TPM Service Command Response Buffer Interface Over FF-A

Document number	DEN0138
Document quality	ALP
Document version	1.0
Document confidentiality	Non-confidential

TPM Service Command Response Buffer Interface Over FF-A

Release information

Date	Version	Changes
2023/Jun/28	v1.0 ALP0	• First release.

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Contents

TPM Service Command Response Buffer Interface Over FF-A

Preface

This specification is meant to describe one approach to TPM integration on Arm systems, namely *firmware-based* TPM integration.

About this document

This document describes TPM service firmware in terms of an interface based on the TCG TPM Command-Response Buffer interface and the Arm Firmware Framework for A-profile.

Using this document

This document consists of explanatory and normative sections, as follows:

- Chapters 1 - 6 describe the TPM service features and provide their explanation;
- Chapters 7 and 8 specify the TPM service interface.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://developer.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *TCG ACPI Specification*. (family “1.2” and “2.0”, version 1.3, revision 8) Trusted Computing Group.

[2] *TCG PC Client Platform TPM Profile Specification for TPM 2.0*. (version 1.05, revision 14) Trusted Computing Group.

[3] *TPM 2.0 Mobile Command Response Buffer Interface*. (family “2.0”, level 00, revision 12) Trusted Computing Group.

[4] *TCG TSS 2.0 Overview and Common Structures Specification*. (version 1.0, revision 10) Trusted Computing Group.

[5] *DEN0077A Arm Firmware Framework for Arm A-profile*. (version 1.1) Arm.

[6] *A Universally Unique IDentifier (UUID) URN Namespace*. IETF - Network Working Group.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (TPM Service Command Response Buffer Interface Over FF-A).
- The number (DEN0138 1.0).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Chapter 1
Introduction

Platform implementors may wish to implement a standard TPM with access control or access indirection through firmware. For example, with Arm TrustZone, some platform implementors may wish to grant privileged access to the TPM only to a Secure principal, or to enable Secure World firmware to interpose itself at the TPM interface to the Normal World. In other use cases indirection through firmware helps circumvent platform limitations, or enables firmware implementation of the TPM. Some of this TPM access indirection through firmware may already be supported with the standard Arm-SMC TPM Start Method specified by the TCG ACPI specification [1].

This document specifies the FF-A TPM Start Method and TPM-frontend firmware that supports the broader TPM2 CRB protocol and is in accordance with the TCG PC-Client-Platform TPM Profile specification [2], for systems with firmware based on the Arm Firmware Framework for A-profile (FF-A).

The FF-A TPM Start Method has uses similar to those of the Arm-SMC Start Method, however, usage of the FF-A Start Method encourages the application of the principle of least privilege and the standardisation of Arm firmware architectures. The FF-A Start Method also offers improvements in integrity and implementation complexity.

1.1 Technical background

The TPM interface for software

The Trusted Platform Module (TPM) is a platform device designed to render the trust in a platform manufacturer to the system itself. The TPM bus-level interface is often protected from most of the platform, while the TPM interface for software enables trusted system-software to restrict the TPM functionality available to untrusted software.

A TPM interface for software is the *Command-Response Buffer* address-mapped interface (CRB interface). Broadly, the CRB underpins the following usage protocol:

1. Software writes a TPM command to the Command Buffer;
2. Software notifies the TPM that a command is ready to be processed by writing a control bit;
3. Software waits for the TPM to process the command by polling a status bit;
4. TPM processes the command from the Command Buffer and writes the result to the Response Buffer;
5. TPM notifies the software that command processing is complete by writing the status bit that the software is polling;

The integral TPM interface consists of multiple instances of the CRB address-mapped interface such that each can be assigned to different software components through platform memory-protection and/or virtual-memory mechanisms. Individual CRB interfaces provide differing capabilities to access the TPM functionality. The CRB interfaces are collectively referred to as *CRB localities*. The CRB localities associate with the TPM localities, which decentralise some TPM functionality into notional *Locality 0* .. *Locality 4*, where Locality 4 is typically accessible only to a highly-privileged entity. CRB Locality 4 must not be accessible to untrusted software entities such as the OS.[1]

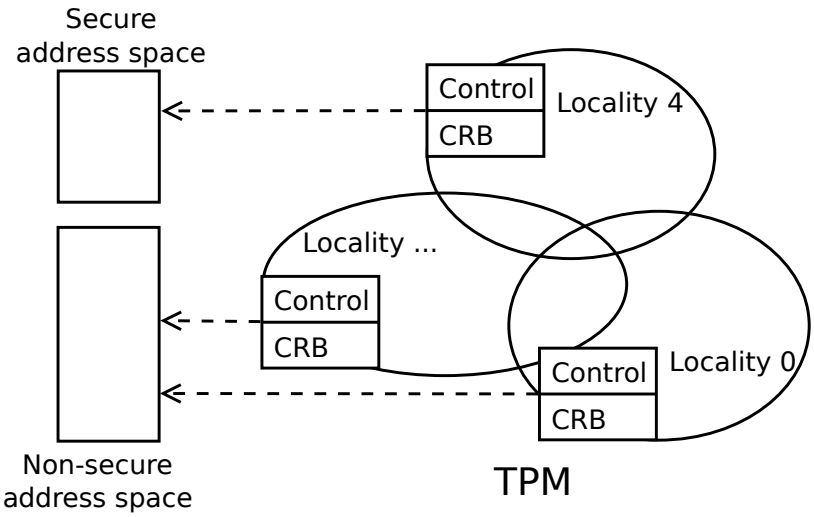

Figure 1.1: Logical representation of TPM localities and TPM CRB interface example in memory.

