TRUSTED® COMPUTING GROUP

SPECIFICATION

TCG TSS 2.0 Response Code API Specification

Version 1.0
Revision 12
May 21, 2019

Contact: admin@trustedcomputinggroup.org

PUBLISHED

Disclaimers, Notices, and License Terms

Acknowledgements

TCG and the TSS Work Group would like to thank the following people for their work on this specification:

- Brenda Baggaley OnBoard Security
- Mike Cox OnBoard Security
- Lee Wilson OnBoard Security
- Andreas Fuchs Fraunhofer SIT
- Ken Goldman IBM
- William Roberts Intel
- Tadeusz Struk Intel
- Philip Tricca Intel

Contents

1 Information and Document Scope

1.1 Scope of this Specification

The TSS2 RC API defines an interface to a utility library. Its intended purpose is to provide application developers with a mechanism for converting instances of the TSS2_RC type into user-readable text. The intended audience for this specification includes software developers and designers implementing TCG TPM2 Software Stack APIs as well as those making use of TCG TPM2 Software Stack APIs in software systems.

1.2 Acronyms

For definitions of the acronyms used in the TSS 2.0 specifications please see the TCG TSS 2.0 Overview and Common Structures Specification [22}.

1.3 Key Words

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

1.4 TCG Software Stack 2.0 (TSS 2.0) Specification Structure

At the time of writing, the documents that specify the TSS 2.0 are:

