TRUSTED® COMPUTING GROUP

SPECIFICATION

Measurement and Attestation RootS (MARS) Library Specification

Version 1
Revision 4
January 5, 2022

Contact: admin@trustedcomputinggroup.org

PUBLIC REVIEW

Work in Progress

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

DISCLAIMERS, NOTICES, AND LICENSE TERMS

ACKNOWLEDGEMENTS

The TCG would like to gratefully acknowledge the contributions of the following individuals and companies who volunteered their time and efforts for the development of this specification.

Tom Broström, Cyber Pack Ventures, Inc.

Brian Dziki, United States Government

Scott Ludwin, Toyota Motor North America

Steve Luther, United States Government

Graeme Proudler, Invited Expert

Vadim Sukhomlinov, Google Inc.

Dick Wilkins, Phoenix Technologies, Ltd.

CONTENTS

TABLES

1 SCOPE

This document is the Measurement and Attestation RootS (MARS) Library Specification. It describes hardware logic that enables devices (e.g., microcontrollers) to provide the functionality described in *MARS Use Cases and Considerations* (TCG, 2021). The primary use case is measurement recording and attesting in a manner inspired by the Trusted Platform Architecture defined in the TPM Library specification (TCG, 2019).

Platform-specific profiles of this specification define options, settings and any additional commands necessary to produce a functional and compliant device.

This specification does not place specific requirements on command, control and transport protocols between driver and device, as the device implementation may be deeply embedded and proprietary.

1.1 Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document form normative statements and are to be interpreted as described in RFC-2119, *Key words for use in RFCs to Indicate Requirement Levels*.

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, it can be considered a kind of normative statements.

EXAMPLE: Start of informative comment

This is the first paragraph of 1–n paragraphs containing text of the kind *informative comment* ...

This is the second paragraph of text of the kind *informative comment* ...

This is the nth paragraph of text of the kind *informative comment* ...

To understand the TCG specification the user must read the specification. (This use of MUST does not require any action).

End of informative comment

2 Abbreviations, Acronyms and Terms Used

AK	Attestation Key
API	Application Programming Interface
DP	Derivation Parent
HMAC	(keyed) Hash-based Message Authentication Code
KDF	Key Derivation Function
MAC	Message Authentication Code
MARS	Measurement and Attestation RootS
PCR	Platform Configuration Register
PS	Primary Seed
RTM	Root of Trust for Measurement
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SHA	Secure Hash Algorithm
TPM	Trusted Platform Module
TSR	Trusted Sensor Register

3 Conventions and Data Types

3.1 Naming Conventions

Table 1 – Conventions

CONVENTION	EXAMPLE
All public names in the API are prefixed with “MARS_”.	See next rows

All function names have the form of
MARS_Verb()
or
MARS_NounVerb()
	
	MARS_Sign()
	
	MARS_SignatureVerify()

Nouns that are acronyms (e.g., PCR) are spelled as words.	MARS_PcrExtend()
All other names are in upper case.	MARS_RC_SUCCESS

3.2 Data Types

This specification uses primitive data types defined in ISO/IEC C18 (ISO/IEC, 2018). The following general rules apply. Exceptions are defined where needed.

- All integer data types are uint32_t.
- All functions return a response code of type uint32_t.
- All data lengths are bytes represented by type size_t.
- Pointers must reference memory that is allocated and aligned.

3.3 Symbols

A || B	concatenation of B to A
REG#	contents of selectable register number #

4 Trusted Platform Architecture

MARS closely follows the Trusted Platform concepts detailed in the TPM Architecture specification (TCG, 2019), and as described below. Section 4 is informative.

4.1 Events

A code or data module that is about to be executed or processed for the first time is considered an *event*. An event may be conceptualized as a link in a transitive trust chain. Each event is represented by a digest/measurement produced by hashing its module's contents. Since digests are statistically unique, the digest identifies the event's module. A sequence of events may be recorded by the host in an event log.

4.2 Root of Trust for Measurement

A Trusted Platform is booted by an RTM whose actions include:

1. Locate and load a module external to the RTM,
2. Measure the module,
3. Deliver the measurement to the RTS (e.g., MARS),
4. Optionally populate an event log, and
5. Execute or process the module

Each subsequent module in an event sequence is responsible for the same five actions. Note that the RTM is not part of MARS. The RTM and MARS reside on the same device and work together to implement a Trusted Platform.

4.3 Root of Trust for Storage

Just as a module is identified by its digest, so too is the event log. Instead of recording the entire event log's digests, the RTS assists in cryptographically building a cumulative digest as the events transpire. Refer to the MARS_PcrExtend() (section 8.3.1) operation for more detail. The RTS maintains this cumulative digest in its Platform Configuration Register (PCR). Since the events are also digests themselves, the PCR is said to contain a digest of digests.

