycle and all subsequent power cycles 01h: Set update to this value to lockout fetch operation for the remainder of this controller power-on reset cycle 02h to FFh: Reserved for future applications.
R11	Write Primary Address 18h “security action details [1]”	Reserved, program to 00h.
-	Skip Buffer Region [2]	Not needed for this security action request
-	Skip Buffer Region [1]	Contains Nonce generated from Target Nonce Request. Do not write this location or it will be invalidated.
R13	Write Buffer Region [0]	Write Expected MAC Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.
R15	Write Primary Address 19h “security action request”	Program to 09h: Starts Firmware fetch lockout command execution,

Note: In response to a firmware fetch lockout command, the security action status register is updated per Section 7.6.

11. PMBus Command Access Controls And Manufacturer Mode

This section of the security document discusses two features. The first feature is a secured version of PMBus access control called “Secure Access Control Action” that executes on a list of command and permission pairs. The second feature regards entering a manufacturing mode to allow PMBus target manufacturers to deal with returned units.

11.1 Secured Access Control Action Request

The secured “access control” action works similarly to the PMBus native ACCESS_CONTROL command while adding PRoT attestation to validate the requestor is authentic. To simplify usage, the secured access control action accepts a list of data “target command code” and “access control byte” pairs. This way, many access control

commands can be requested while calculating only a single attestation calculation. The command code and permission pairs are passed via the PMBus secure device buffer region 2. The security action details [0] register contains the number of pairs being requested. This process uses the PRoT attestation process of Section 9, so buffer region [1] is contains a target generated nonce, and buffer region [0] is used for the expected MAC result from the target.

When in PMBus secure device operation, the traditional PMBus ACCESS_CONTROL command shall be inaccessible, replaced with this command authorized with the PMBus host (PRoT) attestation mechanism. The command operates per the PMBus Host attestation command flow of Section 9 with the data elements for the transaction given by Table 38. Unlike directly accessing the PMBus ACCESS_CONTROL command or accessing it with a PAGE_PLUS command with host attestation, this security action does bypass the PASSKEY check as the attestation of host is stronger than a password transmitted in the clear.

If access control list has some commands that can be executed and other command-code/access pairings that cannot be realized, the error code for “Failed: Operation is Blocked” from Table 12. PMBus Security Action Status Error Codes will be presented. Those pairs that with legally allowed values will be applied, despite the partial failure.

Table 38. Attested Access Control Command, Write

Step from Figure 11	Action	Description
Not Shown	Select command and new access permissions	Determine the number of command/access pairs to be modified. The number of pairs and the data in the pairs shall be used in the expected MAC calculation.
R1-R8	Perform PRoT attestation calculations as required per Section 9.	Request a random nonce and a PRoT attestation algorithm from the target using the process described in Section 9.2. Read the nonce from buffer region [1]. Using the pair data, calculate the expected MAC per Table 30. This expected MAC will be pushed to the target later in this process to help attest that the PRoT is the issuer of the attested access control command.
R9	Write Primary Address 19h “security action request”	8Eh: Start “Secured Access Control Action” request sequence.
R10	Write Primary Address 17h “security action details [0]”	Number of Access Control Pairs in buffer region 2
R11	Write Primary Address 18h “security action details [1]”	00h. (Reserved)

Step from Figure 11	Action	Description
R12	Write Buffer Region [2]	This will contain two-byte pairs starting at address 000000h. The number of two-byte pairs to consider is listed in Primary Address 17h “security action details [0]”. For each data byte pair, n, starting from pair 0 and working up: • Byte at buffer region [2] addr [2n+0] is the target command code. • Byte at buffer region [2] addr [2n+1] is the access control byte. The access control byte is defined as per PMBus specification traditional ACCESS_CONTROL command, with the definition at time of publishing. Note this method does bypass the PASSKEY check as the PRoT attestation is stronger than a password transmitted in the clear.
-	Skip Buffer Region [1]	Do not write. This contains the PRoT requested nonce
R13	Write Buffer Region [0]	Expected MAC: Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.
R15	Write Primary Address 19h “security action request”	0Eh: Secured Access Control Action command. Complete data loading, request command execution.

In response to a secured access control action request, the security action status register is updated per Section 7.6.

Table 39 gives a summary of the definition of the bits for the ACCESS_CONTROL data byte. The ACCESS_CONTROL command is defined in Part II of the PMBus specification [A02]. Table 39 is provided for reference only. If there are any differences between the descriptions in Table 39 and Part II, Part II is definitive.