[1] This description applies to the TPM interface specified by the TCG PC-Client-Platform TPM Profile Specification [2].

Chapter 2
Overview

The FF-A TPM Start Method is an FF-A direct message sent to a *TPM service*. This document models the TPM service as described in Chapter 3. The modelled TPM service consists of a set of functions, and input-output data. The functions are specified by this document, whereas the input-output data consists of control and command-response data standardised by TCG specifications as TPM CRB localities [2][3].

The complete TPM service function ABI is specified by Chapter 8.

One of the TPM service functions is the FF-A TPM Start Method discussed in Chapter 4. A client invokes it to initiate the processing of a command it has stored in a CRB locality. When the TPM service finishes processing a command, the TPM service signals this event through a bit in the CRB, in accordance with the standard TPM2 CRB protocol [2][3]. In addition, the TPM service may send the client an FF-A notification of the event as Chapter 5 describes.

A client must determine compatibility with the TPM service, and must know the FF-A ID of the TPM service and the address of the desired CRB's memory region. Chapter 6 describes the function ABI version compatibility and extension scheme. Chapter 7 specifies how to set up access to the TPM service and how to use FF-A direct messages to access its functions. The complete TPM service function ABI is specified by Chapter 8.

An overview of the TPM CRB protocol over the FF-A is depicted in Figure 2.1, below.