[1] TCG TSS 2.0 Overview and Common Structures Specification

[2] TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification

[3] TCG TSS 2.0 Marshaling/Unmarshaling API Specification

[4] TCG TSS 2.0 System API (SAPI) Specification

[5] TCG TSS 2.0 Enhanced System API (ESAPI) Specification

[6] TCG TSS 2.0 Feature API (FAPI) Specification

[7] TCG TSS 2.0 TAB and Resource Manager Specification

[8] TCG TSS 2.0 TSS Response Code API Specification

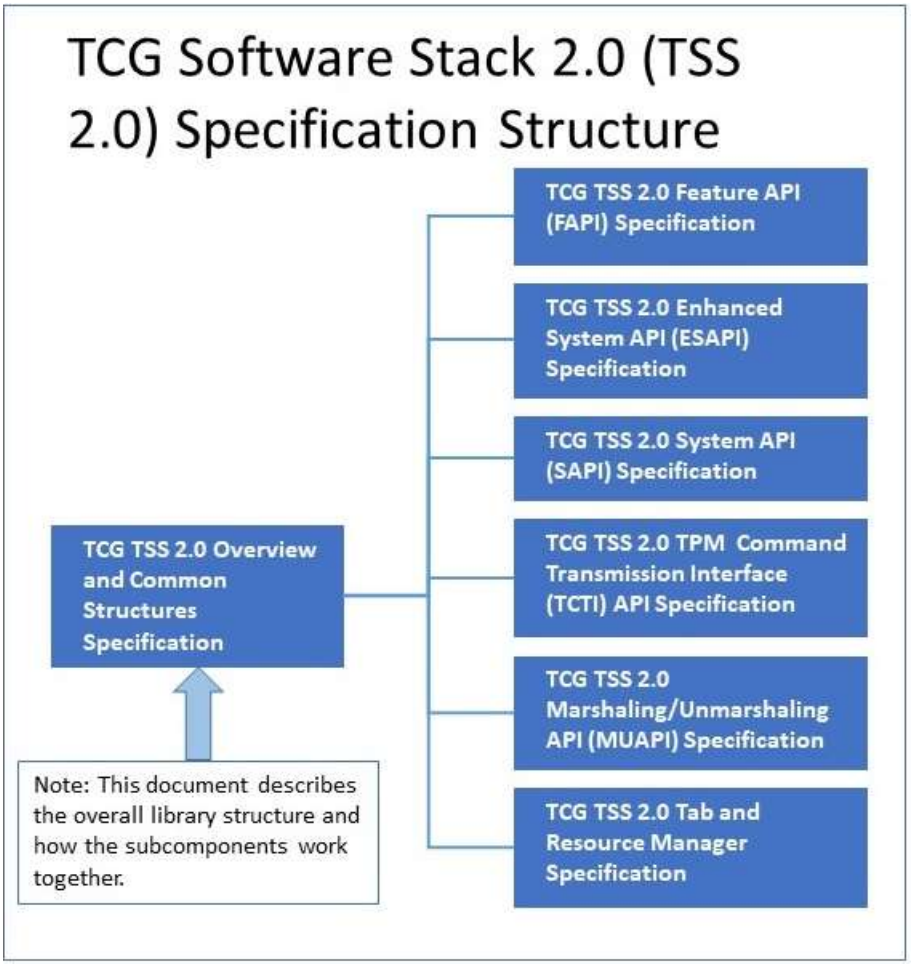

Figure 1: TSS 2.0 Specification Structure

1.5 References

Most references for the TSS 2.0 specifications are provided in the TCG TSS 2.0 Overview and Common Structures Specification [1].

The following additional references are used by this specification. The numbering continues from the previous section:

[9] IETF RFC 3447, Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
[10] ISO C99 standard
[11] Trusted Platform Module Library Part 1: Architecture Family “2.0”
[12] Trusted Platform Module Library Part 2: Structures Family “2.0”

2 Introduction

Response codes returned by TPM 2.0 devices and the TSS2 APIs follow the encoding scheme described in [1][11]. Developers using the TSS2 APIs need to provide actionable information to users of their applications (e.g. system administrators, end users). The `TSS2_RC` (`UINT32`) encoding is insufficient for this purpose as it leaves the burden of decoding this value into a human understandable form to the user.

This issue is addressed for c99 compliant systems through the `strerror` function[10]. The reader is assumed to be familiar with this function. The specification described in this document uses a similar method for mapping response code values in applications using the TSS2 API `TSS2_RC` type to human readable strings. Accommodations for the additional flexibility provided by the TSS2 layer encoding scheme in response codes are made through a callback registration mechanism described in section 3.3.

The reader is assumed to have an understanding of the response code encoding scheme documented in the section titled *Response Code Details* from [11], the section titled *TPM_RC (Response Codes)* from [12] and the section titled *Common Return Codes* from [1].

3 Library Interface

The following subsections describe the two functions in this library as well as specific requirements for implementers. The first function in this library maps `TSS2_RC` values returned from TSS2 API calls into human readable strings. The second is an interface to register functions for custom error decoding using the layer field described in [1].

3.1 Decoding Errors to a String

Function prototype:

```
const char *Tss2_RC_Decode(TSS2_RC rc);
```

The '`Tss2_RC_Decode`' function MUST map the provided `TSS2_RC` to a `NULL` terminated string. The contents of this string MUST adhere to the formats described in section 3.1.2. Implementations of this library MUST respect the '`const`' qualifier on values returned from this function and MUST NOT require the caller to perform memory management operations on the data returned.

NOTE: that passing the `TSS2_RC` value of `TSS2_RC_SUCCESS` results in the message of *'success'*.

3.1.1 Error String Formats

Strings returned by the `Tss2_RC_Decode` shall be in the format: *<layer-name>:<layer-specific-msg>*

The *<layer-name>* MUST be either a human readable string representation of the layer name, or the base 10 layer number if no handler is able to provide a name.

The *<layer-specific-msg>* MUST be a human readable string describing the response code, or the hex representation of the response code with the prefix '0x' when it cannot be decoded.

Example: assuming the default configuration and a non-decodable TSS2_RC value, if the `TSS2_RC` is of value *0x1002A* then the output would be: *1:0x2A*.

Table 1 – Default Layers

Layer ID	Layer Name	Description
0	tpm	Errors returned by the TPM itself.
6	fapi	Errors returned by the Feature API (FAPI).
7	esapi	Errors returned by the Enhanced System API (ESAPI).
8	sapi	Errors returned by the System API (SAPI).
9	mu	Errors returned by the marshaling library.
10	tcti	Errors returned by the Transmission Control Interface (TCTI).
11	rm	Errors returned by the Resource Manager (RM).
12	rmt	Errors returned by the Resource Manager (RM) TPM layer.

Implementations of this library MUST require no configuration before they are able to decode `TSS2_RC` values via calls to `Tss2_RC_Decode`. Additionally, they MUST be pre-configured to handle `TSS2_RC` values produced by the layers defined in Table 1. The decoding routine `Tss2_RC_Decode` MUST be limited to manipulating only thread local data. This allows different threads to have different layer handlers. A second call to this function from the same thread MAY invalidate data returned by previous calls.