Table 39. Access Control Byte Definition From PMBus Specification 1.5 Part II

Access Item	Bit	Bit Function
Write Access	7	When set, the command code or command code group will treat write access attempts as invalid data

Read Access	6	When set, the command code or command code group will treat read access attempts as invalid data
Disable Secure Attested Access	5	When set, the command code or command code group will be blocked from being used via the secure access control action mechanism. If 0, then the host attested access commands can utilize the command. Bit [5] operates upon security “Secured Access Control Action” opcode 0x0E as a PMBus ACCESS_CONTROL write would be affected by bit [7]. Similarly, Bit [5] operates on “PMBus PAGE_PLUS_READ or PAGE_PLUS_WRITE with Host Attestation” opcode 0x0D, allowing properly attested commands bypass Write Access permission bit [7].
NVM Store	4	When set at Power On Reset, STORE commands will not update the NVM stored value for the command code or command code group. **Note:** This must always be set to 1 to block STORE commands in PMBus secure device applications.
NVM Restore	3	When set, RESTORE commands will not update the current value of the command code or command code group with the value from the NVM store. **Note:** This should always be set to 1 to block RESTORE commands in PMBus secure device applications.
Write Once	2	When set at Power On Reset, the command code or command code group will allow write access once, then treat all additional write access attempts as invalid data.
No More	1	When set, the Access Control Byte for this command code or command code group will not be altered by a write to e21 to any command code within the group, even if PASSKEY is unlocked. This bit should default to 0 and thus should not be stored in NVM. If set to 1 at power on, there is no way to clear this bit and enable write access to ACCESS_CONTROL for this command or command group, duplicating the functionality of Never Again [0]
Never Again	0	When set at Power On Reset or RESTORE, the Access Control Byte for this command code or command code group will not be altered by a write to ACCESS_CONTROL to any command code within the group, even if PASSKEY is unlocked. Note that writes to this command are not applied to NVM. Commands must be stored to NVM via authenticated updates.

11.2 Manufacturer Mode

Parts can have a manufacturer mode to allow debug of factory returns without needing to know the PSK loaded on that specific device. The method used to enter manufacturer mode shall require physical presence and the introduction of non-trivial signals that cannot be driven in a production system without a physical presence.

12. Miscellaneous Commands and Options

12.1 Version Reporting

12.1.1 PMBus Security Profile Version Command

These bytes are used to inform the PRoT which PMBus security profile level is supported by the PMBus target device. These bytes are located at addresses 02h and 03h within the PMBus device address space. These data bytes are read-only. The values are as defined in Table 40.

Table 40. Security Profile Support

Relevant Bit Fields	Contents
Primary Address 02h "Security Profile [0]"	00h: No security profile supported 01h: PMBus security level 1 profile support: Password/Checksum 02h: PMBus security level 2 profile support: Attestation-based Authentication 03h: PMBus security level 3 profile support: Signature Verification.
Primary Address 03h "Security Profile [1]"	Reserved, set to 00h.

12.1.2 PMBus Firmware and Configuration Version

These two bytes are used to identify the PMBus target firmware and configuration file versions. They are also used to prevent re-introduction of older firmware by making sure that the candidate firmware and configuration version number in any candidate firmware is no lower than the presently populated version. Hardware must enforce this restriction that firmware and configuration version must either be the same or higher for an update to be accepted. Beyond the targets' expectation that these bytes be monotonically increasing in value, the methodology is left to the supplier to determine. Bitfield definitions are given in Table 41. An example of how to manage firmware version number during an authenticated update completed during several segments is shown in Figure 15. It is permissible to have a more complex method than just version number comparison to prevent loading of potentially compromised firmware.

Table 41. Firmware And Configuration ID

Relevant Bit Fields	Contents
Primary Address 04h "Firmware And Configuration Version [0]"	Least significant byte of the firmware and configuration version ID
Primary Address 05h "Firmware And Configuration Version [1]"	Most significant byte of the firmware and configuration version ID