4.4 Root of Trust for Reporting

To convey the history of the transitive trust chain, the RTR is used to digitally sign the PCR(s). This signature is used by the host device to form an attestation about the host for a remote challenger. The challenger can use this attestation to:

1. Verify the Endorser of the device's identity,
2. Verify the device's identity,
3. Verify the PCR's authenticity,
4. Assess the event log's identity,
5. Verify the event log's integrity, and
6. Assess the events in the event log

The RTR can convey identity either asymmetrically or symmetrically.

- With asymmetric cryptography, the RTR uses an Attestation Key (AK) certified by an Endorser known as the Attestation Certificate Authority. The public portion of an AK can be used to verify RTR signatures produced with its private AK. A challenger that already trusts the Endorser can directly verify the device identity.

- With symmetric cryptography, an AK is shared between the RTR and Endorser. A challenger wishing to verify a device’s identity must trust and contact the Endorser. This contact between challenger and Endorser can be negotiated asymmetrically. The Endorser can then retrieve the shared AK associated with the claimed device identity to verify the identity and attestation signature on behalf of a challenger. For example, the value of the identity may be from a sequence (e.g., serial number) or derived from the Derivation Parent (see sections 5.2.2 and 8.4.1).

5 MARS Device Requirements

5.1 Cryptography

To implement the RTS, a hash function is required to extend a PCR. The RTR also requires a hash function to produce a digest of the information to be attested and a digital signing mechanism to sign this digest. If the AK is generated on-demand, then a KDF mechanism is required. If MARS_Derive() is implemented, a suitable Deterministic Random Bit Generator capability is needed.

Start of informative comment

MARS may exclusively use symmetric cryptography to produce a signature (e.g., MAC or AEAD tag) and need not implement an asymmetric algorithm. This stands in contrast to the TPM which requires at least one asymmetric algorithm.

End of informative comment

In keeping with MARS' minimalist approach, a single core algorithm SHOULD be implemented to support the three required primitives – hash, sign, and KDF. Furthermore, it is desirable to retain compatibility with the TPM so that the TPM can verify MARS signatures. With this first release of this MARS specification, the only algorithms that qualify with a single core algorithm for hash, sign and KDF are SHA-256, HMAC-SHA256 and NIST SP800-108. They appear in the TCG Algorithm Registry (TCG) with identifiers TPM_ALG_SHA256, TPM_ALG_HMAC and TPM_ALG_KDF1_SP800_108 respectively. Additional algorithm identifiers may be requested from the TCG. Profiles of this specification MUST implement appropriate combinations of TCG registered algorithms to support the cryptographic needs of the support functions defined in section 5.4.

5.2 Device State

5.2.1 Primary Seed (PS)

A MARS device SHALL contain a persistent Primary Seed that is the most critical security parameter in MARS' architecture and provides the identity of the device. The PS is the root of the MARS key hierarchy, which can, in part, be used to derive device identities. The PS MUST be in a form appropriate for the implemented KDF to derive the initial Derivation Parent and SHOULD have at least the highest level of protection required for all PS uses.

Establishment of the PS, its type of non-volatile memory, and lifecycle management are beyond the scope of this specification.

5.2.2 Derivation Parent (DP)

The Derivation Parent is first derived from the PS on device power-up or reset, using a platform profile specific procedure. The DP is volatile, and is used to derive an Attestation Key, other derived values, or the next DP.

Start of informative comment

The term “volatile” is used to express the fact that there is no requirement to retain a value after a power cycle. However, this document does not specify which type of memory to utilize for the DP.

End of informative comment

5.2.3 Selectable Registers

MARS supports two types of volatile registers that can be selected for use in a variety of API functions. MARS has PCR, and TSR (Trusted Sensor Registers). A MARS Profile MUST implement at least one PCR. TSR are optional. The maximum number of PCR plus TSR registers allowed is 32. The quantity of PCR and TSR registers in an implementation can be retrieved via MARS_CapabilityGet() with the MARS_PT_PCR and MARS_PT_TSR tags, respectively. The length of all selectable registers is the size of a digest produced by the implemented hash algorithm (see MARS_PT_LEN_DIGEST). PCR and TSR registers MUST only be modifiable by MARS and in the following ways – via initialization (5.2.4), or extend (for PCR), or sampling (for TSR).

5.2.3.1 Selection

When choosing a group of registers to use in certain MARS functions, a uint32_t bitmask parameter named **regSelect** is used. In an implementation with m PCR and n TSR registers, bits 0 through m-1 of regSelect represent the PCRs, and bits m through m+n-1 represent the TSRs. Bits m+n through 31 MUST be zero.

5.2.3.2 PCR

A PCR (see section 4.3) is initialized to zero and MUST only be updated via MARS_PcrExtend() (8.3.1). A device MUST have at least one PCR, known as PCR 0. MARS MAY provide additional PCRs, typically to record some subset of events. When a PCR is updated with events in an event log (section 4.1), that PCR’s value can be used as an integrity check of the corresponding events.