3.1.1.1 TPM Layer 0 special messages

The *tpm* or layer 0 produces two distinct error message formats: format 0 and format 1.

3.1.1.1.1 Format 0 Messages

The *<layer-specific-msg>* field for format 0 response codes MUST be in the format: *<error|warn>(<1.2|2.0>): <description>*

Examples of Format 0 *<layer-specific-msg>* strings:

> *error(1.2): bad tag*
>
> *warn(2.0): the 1st handle in the handle area references a transient object or session that is not loaded*

Values for the *<description>* field are described in section 3.1.1.2.

3.1.1.1.2 Format 1 Messages

The *<layer-specific-msg>* field for format 1 response codes MUST be in the format: *<handle|session|parameter>(<index>):<description>* where the *index* value is displayed in its base 10 representation.

Values for the *<description>* field are described in section 3.1.1.2 below.

Examples of Format 1 Error Messages:

> *parameter(3):structure is the wrong size*

3.1.1.2 Description Value

The text used in the *<description>* field *MAY* match the *description* field in the table titled *Definition of (UINT32) TPM_RC Constants (Actions) <OUT>* in [12]. If the implementation is unable to map the error field to a *description* string then the value from the error field MUST be displayed in hex notation with a leading '0x'.

3.2 Layer Handlers

The numeric layer values defined in [1] are for known layers as well as a range of values for use by libraries and applications. Implementations of these layers are provided with a collection of common error codes in [1]. To accommodate this flexibility, the response decoding library MUST provide an interface to register a layer specific response code decoding function. This function MUST have the following prototype:

```
TSS2_RC_HANDLER Tss2_RC_SetHandler(uint8_t layer, const char *name, TSS2_RC_HANDLER handler);
```

The `TSS2_RC_HANDLER` type MUST be defined in the `tss2_rc.h` header as:

```
typedef const char *(*TSS2_RC_HANDLER)(TSS2_RC rc);
```

3.2.1 Parameters

The *layer* parameter specifies the layer from which the *handler* is capable of decoding response codes. Valid values are in the range of 0 to 255.

The *name* parameter is a string used for the *<layer-name>* value in the format string described in section 3.1.1. The value MUST be either a string of up to 16 characters (not including the terminating null byte) or `NULL`. If `NULL` or length 0 (the empty string), this library MUST substitute a string representation of the layer number in base 10. If the string length is greater than 16, the string MUST be truncated to a length of 16.

The *handler* parameter specifies the callback function used to decode `TSS2_RC` values with the layer field set to the *layer* ID provided in the first parameter. Handler functions implementing this interface MUST rely only on the error

bits in the TSS2_RC parameter as described in the section titled *Response Code Details* in [11]. Other fields in the TSS2_RC parameter SHOULD be set to 0 by the caller. If the handler function is passed a TSS2_RC value that it cannot decode then it SHOULD return NULL. If the TSS2_RC is known to the handler function, the value returned to the caller MUST be a static string representation of the error code. If the handler provided replaces a default layer, the descriptions returned by the new handler function SHOULD be formatted in the same way as the default handler function being replaced. If the *handler* parameter is NULL then the library MUST clear the currently registered handler function for the layer.

3.2.2 Return Value

Tss2_RC_Set_Handler MUST return the TSS2_RC_HANDLER replaced by the handler parameter. If no handler was previously registered for this layer then the value returned MUST be NULL.

4 Response Code Header File

tss2_rc.h

4.1 tss2_rc.h prelude

```
#ifndef TSS2_RC_H
#define TSS2_RC_H

#include <stdint.h>
#include <tss2/tss2_tpm2_types.h>
```

4.2 tss2_rc.h type definitions

```
typedef const char *(*TSS2_RC_HANDLER)(TSS2_RC rc);
```

4.3 tss2_rc.h function prototypes

```
const char *Tss2_RC_Decode(TSS2_RC rc);
TSS2_RC_HANDLER Tss2_RC_SetHandler(uint8_t layer, const char *name, TSS2_RC_HANDLER handler);
```

4.4 tss2_rc.h postlude

```
#endif
```