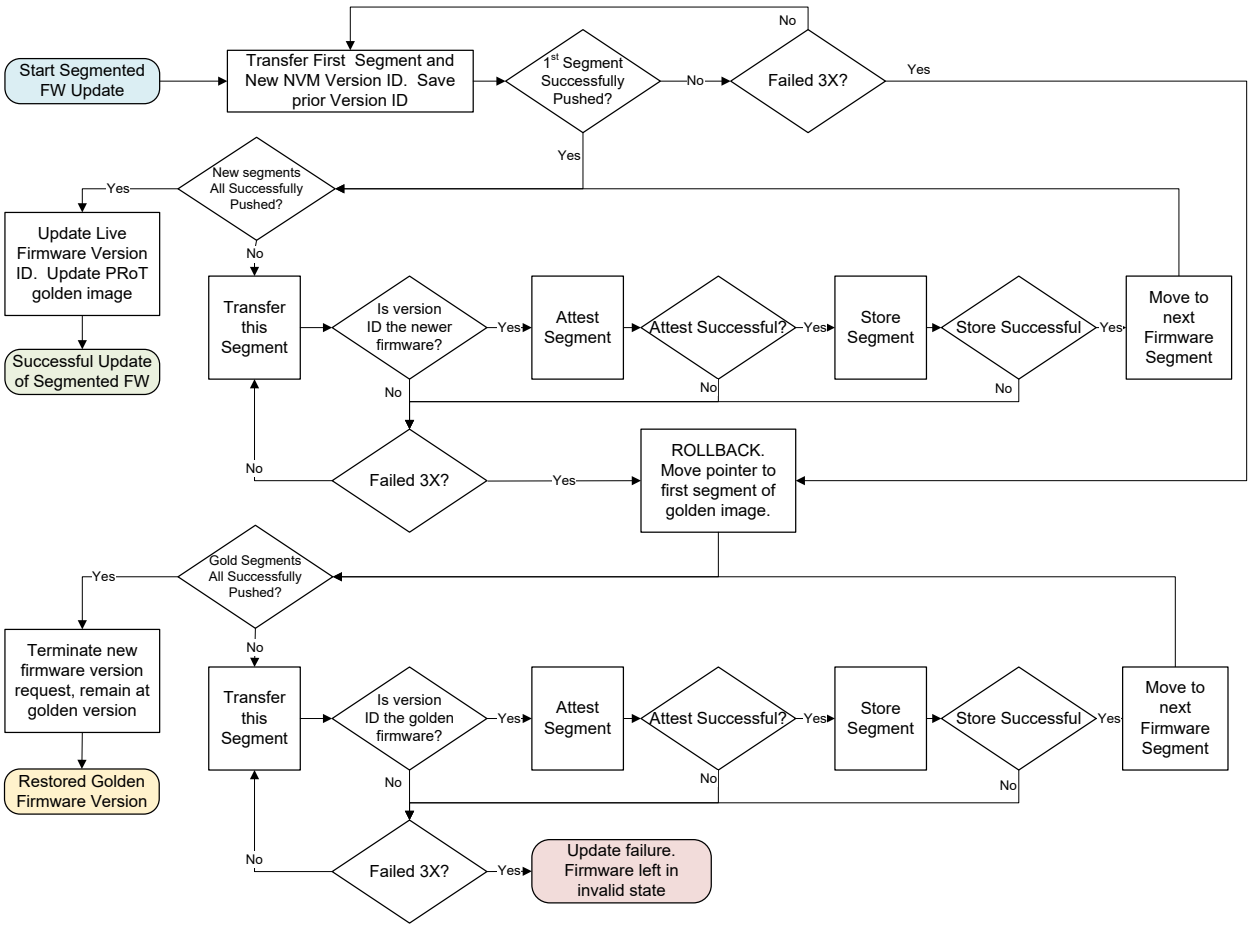


Figure 15. Example Multi-Segment Firmware Update Process

12.2 PMBus Action Completion Alert Generation

12.2.1 PMBus Polling Versus Interrupt Completion

The following optional command code is used to turn on or off SMBus status bit assertion at completion of a security action operation. When unmasked, assertion of this bit will cause an SMBALERT that can tell the PRoT the security action has completed. The PMBus status bit used for reporting security action request completion is STATUS_OTHER Bit [7].

Setting or clearing of this functionality does not alter the behavior of the SMBALERT_MASK bit values, which must be changed via standard PMBus mechanisms. The bit settings and sequencing for this command is shown in Table 42 and executed per the PRoT Attested command sequence of Section 9.

Table 43 describes a register that contains a bit to indicate if the SMBALERT functionality is supported by this device in response to security action status request completion. When SMBALERT is not supported, then polling must be utilized to determine when a request has been completed.

After starting the security action request, the target updates the security action status register in primary memory address 15h per the description in Section 7.6.

