TRUSTED® COMPUTING GROUP

SPECIFICATION

Measurement and Attestation RootS (MARS) Library Specification

Version 1
Revision 12
July 22, 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

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

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.

CONTENTS

TABLES

1 Scope

This document is the Measurement and Attestation RootS (MARS) Library Specification. It describes 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), and in a manner simple enough to reasonably implement with a cryptographic accelerator and a hardware state machine.

Those wishing to create a MARS device need to be aware that this specification does not completely define the options and commands necessary to implement a MARS. To implement a MARS device, a designer needs to refer to the relevant platform-specific "Profile" specification to understand the options and settings required in a specific type of platform or make independent choices as an implementer.

This specification does not place specific requirements on command, control and transport protocols between a host and its MARS 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

AEAD	Authenticated Encryption with Associated Data
AK	Attestation Key
CSR	Certificate Signing Request
DP	Derivation Parent
HMAC	(keyed) Hash-based Message Authentication Code
host	A computing platform with an attached MARS device
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
SoC	System on a Chip
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TSR	Trusted Sensor Register

3 Conventions

3.1 Naming Conventions

Table 1 – Conventions

Convention	Example
All Command Interface names are prefixed with “MARS_”.	See next rows
Command Interface 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
Platform-specific constants are prefixed with “PROFILE_”.	PROFILE_LEN_DIGEST

3.2 Data Types

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

- All commands SHALL return a response code of type MARS_RC (uint16_t).
- Potentially large data lengths are represented by type size_t.
- All other integer data types are uint16_t.

Start of informative comment

The following should be defined with a C18-compliant compiler:

- uint8_t, uint16_t and uint32_t are defined via stdint.h
- size_t is defined via stdlib.h
- bool, true and false are defined via stdbool.h

End of informative comment

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 2.0 Library Specification; Part 1, Architecture (TCG, 2019), and as described below. Subsections of this document's section 4 are informative.

MARS is a Root of Trust that acts as an RTS and RTR (see below) in response to commands from the host device to which it is attached. Attachment refers to a form of host processor isolation that includes physical (discrete chip on an external bus), integrated (SoC logic block on an internal bus) and logical (special execution mode) connections. The primary motivation for MARS is to provide RoT capabilities as hardware logic integrated within a host's microcontroller. However, other manifestations are possible.

Start of informative comment

4.1 Events

A code or data module that is about to be executed or processed by the host is considered an *event* that is represented by a digest/measurement produced by hashing its module's contents. An event may be conceptualized as a link in a transitive trust chain. 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 (TCG, n.d.).

4.2 Root of Trust for Measurement

A Trusted Platform is booted by a host's 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,
4. Optionally populate an event log, and
5. Execute or process the module

To maintain a transitive chain of trust of measurements, each subsequent module in an event sequence is responsible for the same five actions. Note that the RTM is distinct from the RTS and RTR. The RTM, RTS, and RTR reside on the same host device and work together to implement a Trusted Platform.

4.3 Root of Trust for Storage

Instead of storing an indefinite number of event digests in a Root of Trust (RoT), a RoT can build and store a fixed number of cumulative digests of the events as they transpire. An RTS securely maintains these cumulative digests in its Platform Configuration Registers (PCR). Since the events are also digests themselves, a PCR is said to contain a digest of digests. Just as an event is identified by its digest, an event log can be identified by its PCR.

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 and convey an attestation about the host to 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 attestation's freshness,
4. Verify the PCR's authenticity,
5. Assess the PCR's values,
6. If an event log is present,
    a. Verify the integrity of the events in the event log, and
    b. Assess the events in the event log

The RTR can convey identity using asymmetric or symmetric cryptography.

- 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 when used in a suitable cryptographic protocol.

- 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.

End of informative comment

5 MARS Device Requirements

5.1 Cryptography

To extend a PCR, a hash function is required. 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. A MARS device SHALL implement a hash function and a digital signing function. If on-demand generation of keys is required, then a MARS device SHALL implement a KDF.

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.

