TRUSTED® COMPUTING GROUP

SPECIFICATION

MARS API Specification

Version 1
Revision 2
May 9, 2023

Contact: admin@trustedcomputinggroup.org

PUBLISHED

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.

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.

Joerg Borchert, Infineon Technologies

Tom Brostrom, Cyber Pack Ventures, Inc.

Eoin Carroll, Toyota Motor North America

Brian Dziki, United States Government

Mariza Marrero, United States Government

Vadim Sukhomlinov, Google LLC

Dick Wilkins, Phoenix Technologies, Inc.

CONTENTS

TABLES

FIGURES

1 Scope

This document is the Measurement and Attestation RootS (MARS) Application Programmer Interface (API) Specification. It defines a C-language interface for host application code to utilize a MARS device. Host application code means any software/firmware executing on a host including UEFI, boot loaders, operating system, loadable libraries, and high-level applications. The intended audience for this specification includes software developers and designers implementing applications for MARS as specified in [1].

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 which is 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 statement.

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

Table 1 – Abbreviations, Acronyms and Terms Used

API	Application Programmer Interface
host	A computing platform with an attached MARS device
MARS	Measurement and Attestation RootS
MMIO	Memory Mapped Input Output
TCG	Trusted Computing Group
UEFI	Unified Extensible Firmware Interface

3 Introduction

The purpose of this API specification is to enable seamless usage of all available MARS commands with minimal dependency on the choice of MARS architecture or how commands, parameters and responses are formatted and delivered. The remainder of section 3 is informative.

Start of informative comment

MARS architectures are described here as serialized or memory mapped. This specification does not place limits on the type of MARS architecture.

3.1 Serialized Architecture

Externally attached MARS devices (e.g., outside the host's microcontroller die) are accessed via serialization. In a serialized MARS architecture, each command to and response from the MARS device is marshaled into individual blocks that are then transmitted. An API command that is designed to take a potentially large parameter (such as MARS_SequenceUpdate()) may need to be implemented to fragment the parameter to fit within message length limits, resulting in multiple commands issued by the API to MARS. This action is transparent to the caller of the API.

The position of the API layer in the serialized logical architecture is shown in Figure 1.

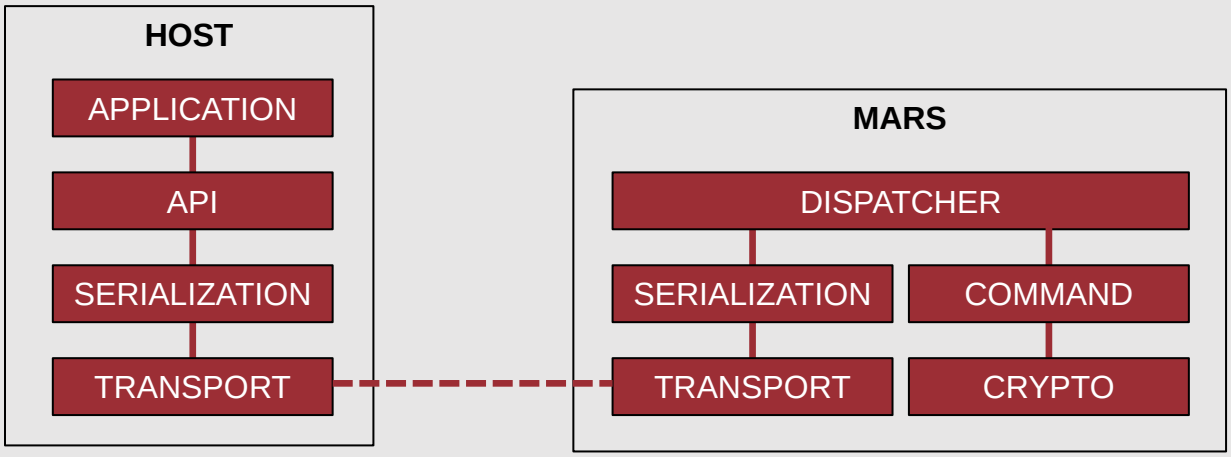

Figure 1 – MARS Serialized Architecture

A description of each of the logical layers follows:

- Application – software/firmware using the MARS API
- API – the layer described by this specification
- Serialization – marshals and unmarshalls commands, parameters and responses between the host API and the MARS dispatcher
- Transport – delivers serialized data between the host and MARS

- Command – implements the MARS commands as defined in the MARS Library Specification
- Crypto – cryptographic services utilized by MARS Commands

3.2 Memory Mapped Architecture

Internally attached MARS devices (e.g., within the host's microcontroller die) are accessed using the host's MMIO functionality. An MMIO-based MARS architecture can be depicted as shown in Figure 2. Since it may not be necessary to serialize commands into blocks, as with the MARS serialized architecture, large parameters could be streamed to MARS via direct register writes.

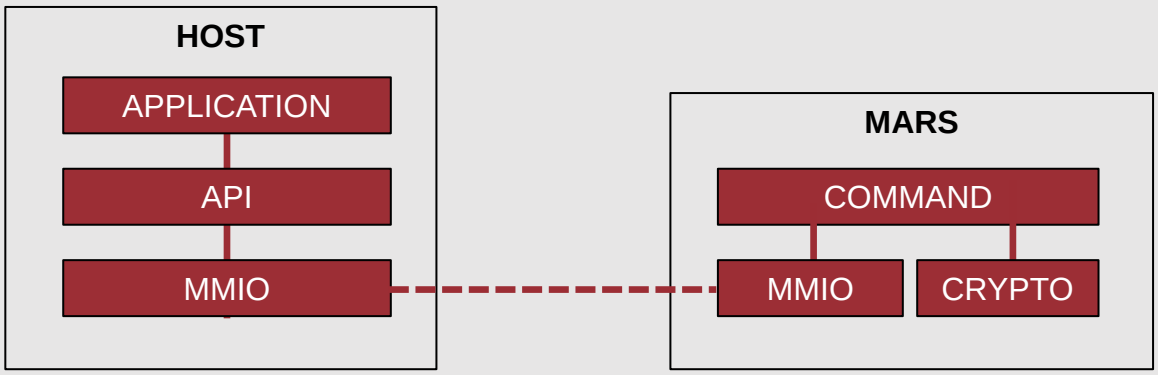

Figure 2 – MARS Memory Mapped Architecture

A description of each of the logical layers follows:

- Application – software/firmware using the MARS API
- API – the layer described by this specification
- MMIO – a memory mapped input/output device controller for an internally attached MARS device
- Command – implements the MARS commands as defined in the MARS Library Specification
- Crypto – cryptographic services utilized by MARS Commands

End of informative comment

4 API

4.1 Response Codes

When MARS API commands (see section 4.3) succeed, they SHALL return a response code of MARS_RC_SUCCESS along with their specified results. When a requested MARS API command cannot perform its function, then it SHALL return a single response code from the table of Response Code Constants [1] other than MARS_RC_SUCCESS. This API SHALL return MARS_RC_COMMAND for commands that are not supported. Additionally, the API SHALL return a response code of MARS_RC_LOCK in the event of locking errors (see Table 2 and section 4.3.3).

Table 2 – Additional Response Codes

Name	Value	Description
MARS_RC_LOCK	3	not locked or calling thread already locked

4.2 Transport

Start of informative comment

The MARS API assumes that the transport mechanism used by the API in the MARS Serialized Architecture has been properly initialized prior to use of any other MARS API functions. Refer to documentation from the provider of the MARS API for more detail.

End of informative comment

4.3 Commands

A MARS API MUST implement all the functions described in this section.

4.3.1 Command Interface

To make usage of MARS commands as seamless as possible, the API MUST implement all the MARS Command Interface definitions specified in [1]. Thus, the dispatcher can be viewed as a proxy for the host API.