MARS commands that use PCR values can be directed to use a specific subset of PCRs (including only one or none) so that the commands’ results will depend on certain device events.

5.2.3.3 TSR – Trusted Sensor Register

A device profile specification utilizing MARS MAY link an onboard sensor (or clock, etc.) to a TSR. TSR are not extendable. Instead, they are implicitly written from a sampled linked sensor whenever they are used in a regSelect. The sampling is performed by logic supplemental to MARS (refer to CryptSnapshot(), see section 5.4.7). TSR registers retain their sampled values until modified by a subsequent use of regSelect. MARS commands that use regSelect do not return the values of the selected registers. To obtain the registers’ values, MARS_RegRead() (8.3.2) MUST be used *after* regSelect is processed.

Start of informative comment

The anticipated use of TSR is to sign sensor values via MARS’ quoting ability. For example, suppose a device is constructed with MARS having four PCR and two TSR linked to an onboard clock and pressure sensor. To attest to both the sensor's reading and the time of the reading, the following events would occur:

- A challenge nonce is received.
- A command to quote registers 5 and 6 is issued (TSR registers 0 and 1).
- The MARS quoting function uses CryptSnapshot() to create a signable digest or "snapshot".
    - CryptSnapshot(), in the process of gathering the selected registers, triggers supplemental hardware to sample linked sensors.
    - The sampled values are written to the selected TSR registers.
    - The regSelect, register values, and nonce are hashed to produce a snapshot.
- The snapshot is signed, and its signature is returned.
- A command to read register 5 is issued.
- A command to read register 6 is issued.

The signature and contents of the registers can be sent to and verified by the challenger.

Note that reading a TSR does not trigger its update.

End of informative comment

5.2.4 Initialization

MARS' Roots of Trust MUST only be reset concurrently with a reset of its host and its host's RTM (see section 4.2). MARS is reset via the _MARS_Init signal.

When _MARS_Init is issued, MARS performs the following:

- Initialize PCRs to zero.
- Initialize TSR to Profile-specified values.
- Reset failure mode to False.
- Perform a self-test as per MARS_SelfTest(), if implemented.
- Derive a volatile DP from a non-volatile PS per section 5.2.2

After MARS successfully completes its reset, MARS MUST be ready to process commands from the RTM and other host software.

5.3 Key Hierarchy

All secrets used by MARS for generating other secrets, keys, signatures, and values to support the Use Cases belong to a single hierarchy rooted in the PS. The only immediate child of the PS is the DP. The DP is a volatile secret used as the source secret for deriving keys, and for deriving and overwriting the next generation DP. Refer to Figure 1 - Key Hierarchy.

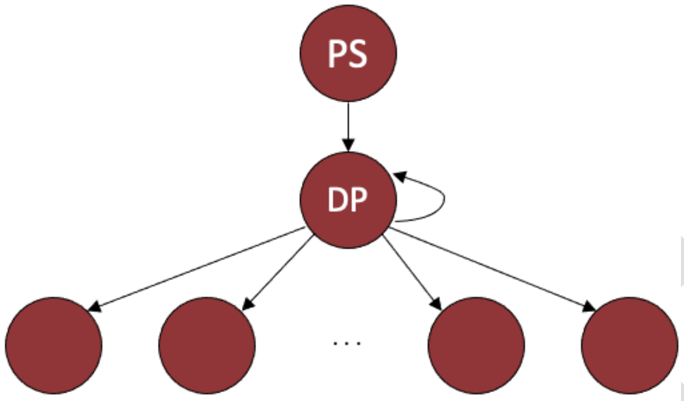

Figure 1 - Key Hierarchy

Keys and values derived from the DP are shown as leaf nodes, which may be created for signing (unrestricted keys), attestation (restricted keys), verifying or as derived bits for external use. Leaf nodes are created on demand and not retained in MARS dedicated registers. Keys are created when needed, using the key derivation function as with:

```
key = kdf(DP, label, context)
```

where the MARS-supplied label designates the purpose of the key, and the host-supplied context is used to differentiate keys used for the same purpose (e.g., multiple attestation keys). The label guarantees that keys used for different purposes will be unique. For example, it will not be possible for the user to create an unrestricted signing key that is the same as a restricted attestation key. All values are derived deterministically given the same inputs.

The label takes on the values defined in Table 2, zero padded as required by the implemented KDF.

Table 2 – Cryptographic Key Label Prefixes

NAME	VALUE	DESCRIPTION
MARS_KX	‘X’	eXternal
MARS_KD	‘D’	Derivation Parent
MARS_KU	‘U’	Unrestricted signing
MARS_KR	‘R’	Restricted attestation

MARS devices MAY support development or debug mode in addition to regular or production mode to ease development. If the MARS device is in debug mode, then the KDF MUST XOR the label prefix with 0x80 prior to its use. This enables derivation of different values depending on whether the device is in or out of debug mode.