In keeping with MARS' minimalist approach, a single, core algorithm is recommended to be implemented to provide 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. At the time of publication of this MARS specification, the only hash, sign and KDF primitives that have the same core algorithm are SHA-256, HMAC-SHA256, and NIST SP800-108's KDF in Counter Mode with HMAC-SHA256. They appear in the TCG Algorithm Registry (TCG) with identifiers TPM_ALG_SHA256, TPM_ALG_HMAC (using the installed TPM_ALG_SHA256), and TPM_ALG_KDF1_SP800_108 respectively.

End of informative comment

5.2 Self-Testing

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. A Profile requiring self-testing MUST ensure that testable logic is tested before it is used. Any failed test causes MARS to enter failure mode (see section 5.3.1). Where self-testing is required, one of the following two behaviors MUST be specified:

1. Full testing at initialization:
   All available tests are executed during system initialization (see section 5.4).

2. Partial testing when needed:
   Only the tests available for logic required by the issuing function are executed.

For on-demand testing, MARS_SelfTest() MAY be supported (see section 8.1.1).

5.3 Device State

5.3.1 Failure Mode

MARS enters failure mode due to a failed self-test. Once in failure mode, all MARS commands, excluding MARS_CapabilityGet(), SHALL return MARS_RC_FAILURE. MARS SHALL remain in failure mode until reinitialized (see section 5.4).

5.3.2 Primary Seed (PS)

A MARS device SHALL contain a persistent Primary Seed (PS). The PS is the most critical security parameter in MARS' architecture. 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 (DP) 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.

5.3.3 Derivation Parent (DP)

The Derivation Parent is first deterministically 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

5.3.4 Selectable Registers

MARS supports two types of volatile registers that can be selected for use in a variety of commands. 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 is specified in a MARS Profile. The length of all selectable registers is the size of a digest produced by the implemented hash algorithm. PCR and TSR registers MUST be modifiable only by MARS and in the following ways – via initialization (see section 5.4), or extend (for PCR), or sampling (for TSR).

5.3.4.1 Register Selection

When choosing a group of registers to use in certain MARS commands, 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 PCR 0 through m-1 respectively, and bits m through m+n-1 represent TSR 0 through n-1 respectively. The remaining bits of regSelect, bits m+n through 31, are unused and MUST be zero.

5.3.4.2 PCR

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

MARS commands can be directed to use any subset of PCR (see section 5.3.4.1). If a non-empty set of PCR is used, then the commands' results will depend on PCR measurements.

5.3.4.3 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 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 0). 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() (see section 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 4 and 5 is issued (TSR registers 0 and 1).
- The MARS quoting command 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 4 is issued.
- A command to read register 5 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. Commands that can trigger a TSR update are MARS_Derive() (preferred, if available) and MARS_Quote(). Though MARS_DpDerive() can also trigger an update, its use is not recommended for this purpose since it also alters the DP. After the TSR are updated, the updated values can be read using MARS_RegRead().

End of informative comment

5.4 Initialization

MARS' Roots of Trust should be reset concurrently with a reset of its host and its host's RTM (see section 4.2). MARS SHALL implement a _MARS_Init signal to reset MARS.

When _MARS_Init is received, MARS SHALL perform the following:

- Initialize PCR to zero.
- Initialize TSR to Profile-specified values.
- Reset failure mode to false.
- Perform a full self-test if required by the Profile.
- Derive the volatile DP from a non-volatile PS per section 5.3.3.

After MARS successfully completes its reset, MARS MUST be ready to process commands.

5.5 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.

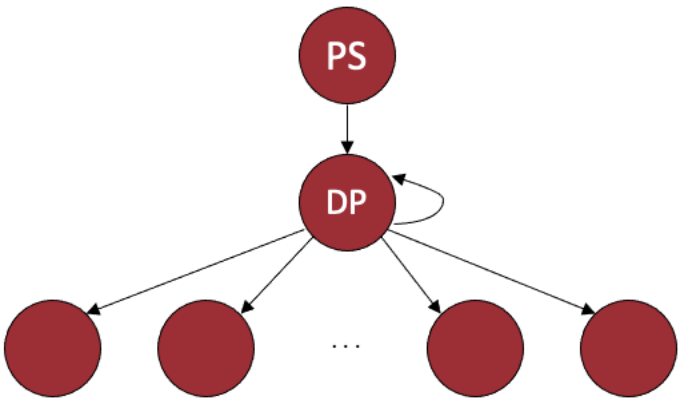

Figure 1 – Key Hierarchy

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


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

where the MARS-supplied *label* designates the purpose of the *key*, and the application specific *context* is used to differentiate keys used for the same purpose (e.g., multiple attestation keys). The one-byte ASCII *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.