Start of informative comment

In this API, MARS_RegRead() uses the serialization and transport layers to convey the command code MARS_CC_RegRead (from section 5.2.9) and its parameter *regIndex* to the MARS dispatcher. The MARS dispatcher then invokes its MARS_RegRead() that retrieves the contents of the specified register. The dispatcher then serializes and transports the response code MARS_RC_SUCCESS (assuming the register index was valid) and the copied register back to the host API where they are returned to the calling application.

End of informative comment

4.3.2 Initialization

MARS_ApiInit() is responsible for setting the initial context needed by the MARS API, such as its mutex and constants retrieved from the MARS device. MARS_ApiInit() is designed to be used once after transport initialization (see section 4.2) and before other MARS API functions are called. If not used in this manner, the MARS API functions SHALL return MARS_RC_IO.

4.3.2.1 Prototype

```
MARS_RC MARS_ApiInit ();
```

4.3.2.2 Response Codes

- MARS_RC_SUCCESS – API initialized successfully
- MARS_RC_IO – error during initialization

4.3.3 Threading

MARS maintains context for only a single series of commands (a session). There is no mechanism to save and restore context.

MARS_Lock() implicitly uses and records an implementation-specific thread or context identifier to restrict subsequent MARS API commands to being issued by the thread/context with the same identifier. Other MARS API commands MUST check that the identifier is the same as recorded by the last MARS_Lock(). If not, the commands SHALL return MARS_RC_LOCK.

MARS_Unlock() clears the recorded identifier.

Start of informative comment

To help maintain a desired MARS context, an operating system could be used to prevent interleaving of multiple sessions amongst processes (e.g., via exclusive device access).

A critical series of MARS commands is a series that is intolerant of unexpected changes in MARS' state. Consequently, a thread depending on MARS state that could be altered by another thread's execution should call MARS_Lock() once at the series' beginning, and MARS_Unlock() at the series' end. For example, suppose that thread A is used to quote PCR 0 and thread B is used to extend PCR 0. It is incorrect for thread A to:

```
MARS_Lock();
MARS_Quote(1, …);
MARS_Unlock();

MARS_Lock();
MARS_RegRead(0, …);
MARS_Unlock();
```


since thread B could update PCR 0 before thread A can read what it quoted. The correct, and simpler, implementation of thread A is:

```
MARS_Lock();
MARS_Quote(1, …);
MARS_RegRead(0, …);
MARS_Unlock();
```

Another example of a critical series is use of MARS Sequence Primitives. MARS does not have a mechanism for saving and restoring Sequence context, so those primitives must be used within a single MARS_Lock() and MARS_Unlock() pair, as in:

```
MARS_Lock();
MARS_SequenceHash();
MARS_SequenceUpdate(…);
…
MARS_SequenceUpdate(…);
MARS_SequenceComplete(…);
MARS_Unlock();
```

End of informative comment

4.3.3.1 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 other thread calls MARS_Unlock().

4.3.3.1.1 Prototype

```
MARS_RC MARS_Lock ();
```

4.3.3.1.2 Response Codes

- MARS_RC_SUCCESS – exclusive access acquired
- MARS_RC_LOCK – calling thread already locked

4.3.3.2 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().

4.3.3.2.1 Prototype

```
MARS_RC MARS_Unlock ();
```

4.3.3.2.2 Response Codes

- MARS_RC_SUCCESS – exclusive access relinquished
- MARS_RC_LOCK – not locked

4.4 Single Threading

In non-threaded environments, simplified implementations of MARS_Lock() and MARS_Unlock() may be implemented. As an example, refer to the informative code snippet below.