Table 42. Firmware PMBus Polling Control Command Sequence

Step from Figure 11	Memory Location	Data
Not Shown	Select Desired SMBALERT behavior and lock	Select the desired SMBALERT generation behavior and whether or not that behavior is either locked until a power cycle or locked permanently.
R1-R8	Perform PRoT attestation calculations as required per Section 9.	Request a random nonce and a PRoT attestation algorithm from the target using the process described in Section 9.2. Read the nonce from buffer region [1]. Calculate the expected MAC per Table 30. This expected MAC will be pushed to the target later in this process to help attest that the PRoT is the issuer of the anytime power-on-reset request.
R9	Write Primary Address 19h “security action request”	8Bh: Update alert request mechanism, start command sequence
R10	Primary Address 17h “security action details [0]”	The value in this field will determine if this and following security action requests will trigger the STATUS_OTHER bit [7] to assert when the request has been completed. Value and meaning: • 00h (Default): Security action request completion will not assert STATUS_OTHER bit [7]. The bit is cleared when this is set. • 01h: Security action request completion will assert STATUS_OTHER bit [7]. New security action request will cause this bit to clear upon submission and set when request is completed to either failure or successful outcome. STATUS_OTHER bit [7] will not assert upon completion of any power-on-reset request. For both anytime and output-off power-on-reset requests, poll the security action status for “Idle” indication. • 02h: Same action as 00h but locked so its action can never be enabled for the remainder of the controller power-on-reset cycle. • 03h: Same action as 01h but locked so the action can never be disabled for the remainder of the controller power-on-reset cycle. Else: Reserved.
R11	Write Primary Address 18h “security action details [1]”	00h. Reserved
-	Buffer Region [2]	Do not write. Not needed for this action request.

Step from Figure 11	Memory Location	Data
-	Buffer Region [1]	Do not write. This contains the PRoT requested nonce
R13	Buffer Region [0]	Expected MAC: Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed.
R15	Primary Address 19h "security action request" at completion of data transfer	0Bh: Update alert request mechanism command. Complete data loading, request command execution.

12.3 NVM Fuel Gauge And SMBALERT Support

PMBus secure device primary register 06h provides two key functions. The first of those function is to provide a "gauge" on the amount of NVM writes remaining. The second of those indicates if this device can be configured to trigger SMBALERT at the conclusion of a requested security action.

12.3.1 NVM Fuel Gauges

There are two separate non-volatile storage remaining gauges. The first gauge, located in bits [2:0] indicates the fraction of NVM firmware or configuration space available. It is on a scale of 7 down to 0 with 0 indicating no more writes to NVM are possible. The exact meaning of the settings 7 down to 1 are not dictated by this specification but should be some measure as to the fraction of the possible updates remaining.

The second gauge indicates the remaining number of times the PSK can be iterated. It is in bits [5:3]. Similar to the first gauge, it is on a scale of 7 down to 0 with 0 indicating no more writes to NVM are possible. The exact meaning of the settings 7 down to 1 are not dictated by this specification but should be some measure as to the fraction of the possible iterations remaining.

Table 43. NVM Fuel Gauge And Interrupt Completion Register

PMBus Security Memory Map Register And Address	Bit	Description
Primary Address 06h "Non-volatile and Alert Capability"	[7] (MSB)	If 1, target is asserting SMBALERT for supported PMBus security action request completions. If 0, target is not asserting SMBALERT to indicate action request completion. Status must be polled.
	[6]	Reserved, set to 0

PMBus Security Memory Map Register And Address	Bit	Description
	[5:3]	PSK iterations remaining gauge: • 0: PSK locked or all PSK iterations are used • 1-7: Metric with 1 being NVM almost exhausted and 7 indicating many future updates are possible.
	[2:0]	Authenticated Updates to NVM (Firmware and Cfg) available • 0: No more free space for authenticated updates • 1-7: Metric with 1 being NVM almost exhausted and 7 indicating many firmware or configuration updates are possible.

12.4 PMBus Host Attested PAGE_PLUS PMBus commands

The following syntax is used to define a format for submitting a “general” PMBus PAGE_PLUS command that is run if and only if the target can attest the PMBus host successfully. This command, along with a separate access control bit [5] permitting commands when the host is attested, may help secure usage of commands such as VOUT_COMMAND without opening the command to any potential aggressor on the bus. This command utilizes the PMBus host attestation flow of Section 9. Buffer region 1 is utilized to send the nonce from the target to the host, buffer region 0 contains the expected MAC. Buffer region 2 holds the details of the command to be sent. If the page-plus command is a read, upon successful completion buffer region 2 will contain the data that would have been provided during the read portion of the transaction.

This command uploaded to buffer region 2 will run only if the command if the target is able to attest the PRoT, otherwise security action status will return an error in the security action status register per Section 7.6. Read commands return data include a PEC to help ensure integrity of the returned data. The buffer region 2 result can be read back using traditional buffer region access commands as described in Section 6.3. Based on security action details settings configured as described in Section 12.4.2, buffer regions [1] and [0] may contain a new nonce and MAC result following a read to protect the data being returned to the host.