*Label*s SHALL be used as defined in Table 2, padded as required by the implemented KDF.

Table 2 – Cryptographic Key Labels

Name	Value	Description
MARS_LX	'X'	eXternal
MARS_LD	'D'	Derivation Parent
MARS_LU	'U'	Unrestricted signing
MARS_LR	'R'	Restricted attestation

5.6 Support Functions

The following internal MARS functions, if implemented, must be inaccessible outside of MARS as they are critical to the MARS security model. These functions will be referenced when defining the behavior of MARS commands, but they are not part of the Command Interface.

All of these commands MUST set failure mode when

- A (partial) self-test fails, or
- Any other internal error is detected.

5.6.1 CryptSelfTest(fullTest)

If partial self-testing is supported (see section 5.2) and fullTest is false, CryptSelfTest() performs only those self-tests that have not yet been executed. Otherwise, all available self-tests are executed. Returns false if any of the executed tests fail, otherwise true.

5.6.2 CryptHash-related functions

These functions compute a one-way cryptographic hash over the supplied data using the Profile-specified hash algorithm.

5.6.2.1 CryptHashInit(shc)

Initializes the provided sequence hash context, *shc*. The parameter type of *shc* is defined by the implementation of the Profile's hash algorithm.

Start of informative comment

Note that "context" is used in two different ways in this specification. Firstly, context can refer to the operational state used by a specific function (e.g., CryptHash-related functions represent this as *shc*) or the state of MARS itself. Secondly, context can refer to an arbitrary number of bytes supplied to MARS by a host application, and as a parameter is passed as "ctx" and "ctxlen" in the Command Interface. This second type of context is used to differentiate output from multiple invocations of the same command.

End of informative comment

5.6.2.2 CryptHashUpdate(shc, data, len)

Updates the hash state in *shc* by hashing in *len* bytes of *data*.

5.6.2.3 CryptHashFinal(shc, out)

Finalizes the hash sequence and returns the resulting digest via *out*.

5.6.3 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*.

5.6.4 CryptVerify(key, digest, signature)

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

5.6.5 CryptSkdf(child, parent, label, ctx, ctxlen)

Deterministically derives a symmetric *child* key using the Profile-specified symmetric KDF from the specified *parent* secret, *label*, and application-provided context, *ctx*. This function is used when establishing the DP from the PS (see section 5.3.3), when extending the DP (see section 8.4.2), when deriving bytes for external use (see section 8.4.1), and possibly where CryptXkdf (see section 5.6.7) is used. Refer to section 5.5 for a description of the label parameter.

5.6.6 CryptAkdf(child, parent, label, ctx, ctxlen)

Deterministically derives an asymmetric *child* key pair using the Profile-specified asymmetric KDF from the provided *parent* secret, *label*, and application-provided context, *ctx*. Refer to section 5.5 for a description of the label parameter. Refer to section 5.6.7 for uses.

5.6.7 CryptXkdf

CryptXkdf is CryptAkdf if CryptAkdf is implemented. Otherwise, CryptXkdf is CryptSkdf. CryptXkdf is used to generate key for quoting (see section 8.5.1), signing (see section 8.5.2), and signature verification (see section 8.5.3).

5.6.8 CryptDpInit()

CryptDpInit() SHALL set the value of the DP using the PS and a profile-specific algorithm. CryptDpInit() is used during device initialization (see section 5.4) and by the command MARS_DpDerive() (see section 8.4.2).