Start of informative comment

```
static bool locked = false;
MARS_RC MARS_Lock()
{ return locked ? MARS_RC_LOCK : (locked = true , MARS_RC_SUCCESS); }

MARS_RC MARS_Unlock()
{ return locked ? locked = false , MARS_RC_SUCCESS : MARS_RC_LOCK; }
```

End of informative comment

5 Header Files

5.1 mars/api.h

```
#pragma once

#include "mars.h"

#define MARS_RC_LOCK    3   // not locked or calling thread already locked

MARS_RC MARS_ApiInit ();
MARS_RC MARS_Lock ();
MARS_RC MARS_Unlock ();
```

5.2 mars/mars.h

5.2.1 Prelude

```
#pragma once
#include <stdint.h> // for uint32_t, etc.
#include <stdlib.h> // for size_t
#include <stdbool.h> // for bool, true, false
```

5.2.2 Property Tags

```
#define MARS_PT_PCR 1    // uint16_t number of consecutive PCRs
implemented on this MARS
#define MARS_PT_TSR 2    // uint16_t number of consecutive TSRs
implemented on this MARS
#define MARS_PT_LEN_DIGEST 3  // uint16_t size of a digest that can be
processed or produced
#define MARS_PT_LEN_SIGN 4    // uint16_t size of signature produced by
CryptSign()
#define MARS_PT_LEN_KSYM 5    // uint16_t size of symmetric key produced
by CryptSkdf()
#define MARS_PT_LEN_KPUB 6    // uint16_t size of asymmetric key returned
by MARS_PublicRead()
#define MARS_PT_LEN_KPRV 7    // uint16_t size of private asymmetric key
produced by CryptAkdf()
#define MARS_PT_ALG_HASH 8    // uint16_t TCG-registered algorithm for
hashing by CryptHash()
#define MARS_PT_ALG_SIGN 9    // uint16_t TCG-registered algorithm for
signing by CryptSign()
#define MARS_PT_ALG_SKDF 10   // uint16_t TCG-registered algorithm for
symmetric key derivation by CryptSkdf()
#define MARS_PT_ALG_AKDF 11   // uint16_t TCG-registered algorithm for
asymmetric key derivation by CryptAkdf()
```


5.2.3 Response Codes

```
typedef uint16_t MARS_RC;

#define MARS_RC_SUCCESS  0   // Command executed as expected
#define MARS_RC_IO       1   // Input / Output or parsing error
#define MARS_RC_FAILURE  2   // self-testing placed MARS in failure mode
or MARS is otherwise inaccessible
// reserved              3
#define MARS_RC_BUFFER   4   // Invalid buffer pointer (null or
misaligned) or length
#define MARS_RC_COMMAND  5   // Command not supported
#define MARS_RC_VALUE    6   // Value out of range or incorrect for
context
#define MARS_RC_REG      7   // Invalid register index specified
#define MARS_RC_SEQ      8   // Not preceded by Sequence start command
```

5.2.4 Management Commands

```
MARS_RC MARS_SelfTest (bool fullTest);

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

5.2.5 Sequence Primitives

```
MARS_RC MARS_SequenceHash ();

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

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

5.2.6 Integrity Collection Commands

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

MARS_RC MARS_RegRead (
```



```
    uint16_t regIndex,
    void * dig);
```

5.2.7 Key Management Commands

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

MARS_RC MARS_DpDerive (
    uint32_t regSelect,
    const void * ctx,
    uint16_t ctxlen);

MARS_RC MARS_PublicRead (
    bool restricted,
    const void * ctx,
    uint16_t ctxlen,
    void * pub);
```

5.2.8 Attestation Commands

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

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

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


5.2.9 Command Codes

```
#define MARS_CC_SelfTest            0
#define MARS_CC_CapabilityGet       1
#define MARS_CC_SequenceHash        2
#define MARS_CC_SequenceUpdate      3
#define MARS_CC_SequenceComplete    4
#define MARS_CC_PcrExtend           5
#define MARS_CC_RegRead             6
#define MARS_CC_Derive              7
#define MARS_CC_DpDerive            8
#define MARS_CC_PublicRead          9
#define MARS_CC_Quote               10
#define MARS_CC_Sign                11
#define MARS_CC_SignatureVerify     12
#define MARS_CC_LAST                12
```


6 Bibliography

[1] TCG, "MARS Library Specification, v1r14," 2 Jan 2023. [Online]. Available: https://trustedcomputinggroup.org/resource/mars-library-specification/.

[2] Trusted Computing Group, "MARS Repository," [Online]. Available: https://github.com/TrustedComputingGroup/MARS.