12.4.1 PAGE_PLUS_WRITE With Attestation of PMBus Host

The sequence of commands in Table 44 used to request a PMBus page-plus write transaction that is executed only if the data transferred by the command can be attested by the host. This is intended to be used for infrequent commands since its execution time for doing attestation will be much slower than a simple unattested PMBus command. It allows a means to ensure a request is arriving from a host that knows the PSK shared with the target, instead of being sent by some attacker also on the bus. In this scenario, attesting with the hash of the PMBus command provides stronger protection against a sophisticated and malicious attack than the PEC.

Table 44. PMBus PAGE_PLUS_WRITE Command Executed After Attestation Of Host

Step From Figure 11	Action	Description
Not shown	Select Write Command	The write command payload and payloads are used in calculation of the expected MAC after retrieving a nonce in the next step.
R1-R8	Request nonce from the PMBus target	Request a random nonce and a PRoT attestation algorithm from the target using the process described in Section 9.2. Read the nonce from buffer region [1]. Using the pair data, calculate the expected MAC per Table 30. This expected MAC will be pushed to the target later in this process to help attest that the PRoT is the issuer of the attested PAGE_PLUS command.
R9	Write Primary Address 19h "security action request"	8Dh: PMBus PAGE_PLUS command with attestation of host, start configuring the command contents and attestation data for target
R10	Write Primary Address 17h "security action details [0]"	This field determines the write-block size of the PMBus PAGE_PLUS_WRITE command per PMBus specification. For these commands, the value of "security action details [1]" must be 00h to ensure no read is made. Value: Meaning • 3: PAGE_PLUS_WRITE Byte [Page, Command, Data Byte] • 4: PAGE_PLUS_WRITE Word [Page, Command, 2 Data Bytes] • *N*+2: PAGE_PLUS_WRITE Block of size *N* Bytes [Page, Cmd, Bytes]
R11	Write Primary Address 18h "security action details [1]"	This data indicates how to handle the returned data. In this example, there is no read data to return. 00h: PAGE_PLUS_WRITE – no read transaction component
R12	Buffer Region [2]	Write the data for the PMBus PAGE_PLUS_WRITE per Table 45
	Buffer Region [1]	Do not write. This contains the PRoT requested nonce

Step From Figure 11	Action	Description
R13	Buffer Region [0]	Expected MAC: Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.
R15	Primary Address 19h “security action request” at completion of data transfer	0Dh: PMBus PAGE_PLUS command with attestation of host, request execution

Table 45. PMBus PAGE_PLUS_WRITE Command Request With Host Attestation By Type

Byte in Region 2	Write Command Type				
	Write Byte	Write Word	Write 32	Write 64	Write Block
Region 2 Byte [0]	Set to 00h (Length is known from the command code itself)				Block Size (N)
Region 2 Byte [1]	Page	Page	Page	Page	Page
Region 2 Byte [2]	Cmd Code	Cmd Code	Cmd Code	Cmd Code	Cmd Code
Region 2 Byte [3]	Data Byte	Data Byte L	Data bits [7:0]	Data bits [7:0]	Data Byte 1
Region 2 Byte [4]	-	Data Byte H	Data bits [15:8]	Data bits [15:8]	Data Byte 2
Region 2 Byte [5]	-	-	Data bits [23:16]	Data bits [23:16]	Data Byte 3
Region 2 Byte [6]	-	-	Data bits [31:24]	Data bits [31:24]	Data Byte 4
Region 2 Byte [7]	-	-	-	Data bits [39:32]	Data Byte 5

Byte in Region 2	Write Command Type				
	Write Byte	Write Word	Write 32	Write 64	Write Block
Region 2 Byte [8]	-	-	-	Data bits [47:40]	Data Byte 6
Region 2 Byte [9]	-	-	-	Data bits [55:48]	Data Byte 7
Region 2 Byte [10]	-	-	-	Data bits [63:56]	Data Byte 8
⋮	-	-	-	-	⋮
Region 2 Byte [N]	-	-	-	-	Data Byte N-1
Region 2 Byte [N+1]	-	-	-	-	Data Byte N

The security action status request is calculated per Table 30 based upon if this command was able to be issued to the target device; specifically, whether the PRoT was successfully attested allowing the command to be issued. Any error handling from the command itself being invalid are handled similarly to if the un-attested equivalent PAGE_PLUS_WRITE command was issued to the device. Errors of this kind are typically reported through the STATUS_CML command.