5.6.9 CryptSnapshot(regSelect, ctx, ctxlen)

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
- ctx – application-provided contextual 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

$$snapshot = CryptHash\ (regSelect\ ||\ REG_{\#}\ ||\ \ldots\ ||\ REG_{\#}\ ||\ ctx)$$

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

Start of informative comment

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

$$CryptSnapshot\ (0b101,\ nonce,\ sizeof(nonce))$$

using 32-byte digests and nonce would result in a hash of these 4 + 3 * 32 = 100 bytes:

$$0\ ||\ 0\ ||\ 0\ ||\ 5\ ||\ PCR_0\ ||\ PCR_2\ ||\ nonce$$

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

End of informative comment

5.7 Session Management

MARS MUST maintain context for only a single series of commands (session). There is no mechanism to save and restore context. To help maintain the desired MARS context, the host is responsible for preventing unwanted interleaving of commands from multiple applications and threads.

5.8 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 protection against arbitrary alteration (e.g., of the PCR, TSR, or signing algorithm), some require confidentiality protection against disclosure. The _MARS_Init signal MUST be protected to prevent unauthorized initializations. Volatile secrets (e.g., DP, AK) MUST NOT be readable by anything other than MARS at run time. When MARS is powered off, profile-specific protection is anticipated for data-at-rest – specifically, the PS.

6 Constants

6.1 Profile Constants

The constants in Table 3 are used in C code provided for use in definitions of MARS Commands (see section 8). These constants are defined in a MARS Profile specification. TPM_ALG_ERROR (defined as 0) MUST be used as the definition of PROFILE_ALG_ constants for unimplemented algorithms.

Table 3 – PROFILE Constants

Name	Description
PROFILE_COUNT_PCR	number of consecutive PCR implemented on this MARS
PROFILE_COUNT_TSR	number of consecutive TSR implemented on this MARS
PROFILE_COUNT_REG	(PROFILE_COUNT_PCR + PROFILE_COUNT_TSR)
PROFILE_LEN_DIGEST	length of a digest that can be processed or produced
PROFILE_LEN_SIGN	length of signature produced by CryptSign()
PROFILE_LEN_KSYM	length of symmetric key produced by CryptSkdf() if implemented, otherwise 0
PROFILE_LEN_KPUB	length of public asymmetric key returned by MARS_PublicRead() if implemented, otherwise 0
PROFILE_LEN_KPRV	length of asymmetric key produced by CryptAkdf() if implemented, otherwise 0
PROFILE_LEN_XKDF	PROFILE_LEN_KPRV if defined, else PROFILE_LEN_KSYM
PROFILE_ALG_HASH	TCG-registered algorithm (TCG) for hashing by CryptHash functions
PROFILE_ALG_SIGN	TCG-registered algorithm (TCG) for signing by CryptSign()
PROFILE_ALG_SKDF	TCG-registered algorithm (TCG) for symmetric key derivation by CryptSkdf()
PROFILE_ALG_AKDF	TCG-registered algorithm (TCG) for asymmetric key derivation by CryptAkdf()

6.2 Response Codes

MARS commands MUST return Response Codes defined in Table 4, and as documented for each command. Other values of response code are reserved for future use by the TCG.

Table 4 – Definition of Response Code Constants

Name	Value	Description
MARS_RC_SUCCESS	0	Command executed as expected
MARS_RC_IO	1	Input / Output or parsing error
MARS_RC_FAILURE	2	Self-testing placed MARS in failure mode or MARS is otherwise inaccessible
	3	Reserved
MARS_RC_BUFFER	4	Buffer pointer (null or misaligned) or length invalid
MARS_RC_COMMAND	5	Command not supported
MARS_RC_VALUE	6	Value out of range or incorrect for command
MARS_RC_REG	7	Invalid register index specified
MARS_RC_SEQ	8	Sequence not started

7 Compliance

Table 5 specifies which MARS commands (see 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 commands add complexity beyond what would otherwise be Recommended, and MAY be implemented. A MARS Profile specification defines inclusion or exclusion of specific commands.

Table 5 – MARS Command Compliance

Command	M / R / O
MARS_SelfTest	R
MARS_CapabilityGet	M
MARS_SequenceHash	R
MARS_SequenceUpdate	R
MARS_SequenceComplete	R
MARS_PcrExtend	M
MARS_RegRead	M
MARS_Derive	R
MARS_DpDerive	O
MARS_PublicRead	M*
MARS_Quote	M
MARS_Sign	R
MARS_SignatureVerify	R

* Only if CryptAkdf() is supported

8 Command Interface

Commands within the MARS Command Interface are defined below with a behavioral description, C function prototype, parameter description, returned Response Codes, and often C code that implements each command. For informative examples, refer to the posted MARS emulator source code (Trusted Computing Group, n.d.). Although C code is provided, it is possible to create MARS devices using other languages, such as a hardware description language (HDL). Consequently, only the *behavior* of the C code below is normative. Parameter checking MAY vary as appropriate for the programming language, hardware, and Profile requirements.