5.4 Support Functions

The following support functions or equivalent functionality MUST be implemented within MARS, and inaccessible elsewhere. These functions will be referenced when defining the behavior of MARS commands but they are not part of the API.

5.4.1 CryptHash(data)

Computes a one-way cryptographic hash over the supplied data using the Profile-specified hash algorithm. The resulting digest is returned.

5.4.2 CryptSign(key, digest)

Produces a digital signature of the digest using the Profile-specified algorithm (e.g., MAC, Digital Signature Algorithm) and the provided key. The size of the resulting signature can be retrieved via MARS_CapabilityGet() with the MARS_PT_LEN_SIGN tag.

5.4.3 CryptVerify(key, digest, signature)

Returns a Boolean result to indicate whether the signature of the digest has been verified using the Profile-specified signature verification algorithm and the provided key.

5.4.4 CryptSkdf(parent, label, context)

Derives a symmetric key using the Profile-specified symmetric KDF from the specified parent secret, label and API-provided context. This function is used when establishing the DP from the PS (section 5.2.2), when extending the DP (section 8.4.2), when creating the AK (if symmetric, see comment below) from the DP, or when deriving bytes for external use (section 8.4.1). Refer to section 5.3 for a description of the label parameters.

5.4.5 CryptAkdf(parent, label, context)

Derives an asymmetric key pair using the Profile-specified asymmetric KDF from the provided parent secret, label and API-provided context. Refer to section 5.3 for a description of the label parameters.

5.4.6 CryptXkdf

CryptXkdf is CryptAkdf if CryptAkdf is implemented. Otherwise, CryptXkdf is CryptSkdf.

5.4.7 CryptSnapshot(regSelect, context)

A “snapshot” is a digest created by MARS as input for quoting or deriving other values. The snapshot MUST be computed as defined here, using:

- regSelect – a 32-bit bitmask indicating the register indices whose contents will be used
- register values – contents of selected PCR and/or TSR
- context – caller provided data

During the execution of CryptSnapshot(), the TSR registers identified in regSelect are written, as specified by the applicable MARS Profile.

The snapshot is then computed by

```
    snapshot = CryptHash (regSelect || REG# || … || REG# || context)
```

with the selected registers being concatenated in ascending order of their indices.

Start of informative comment

For example, in an implementation with three PCR, a call to

```
    CryptSnapshot (0b101 || nonce)
```

using 32-byte digests and nonce would result in 4 + 3 * 32 = 100 bytes hashed with

```
    CryptHash (0 || 0 || 0 || 5 || PCR0 || PCR2 || nonce)
```

where "0 || 0 || 0 || 5" is the four-byte, big endian representation of regSelect 0b101.

End of informative comment

5.5 Session Management

MARS MUST maintain context for a single series of commands (session) only. There is no mechanism to save and restore context. The operating system should prevent interleaving of multiple sessions amongst processes (e.g., via exclusive device access). MARS_Lock() (section 8.1.2) and MARS_Unlock() (section 8.1.3) MUST be used around a set of MARS commands for analogous prevention amongst threads. Attempts to use the API when MARS is not locked SHALL return MARS_RC_ACCESS.

5.6 Protected Capabilities and Locations

MARS' Roots of Trust maintain sensitive values and capabilities that require protections commensurate with the security needs of the manufactured device. While all MARS' resources require integrity protection against arbitrary alteration (e.g., of the PCR, TSR, signing algorithm, or _MARS_Init signal), some require confidentiality protection against disclosure. Volatile secrets (e.g., DP, AK) MUST NOT be readable at run time. When MARS is powered off, profile-specific protection is anticipated for data-at-rest – specifically, the PS.

Though the design of a protected capability may not be sensitive, its operation may be. MARS SHOULD provide protection against leakage of sensitive information from the operation of a sensitive capability. For example, the signing mechanism in MARS should resist leakage of key or plaintext through side channel analysis or other observable means.

6 Constants

6.1 Response Codes

MARS functions MUST return Response Codes defined in Table 3, and as documented for each function. Other values of response code are reserved for future use by the TCG.

Table 3 – Definition of Response Code Constants

Name	Value	Description
MARS_RC_SUCCESS	0	Command executed as expected
MARS_RC_FAILURE	1	MARS_SelfTest() placed MARS in failure mode or MARS is otherwise inaccessible
MARS_RC_ACCESS	2	MARS is not locked
MARS_RC_SIZE	3	Invalid buffer pointer parameter (null or misaligned) or length parameter invalid for specified operation
MARS_RC_COMMAND	4	Command not supported
MARS_RC_VALUE	5	Value out of range or incorrect for context
MARS_RC_REG	6	Invalid register index specified
MARS_RC_SEQ	7	Not preceded by Sequence start command

7 Compliance

Table 4 identifies which features defined in the API (section 8) are Mandatory, Recommended, or Optional. Mandatory commands are essential to support basic measurement and attestation and MUST be implemented. Recommended commands fulfill most other Use Cases (TCG, 2021) and MAY be implemented. Optional features support convenience functions, or commands that add complexity beyond what would otherwise be recommended, and MAY be implemented. A MARS Profile specification defines inclusion or exclusion of specific features.