12.4.2 PAGE_PLUS_READ And Process Calls With Attestation of Host

The sequence of commands in Table 46 is used to request a PAGE_PLUS_READ with or without a process call component while attesting the host and ensuring data integrity of the request. After the command has completed, the returned data overwrites the data that was previously in buffer region 2 to configure the read. Any errors in the PRoT attestation process will be reported via the “security action status” register per section

To protect the data read back via buffer region 2 from being altered in transmission, the MAC in buffer region 0 is also updated using a hash of the resulting read data. There is a hash that must be attested by the target during write, and a similar hash returned during read.

If the requested command data passes the host-attestation, then the command will execute and return the data read into buffer region [2]. If “security action details [1]” register content is 02h, then to protect the integrity of the resulting read data via buffer region [2], a nonce is generated and stored in buffer region [1], a new expected MAC is calculated using the returned data from buffer region [2] as the data to be hashed per Table 30. That new MAC is stored in buffer region [0]. If any buffer region is written following successful execution of the security action, then the “security action status” register will change from 01h indicating success to 00h indicating idle. A read

of this register after completely reading the buffer regions shall ensure the data was unaltered from what was returned by the commands.

Table 46. PAGE_PLUS_READ Security Action Request Flow

Step From Figure 11	Memory Location	Data
R1-R8	Request nonce from the PMBus target	Nonce is generated and made available in buffer region [1] for the PRoT to download and use in its attestation calculations.
R9	Write Primary Address 19h “security action request”	8Dh: Start "Secured (Host Attested) PAGE_PLUS_READ" request sequence
R10	Primary Address 17h “security action details [0]”	This byte contains the number of data bytes in the write portion of the PMBus PAGE_PLUS_READ command outlined in PMBus specification Part II [A02]. This will be 2 data bytes (the page and command code, for simple reads), this will increase with the block size to be transmitted for process calls. Value and meaning: 2: PAGE_PLUS_READ without any process call component. Only write data items are [Page, Command Code]. PAGE_PLUS_READ Byte PAGE_PLUS_READ Word PAGE_PLUS_READ 32 PAGE_PLUS_READ 64 PAGE_PLUS_READ Block 4: PAGE_PLUS variant of Process Call PAGE_PLUS Process Call (Write word, Read Word) *N*+2: PAGE_PLUS variant of BLOCK WRITE-BLOCK READ PROCESS CALL write-block portion of block size of *N*.

Step From Figure 11	Memory Location	Data
R11	Write Primary Address 18h “security action details [1]”	01h: PAGE_PLUS_READ including • PAGE_PLUS_READ Byte • PAGE_PLUS_READ Word • PAGE_PLUS_READ 32 • PAGE_PLUS_READ 64 • PAGE_PLUS_READ Block • PAGE_PLUS Process Call • PAGE_PLUS BLOCK WRITE-BLOCK READ PROCESS CALL. Returned read data is stored in buffer region [2] No new hash is calculated for the return data. 02h: Same as 01h (PAGE_PLUS_READ), but additionally: • Generates a new nonce in buffer region [1] using the same attestation algorithm as used to generate the first nonce. • Target calculates an expected MAC based on return output data from buffer region [2], and a new target generated nonce (stored in buffer region [1 • Expected MAC is stored in buffer region [0].
R12	Write Buffer Region [2]	This buffer is loaded per the setup of Table 47. In the case of simple read transactions, it contains only the page and command. For process call, it also contains the data word. For block-read block-write process calls it contains the data to write. After execution of the command, this data is overwritten with the data read back from the resulting read portion of the PAGE_PLUS command.
-	Skip Buffer Region [1]	Do not write. This contains the PRoT requested nonce.
R13	Write Buffer Region [0]	Expected MAC: Write the MAC result that target should match when replicating the calculation using target-determined nonce and secured PSK. If matching fails, the command will not be executed. Byte ordering within the memory region is to be given in the device literature.

Step From Figure 11	Memory Location	Data
R15	Write Primary Address 19h “security action request”	0Dh: PMBus PAGE_PLUS command with attestation of host, request execution.
R19	Read Primary Address 15h “security action status”	Ensure request completed successfully. Register 15h from the target contains data per Section 7.6.
R20	Read output data from buffer regions	If previous read of security action status was 01h indicating success, then: Buffer region [2] contains the read results per Table 48 If security action details [1] was 02h to indicate stronger protection of read back data, then additionally: • Buffer region [1] contains the new nonce • Buffer region [0] contains a MAC for the read data. • Read the data back using the buffer region access commands of Section 6.3.
R21	Read Primary Address 15h “security action status”	If the previous read of security action status was 01h indicating success, then re-read it here and ensure that it remained 01h indicating that the buffer regions were not written to since storing results. Summary of results shown in Section 7.6.