Start of informative comment

The commands below are assumed to be invoked by an internal MARS dispatcher whose responsibility is to:

- Receive a command and parameters from the Profile-specified interface
- Return MARS_RC_FAILURE if in failure mode, unless the command is MARS_CapabilityGet
- Return MARS_RC_COMMAND if the command is unsupported
- Unmarshall the parameters *
- Invoke the command with the received parameters
- Return MARS_RC_FAILURE if failure mode was triggered during execution of the command
- Marshall the response code and results *
- Return the marshaled data to the requestor via the Profile-specified interface

* [Un]marshaling, if needed, may be performed within the command (not shown) instead of the dispatcher.

End of informative comment

8.1 Management

8.1.1 MARS_SelfTest

If MARS is not already in failure mode, MARS_SelfTest() invokes self-testing via CryptSelfTest(). See section 5.6.1.

Any ongoing sequenced command SHALL be cancelled, and any remaining sequence commands SHALL return MARS_RC_SEQ.

8.1.1.1 Prototype

```
MARS_RC MARS_SelfTest (
     bool fullTest);
```

8.1.1.2 Parameters

- fullTest – true to perform all tests, false to perform unexecuted tests

8.1.1.3 Response Codes

- MARS_RC_SUCCESS – all executed tests passed
- MARS_RC_FAILURE – was in or entered failure mode

8.1.1.4 C Code

```
failure = failure || !CryptSelfTest(fullTest);
return failure ? MARS_RC_FAILURE : MARS_RC_SUCCESS;
```

8.1.2 MARS_CapabilityGet

This command returns various information regarding MARS capabilities according to the requested property tag. Table 6 lists the property tags that MUST be supported. Additional values are reserved by the TCG. Refer to Table 3 for definitions of the returned values.

Table 6 – MARS Property Tags

Name	Value	Returned Type	Returned Value
MARS_PT_PCR	1	uint16_t	PROFILE_COUNT_PCR
MARS_PT_TSR	2	uint16_t	PROFILE_COUNT_TSR
MARS_PT_LEN_DIGEST	3	uint16_t	PROFILE_LEN_DIGEST
MARS_PT_LEN_SIGN	4	uint16_t	PROFILE_LEN_SIGN
MARS_PT_LEN_KSYM	5	uint16_t	PROFILE_LEN_KSYM
MARS_PT_LEN_KPUB	6	uint16_t	PROFILE_LEN_KPUB
MARS_PT_LEN_KPRV	7	uint16_t	PROFILE_LEN_KPRV
MARS_PT_ALG_HASH	8	uint16_t	PROFILE_ALG_HASH
MARS_PT_ALG_SIGN	9	uint16_t	PROFILE_ALG_SIGN
MARS_PT_ALG_SKDF	10	uint16_t	PROFILE_ALG_SKDF
MARS_PT_ALG_AKDF	11	uint16_t	PROFILE_ALG_AKDF

8.1.2.1 Prototype

```
MARS_RC MARS_CapabilityGet (
     uint16_t pt,
     void * cap,
     uint16_t caplen);
```


8.1.2.2 Parameters

- pt – property tag value from Table 6 – MARS Property Tags
- cap – pointer to result defined in Table 6 – MARS Property Tags
- caplen – number of bytes in buffer provided in cap

8.1.2.3 Response Codes

- MARS_RC_SUCCESS – capability result written to cap
- MARS_RC_VALUE – invalid pt
- MARS_RC_BUFFER – buffer pointer or length invalid