Table 4 – MARS Compliance Features

Feature	M / R / O	Comment
MARS_SelfTest	R	
MARS_Lock	M	
MARS_Unlock	M	
MARS_CapabilityGet	M	
MARS_SequenceHash	R	
MARS_SequenceUpdate	R	
MARS_SequenceComplete	R	
MARS_PcrExtend	M	
MARS_RegRead	M	
MARS_Derive	R	
MARS_DpExtend	O	
MARS_PublicRead	M	Only needed if asymmetric AK is supported
MARS_Quote	M	
MARS_Sign	R	
MARS_SignatureVerify	R	
ctxiskey	O	If a MARS does not support ctxiskey functionality and the ctxiskey parameter in a function is set to TRUE, the API MUST return MARS_RC_VALUE.

8 API

Functions within the MARS Application Programming Interface are defined below with a behavioral description, C function prototype, parameter description, returned Response Codes, and often C-like pseudocode. Though the pseudocode is somewhat abbreviated (e.g., excluding error checking, length and response code usage), its behavior, together with the C function prototype, parameters descriptions and response codes are normative.

8.1 Management

8.1.1 MARS_SelfTest

Compliance to standards for hardware security modules may require certain aspects of MARS be tested prior to their use. The features to be tested depend on the implementation of MARS, what security level is desired and direction from the pertinent Profile specification. If a Profile requires MARS_SelfTest(), then MARS output SHALL be disabled until all the tests have passed and SHALL remain disabled when a test fails. A non-destructive self-test can be triggered by a system-wide reset (see section 8.1), or on demand by the host invoking MARS_SelfTest(). Any ongoing sequenced command SHALL be cancelled, and any remaining sequence commands SHALL return MARS_RC_SEQ. If a self-test error occurs, MARS enters failure mode where all MARS commands SHALL return MARS_RC_FAILURE.

8.1.1.1 Prototype

MARS_RC MARS_SelfTest ();

8.1.1.2 Response Codes

- MARS_RC_SUCCESS – all tests executed and passed
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_FAILURE – one or more tests failed; MARS entered failure mode
- MARS_RC_COMMAND – command not supported

8.1.2 MARS_Lock

MARS_Lock() prepares MARS for exclusive access to support a series of non-interleaved commands. If already locked by another thread, this function SHALL block until that thread calls MARS_Unlock(). This command is for software implementing a hardware abstraction layer only. MARS cannot distinguish between software threads accessing it.

8.1.2.1 Prototype

MARS_RC MARS_Lock ();

8.1.2.2 Response Codes

- MARS_RC_SUCCESS – exclusive access acquired
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible

8.1.3 MARS_Unlock

MARS_Unlock() SHALL render inaccessible all data (e.g., plaintext, keys) provided by or for code accessing the MARS, and relinquish control of the MARS after a previous MARS_Lock().

8.1.3.1 Prototype

MARS_RC MARS_Unlock ();

8.1.3.2 Response Codes

- MARS_RC_SUCCESS – exclusive access relinquished
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked

8.1.4 MARS_CapabilityGet

This command returns various information regarding MARS capabilities according to the requested property tag. Property tags that MUST be supported are shown in Table 5 – MARS Property Tags. Additional values are reserved by the TCG. If an algorithm is not implemented (e.g. AKDF), the returned value SHALL be TPM_ALG_ERROR.

Table 5 – MARS Property Tags

Name	Value	Returned Type	Description
MARS_PT_PCR	1	size_t	number of consecutive PCRs implemented on this MARS
MARS_PT_TSR	2	size_t	number of consecutive TSRs implemented on this MARS
MARS_PT_LEN_DIGEST	3	size_t	size of a digest that can be processed or produced
MARS_PT_LEN_SIGN	4	size_t	size of signature produced by CryptSign()
MARS_PT_LEN_SKEY	5	size_t	size of symmetric key produced by CryptSkdf()
MARS_PT_LEN_AKEY	6	size_t	size of public asymmetric key produced by CryptAkdf()
MARS_PT_ ALG_HASH	7	uint32_t	TCG-registered algorithm (TCG) for hashing by CryptHash()
MARS_PT_ ALG_SIGN	8	uint32_t	TCG-registered algorithm (TCG) for signing by CryptSign()
MARS_PT_ ALG_SKDF	9	uint32_t	TCG-registered algorithm (TCG) for symmetric key derivation by CryptSkdf()
MARS_PT_ ALG_AKDF	10	uint32_t	TCG-registered algorithm (TCG) for asymmetric key derivation by CryptAkdf()
MARS_PT_CTXISKEY	11	bool	Indicates whether ctxiskey as a parameter may be passed as TRUE