Table 47. Buffer Region [2] Input For PAGE_PLUS_READ Action With Attested PRoT

Byte In Region 2	Read Command Protocol						
	READ BYTE	READ WORD	READ 32	READ 64	BLOCK READ	PROCESS CALL	BLOCK-WRITE, BLOCK READ PROCESS CALL
Region 2 Byte [0]	Page	Page	Page	Page	Page	Page	Page
Region 2 Byte [1]	Cmd	Cmd	Cmd	Cmd	Cmd	Cmd	Cmd

Byte In Region 2	Read Command Protocol						
	READ BYTE	**READ WORD**	**READ 32**	**READ 64**	**BLOCK READ**	**PROCESS CALL**	**BLOCK-WRITE, BLOCK READ PROCESS CALL**
Region 2 Byte [2]	-	-	-	-	-	Write Data Byte L	Write Data Byte 1
Region 2 Byte [3]	-	-	-	-	-	Write Data Byte H	Write Data Byte 2
⋮	-	-	-	-	⋮	-	⋮
Region 2 Byte [*N*]	-	-	-	-	-	-	Write Data Byte *N-1*
Region 2 Byte [*N+1*]	-	-	-	-	-	-	Write Data Byte *N*

Table 48. Buffer Region [2] Output For PAGE_PLUS_READ Action With Attested PRoT

Byte in Region 2	Read Command Protocol						
	READ BYTE	**READ WORD**	**READ 32**	**READ 64**	**READ BLOCK**	**PROCESS CALL**	**BLOCK-WRITE, BLOCK READ PROCESS CALL**
Region 2 Byte [0]	2	3	5	7	*M+1*	3	*M+1*
Region 2 Byte [1]	Data Byte	Data Byte L	Data Bits [7:0]	Data Bits [7:0]	Data Byte 1	Read Data Byte L	Read Data Byte 1
Region 2 Byte [2]	PEC	Data Byte H	Data Bits [15:8]	Data Bits [15:8]	Data Byte 2	Read Data Byte H	Read Data Byte 2
Region 2 Byte [3]	-	PEC	Data Bits [23:16]	Data Bits [23:16]	Data Byte 3	-	Read Data Byte 3
Region 2 Byte [4]	-	-	Data Bits [31:24]	Data Bits [31:24]	Data Byte 4	-	Read Data Byte 4

Byte in Region 2	Read Command Protocol						
	READ BYTE	**READ WORD**	**READ 32**	**READ 64**	**READ BLOCK**	**PROCESS CALL**	**BLOCK-WRITE, BLOCK READ PROCESS CALL**
Region 2 Byte [5]	-	-	PEC	Data Bits [39:32]	Data Byte 5	-	Read Data Byte 5
Region 2 Byte [6]	-	-	-	Data Bits [47:40]	Data Byte 6	-	Read Data Byte 6
Region 2 Byte [7]	-	-	-	Data Bits [55:48]	Data Byte 7	-	Read Data Byte 7
Region 2 Byte [8]	-	-	-	Data Bits [63:56]	Data Byte 8	-	Read Data Byte 8
Region 2 Byte [9]	-	-	-	PEC	Data Byte 9	-	Read Data Byte 9
⋮	-	-	-	-	⋮	-	⋮
Region 2 Byte [*M*-1]	-	-	-	-	Data Byte *M-1*	-	Read Data Byte *M-1*
Region 2 Byte [*M*]	-	-	-	-	Data Byte *M*	-	Read Data Byte *M*
Region 2 Byte [*M*+1]					PEC	-	PEC

13. Mapping Of API To PMBus Standard Hardware Specifications

Table 49 maps API descriptions of this document to their recommended standard hardware definition with full details in the PMBus Part IV specification. This standard hardware is strongly recommended to ease compatibility between suppliers.