8.1.2.4 C Code

```
if (!cap || caplen != sizeof(uint16_t))
    return MARS_RC_BUFFER;
switch (pt)
    {
    case MARS_PT_PCR:
    *(uint16_t *)cap = PROFILE_COUNT_PCR;
    break;

    case MARS_PT_TSR:
    *(uint16_t *)cap = PROFILE_COUNT_TSR;
    break;

    case MARS_PT_LEN_DIGEST:
    *(uint16_t *)cap = PROFILE_LEN_DIGEST;
    break;

    case MARS_PT_LEN_SIGN:
    *(uint16_t *)cap = PROFILE_LEN_SIGN;
    break;

    case MARS_PT_LEN_KSYM:
    *(uint16_t *)cap = PROFILE_LEN_KSYM;
    break;

    case MARS_PT_LEN_KPUB:
    *(uint16_t *)cap = PROFILE_LEN_KPUB;
    break;

    case MARS_PT_LEN_KPRV:
    *(uint16_t *)cap = PROFILE_LEN_KPRV;
    break;

    case MARS_PT_ALG_HASH:
    *(uint16_t *)cap = PROFILE_ALG_HASH;
    break;
```



```
        case MARS_PT_ALG_SIGN:
        *(uint16_t *)cap = PROFILE_ALG_SIGN;
        break;

        case MARS_PT_ALG_SKDF:
        *(uint16_t *)cap = PROFILE_ALG_SKDF;
        break;

        case MARS_PT_ALG_AKDF:
        *(uint16_t *)cap = PROFILE_ALG_AKDF;
        break;

        default:
        return MARS_RC_VALUE;
        }
    return MARS_RC_SUCCESS;
```

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 such to form a single parameter is called 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 indicated by MARS_SequenceComplete(). A null parameter is indicated by MARS_SequenceComplete() without an intervening MARS_SequenceUpdate().

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

Start of informative comment

Hashing is the only sequence type supported by this revision of this MARS specification.

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 PROFILE_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

8.2.1.4 C Code

This code assumes that hashing is the only supported sequence type.

```
    CryptHashInit(&shc);
    return MARS_RC_SUCCESS;
```

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 size of the caller-provided out buffer is indicated via outlen. Upon return, outlen SHALL contain the number of bytes written to out.

8.2.2.1 Prototype

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

8.2.2.2 Parameters

- in – pointer to source data to be sequenced
- inlen – length of in buffer in bytes
- out – pointer to output data results
- outlen – see description above

8.2.2.3 Response Codes

- MARS_RC_SUCCESS – sequence successfully updated
- MARS_RC_SEQ – sequence not started
- MARS_RC_BUFFER – buffer pointer or length invalid

8.2.2.4 C Code

This code assumes that hashing is the only supported sequence type.

```
    if ((inlen && !in) || !outlen)
        return MARS_RC_BUFFER;
    if (!hash_sequence_in_progress)   // implementation dependent
        return MARS_RC_SEQ;
    CryptHashUpdate(&shc, in, inlen);
    *outlen = 0;
    return MARS_RC_SUCCESS;
```

8.2.3 MARS_SequenceComplete

The end of a sequenced parameter is indicated by MARS_SequenceComplete(). The size of the caller-provided out buffer is indicated via outlen. Upon return, outlen SHALL contain the number of bytes written to out. If additional sequenced parameters are required, then MARS_SequenceComplete() SHALL also indicate the start of the next parameter.

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 – see description above

8.2.3.3 Response Codes

- MARS_RC_SUCCESS – sequence processed successfully
- MARS_RC_SEQ – sequence not started
- MARS_RC_BUFFER – buffer pointer or length invalid

8.2.3.4 C Code

This code assumes that hashing is the only supported sequence type.

```
    if (!out || !outlen || *outlen < PROFILE_LEN_DIGEST)
        return MARS_RC_BUFFER;
    // assumes sequence is hash
    if (!hash_sequence_in_progress)   // implementation dependent
        return MARS_RC_SEQ;
    CryptHashFinal(&shc, out);
    *outlen = PROFILE_LEN_DIGEST;
    return MARS_RC_SUCCESS;
```

8.3 Integrity Collection

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

8.3.1 MARS_PcrExtend

MARS_PcrExtend() SHALL update the specified PCR with the supplied digest as described in the C Code.