8.1.4.1 Prototype

```
MARS_RC MARS_CapabilityGet (
     uint32_t pt,
     void * cap,
     size_t caplen);
```

8.1.4.2 Parameters

- pt – property tag value from Table 5 – MARS Property Tags
- cap – pointer to result defined in Table 5 – MARS Property Tags
- caplen – number of bytes in buffer provided in cap

8.1.4.3 Response Codes

- MARS_RC_SUCCESS – capability result written to cap
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_VALUE – invalid pt
- MARS_RC_SIZE – Buffer pointer or size invalid

8.2 Sequence Primitives

Functions such as hashing can consume large amounts of data as well as data from noncontiguous regions of memory. The concatenation of data to form a single parameter is known as a sequence. To support sequenced parameters, a Start/Update/Complete approach is used. The start of a function requiring a sequenced parameter(s) is via the MARS_Sequence*Func*() command, where Func refers to the type of function (e.g., Hash). Sequenced bytes to be supplied effectively as a single parameter are given via successive calls to MARS_SequenceUpdate(). Fixed (non-sequenced) parameters are specified by each MARS_Sequence*Func*() command. The end of a sequence, and possibly the start of the next, is signaled by MARS_SequenceComplete(). A null parameter is signaled by MARS_SequenceComplete() without any preceding MARS_SequenceUpdate()s.

MARS always requires data to be submitted in a sequence, even for a sequence of one. A higher-level API may provide a function (that uses sequence commands) for atomic submission of data, so callers themselves do not need to use sequence commands, but this is not a MARS requirement.

The Start/Update/Complete set of commands should not be interleaved with other MARS commands. If other commands are used, the sequence is terminated. In this event, MARS_Update() and MARS_Complete() MUST return MARS_RC_SEQ.

Start of informative comment

While the only sequenced function supported in this initial specification is for hashing, additional support is anticipated, e.g., for encrypt and decrypt functions.

End of informative comment

8.2.1 MARS_SequenceHash

A hash sequence is started by MARS_SequenceHash(). The final digest is written during MARS_SequenceComplete(). The digest's length is indicated via MARS_PT_LEN_DIGEST.

8.2.1.1 Prototype

```
MARS_RC MARS_SequenceHash ();
```

8.2.1.2 Parameters

- None

8.2.1.3 Response Codes

- MARS_RC_SUCCESS – hash sequence initiated
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_COMMAND – command not supported

8.2.2 MARS_SequenceUpdate

MARS_SequenceUpdate() SHALL process additional data under the sequenced algorithm. In the course of performing the update, MARS_SequenceUpdate() MAY, depending on the MARS_Sequence*Func*() algorithm, produce additional output that SHALL be written to the output buffer specified. The outlen parameter indicates the size of the destination buffer out. Upon return, outlen SHALL contain the number of bytes written.

8.2.2.1 Prototype

```
MARS_RC MARS_SequenceUpdate(
     const void * in,
     size_t inSize,
     void * out,
     size_t * outlen);
```

8.2.2.2 Parameters

- in – pointer to source data to be sequenced
- inSize – length of in buffer in bytes
- out – pointer to output data results
- outlen – length of out in bytes

8.2.2.3 Response Codes

- MARS_RC_SUCCESS – sequence successfully updated
- MARS_RC_COMMAND – command not supported
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_SEQ – Not preceded by Sequence start command
- MARS_RC_SIZE – Buffer pointer or size invalid

8.2.3 MARS_SequenceComplete

The end of a sequenced parameter is signaled by MARS_SequenceComplete(). The outlen parameter indicates the size of the destination buffer out. Upon return, outlen SHALL contain the number of bytes written. If additional sequenced parameters are required, then MARS_SequenceComplete() SHALL also signal the start of the next sequence.

8.2.3.1 Prototype

```
MARS_RC MARS_SequenceComplete(
     void * out,
     size_t * outlen);
```

8.2.3.2 Parameters

- out – pointer to output data results
- outlen – length of out in bytes

8.2.3.3 Response Codes

- MARS_RC_SUCCESS – sequence processed successfully
- MARS_RC_COMMAND – command not supported
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_SEQ – Not preceded by Sequence start command
- MARS_RC_SIZE – Buffer pointer or size invalid

8.3 Integrity Collection

The following commands support the implementation of the RTS as described in section 4.3.

8.3.1 MARS_PcrExtend

The specified PCR SHALL be updated with a supplied digest as described in the pseudocode.

8.3.1.1 Prototype

```
MARS_RC MARS_PcrExtend (
     uint32_t pcrIndex,
     const void * dig,
     size_t diglen);
```

8.3.1.2 Parameters

- pcrIndex – specifies which PCR to update
- dig – address containing source digest used in updating the PCR
- diglen – number of bytes in dig

8.3.1.3 Response Codes

- MARS_RC_SUCCESS – PCR extended
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_REG – invalid pcrIndex
- MARS_RC_SIZE – bad pointer or size for dig

8.3.1.4 Pseudocode

```
    $PCR_{pcrIndex}$ = CryptHash($PCR_{pcrIndex}$ || dig)
```

8.3.2 MARS_RegRead

The content of the specified register SHALL be returned by MARS_RegRead().