Table 49. Mapping API To PMBus Part IV Standard Hardware Implementation

API	App Profile Specification Section [A03]	PMBus 1.5 Part IV Standard Hardware Specification Section [This Document]
PSK Management		
PMBus_ProvisionPSK0	8.2.1	8.1.3

API	App Profile Specification Section [A03]	PMBus 1.5 Part IV Standard Hardware Specification Section [This Document]
PMBus_ReqNewPSK_Algo	8.2.2	8.2.1
PMBus_ReqNewPSK	8.2.3	8.2.2
PMBus_LockPSK	8.2.4	8.2.3, 8.2.4
Attestation		
PMBus_AttestTarget	8.3.1	8.4.1
PMBus_AttestationAlgoSupport	8.3.2	8.4.1.1
PMBus_ReqAttestTarget	8.3.3	8.4.1
PMBus_RetrieveAttestTarget	8.3.4	8.4.1.4
PMBus_HashCalc	8.4.1	8.4.2
PMBus_KDFCalc	8.4.2	8.4.3
PMBus_MACCalc	8.4.3	8.4.4
Firmware Update Commands		
PMBus_NewFwUpdatesRem	8.5.1	12.3.1
PMBus_NewFwCommitSL1	8.5.2	10
PMBus_NewFwCommitSL2	8.5.3	10
PMBus_NewFwCommitSL3	8.5.4	10
PMBus_Reboot	8.5.5	10.4.2
PMBus_NewFwDownload	8.5.6	10.5.1
PMBus_FetchFw	8.5.7	10.5.1
Security Level 3 Commands		
PMBus_RequestNonce	8.6.1	9.2
PMBus_Secure_Access_Control	8.6.2	11.1
PMBus_Secure_PagePlus	8.6.3	12.4
PMBus_Secure_AlertConfig	8.6.4	12.2
PMBus_RebootLockout	8.6.5	10.4.1
PMBus_RebootFromOn	8.6.6	10.4.3
PMBus_FetchFwLockout	8.6.7	10.5.2
Miscellaneous Commands		
PMBus_Profile_SecurityVersion	8.7.1	12.1
PMBus_Device_FwConfigVersion	8.7.2	12.2
PMBus_Device_Profile	8.7.3	N/A

Appendix I. Pseudo-code For Security Action Request Ordering Enforcement

Pseudo-code illustrating the enforcement of the ordering of a Security Action Request is shown here.

```
Wait for byte to be written by either 70h, 71h, or 72h.
dispatchCommand = 0

// Detect the start of a new action request sequence
if (write to primary address 19h "security action request" && data[7]==1) begin
    completed = 0
    dispatchCommand = 0
    opCodeExp   = data[6:0]
    orderOk     = 1
    lastRegion  = 4
    lastAddress = 0
    wrote_actDetails0 = 0
    wrote_actDetails1 = 0
end // security action request -- start action request sequence

// Security action details [0] must after requesting action sequence
if (write to primary address 17h "security action details [0]") begin
    wrote_actDetails0 = 1
    if ((wrote_actDetails1 == 1) or (lastRegion<4))
        orderOk = 0
end

// Security action details [1] must be written after security action details [0]
if (write to primary address 18h "security action details [1]") begin
    wrote_actDetails1 = 1
    if ((wrote_actDetails0 == 0) or (lastRegion<4))
        orderOk = 0
end


// Make sure all writes to the buffer regions go first and in order
// If needed, write buffer region [2] starting from lowest address and monotonically
increasing
// If needed, write buffer region [1] starting from lowest address and monotonically
increasing
// If needed, write buffer region [0] starting from lowest address and monotonically
increasing
if (write through window to buffer region) begin
    address_within_buffer = window_low_byte_index + primary address – 20h
    if ( ( region >  lastRegion                                                  ) or
         ((region == lastRegion) and not (address_within_buffer > lastAddress)) or
         (wrote_actDetails0 ==0                                                  ) or
         (wrote_actDetails1 ==0                                                  ) )
        orderOk = 0
    else begin
        lastRegion  = region
```

```
        lastAddress = address_within_buffer
    end
end

// Check that requested action matches the reaction deemed as imminent
if (write to primary address 19h "security action request" && (data[7]==0)):
    if (completed == 1):
        // There's been no preparation for action request with msb == 1
        dispatchCommand = 0
        completed = 1
    elseif ((orderOk           == 1 ) and
            (wrote_actDetails0 == 1 ) and
            (wrote_actDetails1 == 1 ) and
            (opCodeExp == data[6:0] )) begin
        // Request integrity is good, proceed to action-specific conditions
        dispatchCommand = 1
        completed = 1
    end
    else begin
        // Request Integrity fails, block action execution
        dispatchCommand = 0
        completed = 1
    end
end  // validate that action requested matches expected
```

Appendix II. Summary Of Changes

DISCLAIMER: The section is provided for reference only and for the convenience of the reader. No suggestion, statement or guarantee is made that the description of the changes listed below is sufficient to design a device compliant with this document.

Note: As thus is the initial release of PMBus Specification Part IV, there are no changes from a previous version.

This document is released as Revision 1.5 to stay in synch with other parts of the PMBus specification even though there are no previous releases.