8.3.1.1 Prototype

```
MARS_RC MARS_PcrExtend (
     uint16_t pcrIndex,
     const void * dig);
```

8.3.1.2 Parameters

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

8.3.1.3 Response Codes

- MARS_RC_SUCCESS – PCR extended
- MARS_RC_REG – invalid pcrIndex
- MARS_RC_BUFFER – buffer pointer invalid

8.3.1.4 C Code

```
     if (pcrIndex >= PROFILE_COUNT_PCR)
         return MARS_RC_REG;
     if (!dig)
         return MARS_RC_BUFFER;

     CryptHashInit(&shc);
     CryptHashUpdate(&shc, REG[pcrIndex], PROFILE_LEN_DIGEST);
     CryptHashUpdate(&shc, dig, PROFILE_LEN_DIGEST);
     CryptHashFinal(&shc, REG[pcrIndex]);
     return MARS_RC_SUCCESS;
```

8.3.2 MARS_RegRead

The content of the specified register SHALL be returned by MARS_RegRead(). The number of bytes written is PROFILE_LEN_DIGEST.

8.3.2.1 Prototype

```
MARS_RC MARS_RegRead (
     uint16_t regIndex,
     void * dig);
```

8.3.2.2 Parameters

- regIndex – specifies which register to read
- dig – address to write a copy of register content

8.3.2.3 Response Codes

- MARS_RC_SUCCESS – the contents of the selected register was returned in digest
- MARS_RC_REG – invalid regIndex
- MARS_RC_BUFFER – buffer pointer invalid

8.3.2.4 C Code

```
    if (regIndex >= PROFILE_COUNT_REG)
        return MARS_RC_REG;
    if (!dig)
        return MARS_RC_BUFFER;

    memcpy(dig, REG[regIndex], PROFILE_LEN_DIGEST);
    return MARS_RC_SUCCESS;
```

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_LX. The application's context ctx SHALL be used to distinguish between snapshots with the same regSelect. The number of bytes written is PROFILE_LEN_KSYM.

8.4.1.1 Prototype

```
MARS_RC MARS_Derive (
    uint32_t regSelect,
    const void * ctx,
    uint16_t ctxlen,
    void * out);
```

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

8.4.1.3 Response Codes

- MARS_RC_SUCCESS – n bytes generated
- MARS_RC_REG – selected register not implemented
- MARS_RC_BUFFER – buffer pointer or length invalid

8.4.1.4 C Code

```
    if (regSelect >> PROFILE_COUNT_REG)
```



```
    return MARS_RC_REG;
if (!out || (ctxlen && !ctx))
    return MARS_RC_BUFFER;

uint8_t snapshot[PROFILE_LEN_DIGEST];
CryptSnapshot(snapshot, regSelect, ctx, ctxlen);
CryptSkdf(out, DP, MARS_LX, snapshot, sizeof(snapshot));
return MARS_RC_SUCCESS;
```

8.4.2 MARS_DpDerive

This command 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 (see section 5.6.8). When binding DP to register values is needed, regSelect may specify a non-empty set of registers.

Start of informative comment

MARS_DpDerive() supports the Chain of Custody use case documented in (TCG, 2021).

End of informative comment

8.4.2.1 Prototype

```
MARS_RC MARS_DpDerive (
    uint32_t regSelect,
    const void * ctx,
    uint16_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_REG – selected register not implemented
- MARS_RC_BUFFER – buffer pointer or length invalid

8.4.2.4 C Code

```
    if (regSelect >> PROFILE_COUNT_REG)
        return MARS_RC_REG;
    if (ctxlen && !ctx)
        return MARS_RC_BUFFER;
```



```
    if (ctx)
        {
        uint8_t snapshot[PROFILE_LEN_DIGEST];
        CryptSnapshot(snapshot, regSelect, ctx, ctxlen);
        CryptSkdf(DP, DP, MARS_LD, snapshot, sizeof(snapshot));
        }
    else
        CryptDpInit();

    return MARS_RC_SUCCESS;
```

8.4.3 MARS_PublicRead

MARS_PublicRead() SHALL return the public portion of the specified key. The format of the result is dependent upon the algorithm selected within the corresponding MARS Profile. The number of bytes written is PROFILE_LEN_KPUB.

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 support CSR signing of restricted (AK) keys. An alternate method of creating 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,
    uint16_t ctxlen,
    void * pub);
```

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

8.4.3.3 Response Codes

- MARS_RC_SUCCESS – public key read
- MARS_RC_BUFFER – buffer pointer or length invalid

8.4.3.4 C Code

```
    if (ctxlen && !ctx)
        return MARS_RC_BUFFER;
    uint8_t key[PROFILE_LEN_KPRV];
    uint8_t label = restricted ? MARS_LR : MARS_LU;
    CryptAkdf(key, DP, label, ctx, ctxlen);
    profile_copy_pub(pub, key); // implementation dependent
    return MARS_RC_SUCCESS;
```

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 is PROFILE_LEN_SIGN.