8.3.2.1 Prototype

```
MARS_RC MARS_RegRead (
     uint32_t regIndex,
     void * dig,
     size_t diglen);
```

8.3.2.2 Parameters

- regIndex – specifies which register to read
- dig – address to write a copy of register content
- diglen – number of bytes reserved in dig

8.3.2.3 Response Codes

- MARS_RC_SUCCESS – the contents of the selected register was returned in digest
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_REG – invalid regIndex
- MARS_RC_SIZE – Buffer pointer or size invalid

8.4 Key Management

8.4.1 MARS_Derive

MARS_Derive() SHALL use CryptSkdf() to generate bytes for external use from the DP, a device snapshot, and a label of MARS_KX. The caller's context ctx SHALL be used to distinguish between snapshots with the same regSelect. The number of bytes written can be retrieved via MARS_PT_LEN_SKEY.

8.4.1.1 Prototype

```
MARS_RC MARS_Derive (
     uint32_t regSelect,
     const void * ctx,
     size_t ctxlen,
     void * out,
     size_t outlen);
```

8.4.1.2 Parameters

- regSelect – bitmask identifying registers

- ctx – context that distinguishes between derivations with the same regSelect
- ctxlen – number of bytes in ctx
- out – destination buffer
- outlen – size of output buffer

8.4.1.3 Response Codes

- MARS_RC_SUCCESS – n bytes generated
- MARS_RC_COMMAND – command not supported
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_REG – selected register not implemented
- MARS_RC_SIZE – bad pointer or size for out

8.4.1.4 Pseudocode

```
    snapshot = CryptSnapshot(regSelect, ctx)
    *out = CryptSkdf(DP, MARS_KX, snapshot)
```

8.4.2 MARS_DpExtend

This function SHALL derive a new value of DP from the current DP, register selection, selected register values and provided context, ctx. If ctx is NULL, the DP SHALL be reset to its initial state (section 5.2.2). When binding DP to register values is needed, regSelect may specify a non-empty set of registers.

Start of informative comment

MARS_DpExtend() supports the Chain of Custody use case documented in (TCG, 2021). Additional guidance on the use of this feature may be provided in future.

End of informative comment

8.4.2.1 Prototype

```
MARS_RC MARS_DpExtend (
    uint32_t regSelect,
    const void * ctx,
    size_t ctxlen);
```

8.4.2.2 Parameters

- regSelect – bitmask identifying registers
- ctx – context for deriving a new DP
- ctxlen – number of bytes in ctx

8.4.2.3 Response Codes

- MARS_RC_SUCCESS – DP extended

- MARS_RC_COMMAND – command not supported
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_REG – selected register not implemented
- MARS_RC_SIZE – Buffer pointer or size invalid

8.4.2.4 Pseudocode

```
if (ctx)
     snapshot = CryptSnapshot(regSelect, ctx)
     DP = CryptSkdf(DP, MARS_KD, snapshot, sizeof(DP))
else
     reset DP to initial state
```

8.4.3 MARS_PublicRead

The public portion of the specified key SHALL be returned. The format of the result is dependent upon the algorithm selected within the corresponding MARS Profile. The number of bytes written can be retrieved via MARS_PT_LEN_AKEY.

Start of informative comment

Typically, endorsement of an asymmetric public key begins with the creation of a Certificate Signing Request (CSR). A CSR is signed by the paired private key. However, MARS does not currently support CSR signing. An alternate method to create an AK cert is to use the desired AK_{PUB} and a proxy CSR with metadata during the certificate creation process. For example, openssl supports this via the x509 “-force_pubkey” option.

The “-force_pubkey” option is documented in openssl as being “useful for creating certificates where the algorithm can’t normally sign requests.”

End of informative comment

8.4.3.1 Prototype

```
MARS_RC MARS_PublicRead (
     bool restricted,
     const void * ctx,
     size_t ctxlen,
     void * pub,
     size_t n);
```

8.4.3.2 Parameters

- restricted – indicates whether the specified key is restricted
- ctx – context for asymmetric key differentiation
- ctxlen – number of bytes in ctx
- pub – destination buffer
- n – size in bytes of pub buffer

8.4.3.3 Response Codes

- MARS_RC_SUCCESS – public key read
- MARS_RC_COMMAND – command not supported
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_SIZE – Buffer pointer or size invalid

8.4.3.4 Pseudocode

```
label = restricted ? MARS_KR : MARS_KU
K = CryptAkdf(DP, label, ctx)
*pub = $K_{pub}$
```

8.5 Attestation

The following commands support the implementation of the RTR and related functionality as described in section 4.4.