8.5.1.1 Prototype

```
MARS_RC MARS_Quote (
    uint32_t regSelect,
    const void * nonce,
    uint16_t nlen,
    const void * ctx,
    uint16_t ctxlen,
    void * sig);
```

8.5.1.2 Parameters

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

8.5.1.3 Response Codes

- MARS_RC_SUCCESS – signature produced
- MARS_RC_REG – selected register not implemented
- MARS_RC_BUFFER – buffer pointer or length invalid

8.5.1.4 C Code

```
    if (regSelect >> PROFILE_COUNT_REG)
        return MARS_RC_REG;
```

```
    if ((nlen && !nonce) || (ctxlen && !ctx) || !sig)
        return MARS_RC_BUFFER;

    uint8_t AK[PROFILE_LEN_XKDF];
    uint8_t snapshot[PROFILE_LEN_DIGEST];
    CryptSnapshot(snapshot, regSelect, nonce, nlen);
    CryptXkdf(AK, DP, MARS_LR, ctx, ctxlen);
    CryptSign(sig, AK, snapshot);

    return MARS_RC_SUCCESS;
```

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 is PROFILE_LEN_SIGN.

8.5.2.1 Prototype

```
MARS_RC MARS_Sign (
    const void * ctx,
    uint16_t ctxlen,
    const void * dig,
    void * sig);
```

8.5.2.2 Parameters

- ctx – context for key differentiation
- ctxlen – number of bytes in ctx
- dig – source data to be signed
- sig – location to return resulting signature

8.5.2.3 Response Codes

- MARS_RC_SUCCESS – signing successful
- MARS_RC_BUFFER – buffer pointer or length invalid

8.5.2.4 C Code

```
    if (!(dig && sig) || (ctxlen && !ctx))
        return MARS_RC_BUFFER;

    uint8_t key[PROFILE_LEN_XKDF];
    // MARS_LU forces CryptXkdf to derive an unrestricted key
    CryptXkdf(key, DP, MARS_LU, ctx, ctxlen);
    CryptSign(sig, key, dig);
    return MARS_RC_SUCCESS;
```

8.5.3 MARS_SignatureVerify

MARS SHALL return, via the result parameter, a verdict of digital signature verification using CryptVerify(). 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.

8.5.3.1 Prototype

```
MARS_RC MARS_SignatureVerify (
     bool restricted,
     const void * ctx,
     uint16_t ctxlen,
     const void * dig,
     const void * sig,
     bool * result);
```

8.5.3.2 Parameters

- restricted – selects label for key derivation
- ctx – context for key differentiation
- ctxlen – number of bytes in ctx
- dig – source digest that was signed
- sig – signature of dig to verify
- result – outcome of CryptVerify

8.5.3.3 Response Codes

- MARS_RC_SUCCESS – signature verified correctly
- MARS_RC_BUFFER – buffer pointer or length invalid

8.5.3.4 C Code

```
     if (!(dig && sig && result) || (ctxlen && !ctx))
         return MARS_RC_BUFFER;

     uint8_t key[PROFILE_LEN_XKDF];
     uint8_t label = restricted ? MARS_LR : MARS_LU;
     CryptXkdf(key, DP, label, ctx, ctxlen);
     *result = CryptVerify(key, dig, sig);
     return MARS_RC_SUCCESS;
```


9 Bibliography

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.). *Canonical Event Log Format*. Retrieved from https://trustedcomputinggroup.org/resource/canonical-event-log-format/

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

Trusted Computing Group. (n.d.). *MARS Repository*. Retrieved from https://github.com/TrustedComputingGroup/MARS