8.5.1 MARS_Quote

MARS_Quote() SHALL sign a snapshot of the current device state as reflected in the selected registers with the designated restricted key.

The number of bytes written to sig can be retrieved via MARS_PT_LEN_SIGN.

8.5.1.1 Prototype

```
MARS_RC MARS_Quote (
     uint32_t regSelect,
     const void * nonce,
     size_t nlen,
     const void * ctx,
     size_t ctxlen,
     void * sig,
     size_t siglen);
```

8.5.1.2 Parameters

- regSelect – bitmask identifying registers
- nonce – challenge data, same size as digest
- nlen – number of bytes in nonce
- ctx – context for AK differentiation
- ctxlen – number of bytes in ctx
- sig – location to return resulting signature
- siglen – number of bytes in sig

8.5.1.3 Response Codes

- MARS_RC_SUCCESS – signature produced
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_REG – selected register not implemented
- MARS_RC_SIZE – Buffer pointer or size invalid

8.5.1.4 Pseudocode

```
    snapshot = CryptSnapshot( regSelect, nonce )
    AK = CryptXkdf(DP, MARS_KR, ctx)
    *sig = CryptSign(AK, snapshot)
```

8.5.2 MARS_Sign

This command SHALL sign an externally provided digest with the designated unrestricted key.

The number of bytes written to sig can be retrieved via MARS_PT_LEN_SIGN.

8.5.2.1 Prototype

```
MARS_RC MARS_Sign (
    bool ctxiskey,
    const void * ctx,
    size_t ctxlen,
    const void * dig,
    size_t diglen,
    void * sig,
    size_t siglen);
```

8.5.2.2 Parameters

- ctxiskey – indicates whether ctx contains a key (instead of a context for derivation)
- ctx – context for key differentiation
- ctxlen – number of bytes in ctx
- dig – source data to be signed
- diglen – number of bytes in digest
- sig – location to return resulting signature
- siglen – number of bytes in sig

8.5.2.3 Response Codes

- MARS_RC_SUCCESS – signing successful
- MARS_RC_COMMAND – command not supported
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_SIZE – bad pointer or size parameter

8.5.2.4 Pseudocode

```
if (ctxiskey)
     key = ctx
else
     key = CryptXkdf(DP, MARS_KU, ctx)
*sig = CryptSign(key, dig)
```

8.5.3 MARS_SignatureVerify

MARS SHALL return, via the result parameter, a verdict of digital signature verification using CryptVerify(). MARS SHALL process ctx as a verification key when ctxiskey is True. Otherwise, MARS SHALL derive a verification key using the restricted and ctx parameters. If restricted is True, MARS SHALL use ctx to derive a restricted attestation key, else an unrestricted signing key. The context parameter ctx is used to generate different keys.

8.5.3.1 Prototype

```
MARS_RC MARS_ SignatureVerify (
     bool ctxiskey,
     bool restricted,
     const void * ctx,
     size_t ctxlen,
     const void * dig,
     size_t diglen,
     const void * sig,
     size_t siglen,
     bool * result);
```

8.5.3.2 Parameters

- ctxiskey – indicates whether ctx contains a key (instead of context for derivation)
- restricted – selects label for key derivation
- ctx – key or context for key differentiation
- ctxlen – number of bytes in ctx
- dig – source digest that was signed
- diglen – number of bytes in digest
- sig – signature of dig to verify
- siglen – number of bytes in sig
- result – outcome of CryptVerify

8.5.3.3 Response Codes

- MARS_RC_SUCCESS – signature verified correctly
- MARS_RC_COMMAND – command not supported
- MARS_RC_FAILURE – MARS is in failure mode or otherwise inaccessible
- MARS_RC_ACCESS – MARS is not locked
- MARS_RC_SIZE – bad pointer or size parameter

8.5.3.4 Pseudocode

```
if (ctxiskey)
     key = ctx
else {
     label = restricted ? MARS_KR : MARS_KU
     key = CryptXkdf(DP, label, ctx)
}
*result = CryptVerify(key, dig, sig)
```

9 Bibliography

ISO/IEC. (n.d.). *10116:2017, Information technology — Security techniques — Modes of operation for an n-bit block cipher.* Retrieved from https://www.iso.org/standard/64575.html

ISO/IEC. (2018, Jun). *ISO/IEC 9899:2018 Information technology — Programming languages — C.* Retrieved from https://www.iso.org/standard/74528.html

TCG. (2019, Nov 8). *TPM 2.0 Library Specification.* Retrieved from https://trustedcomputinggroup.org/resource/tpm-library-specification

TCG. (2021, Jan 27). *MARS Use Cases and Considerations.* Retrieved from https://trustedcomputinggroup.org/resource/mars-use-cases-and-considerations/

TCG. (n.d.). *TCG Algorithm Registry.* Retrieved from https://trustedcomputinggroup.org/resource/tcg-algorithm-registry/