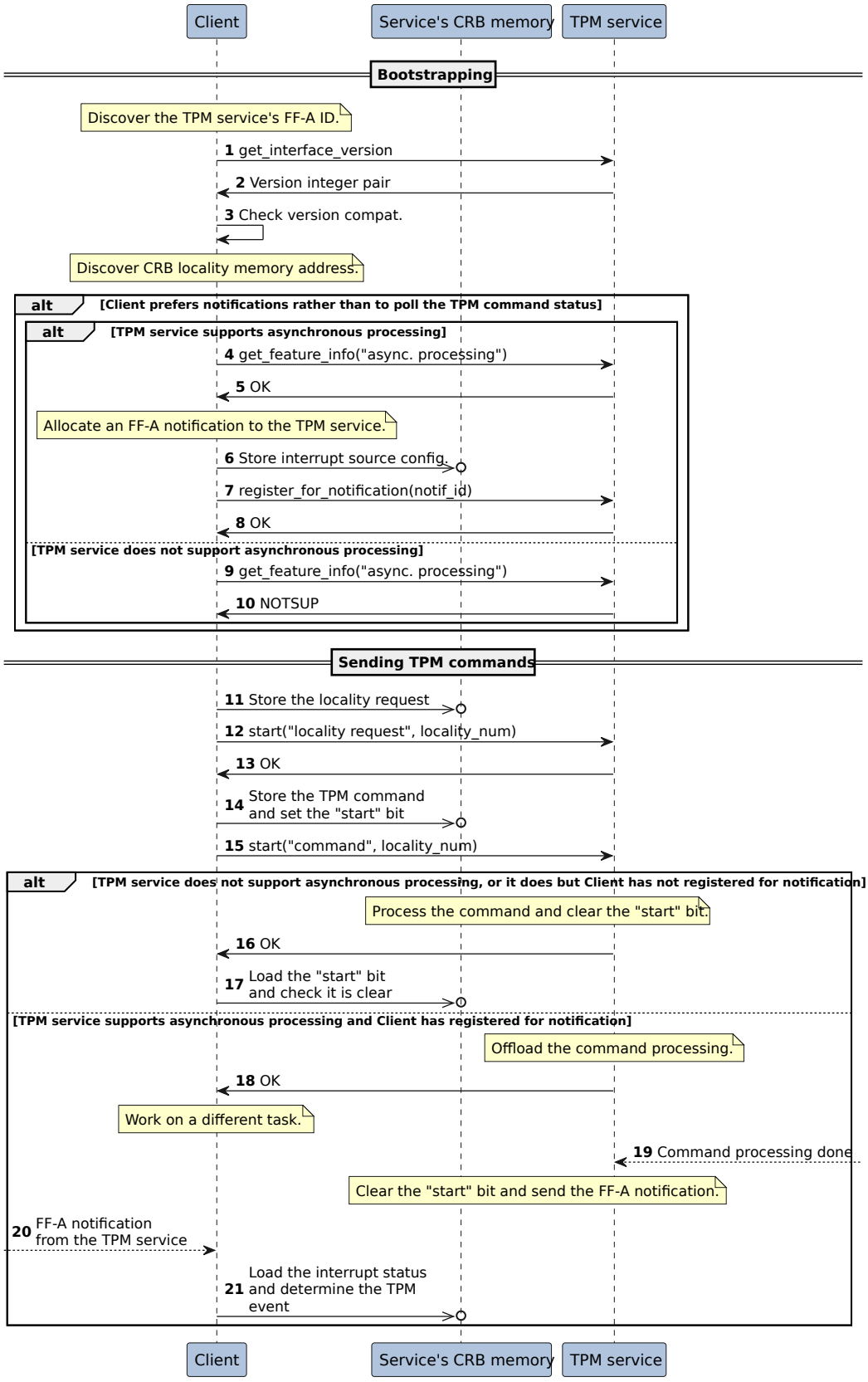

Figure 2.1: Overview of the TPM CRB protocol over FF-A.

Chapter 3
The TPM service model

The TPM service is firmware contained within an FF-A Partition. It consists of a set of functions, and input-output data in the form of standard TPM CRB localities [2][3]. The functions are accessed through the FF-A direct messaging ABI, while the CRB localities are accessed through fixed RAM regions. Figure 3.1 illustrates the TPM service model.

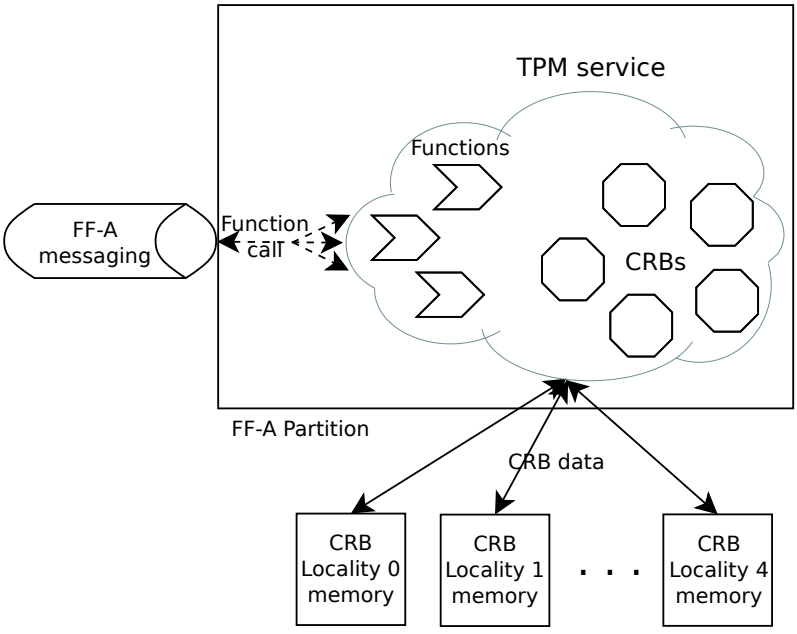

Figure 3.1: The TPM service model adopted by this specification.

The purpose of adopting this TPM service model is to enable the TPM service to implement standard TPM CRB localities within FF-A, and hence be compatible for the TCG Software Stack [4].

The way in which the TPM service meets the TCG requirement [2] that its CRB Locality 4 is protected from untrusted software entities is IMPLEMENTATION DEFINED. Firmware might, for example, use TrustZone memory protection by laying the TPM service's CRB Locality 4 in a Secure memory region and implementing the TPM service in a Secure FF-A Partition with access to this Secure memory region.

The TPM service carries out its functions in an IMPLEMENTATION DEFINED way. TPM service implementations might consist of a frontend tasked with read-writing the TPM service access interface and with driving a backend, which might in turn drive a discrete TPM, or itself implement the TPM in software.

Chapter 4
TPM start method

The FF-A TPM Start Method is an FF-A direct message that is sent to the TPM service and that indicates the `start` TPM service function call. The FF-A TPM Start Method brings about the isolation of the TPM service into an FF-A Partition, and in this way, it is conducive to firmware-privilege separation.

Compared to the Arm-SMC Start Method, the FF-A Start Method has additional parameters that simplify TPM service's task of scanning its CRB interface and improve the function's integrity. The FF-A Start Method also returns a status, which may be an error status.

An FF-A TPM Start Method call means that a client allows the TPM service to process a TPM command immediately, within FF-A direct message processing. However, the TPM service may return successfully from the call even if the TPM operation is still in progress. The TPM service must indicate when the TPM command processing is complete by clearing the CRB Start bit, and the client must wait for this indication as per the standard TPM CRB protocol.

Long-running synchronous command-processing within the TPM service

An FF-A Start Method call that returns successfully only when the TPM operation completes is said to trigger *synchronous command-processing* within the TPM service.

The TPM service may use one of the following FF-A features to prevent synchronous command-processing from impacting the responsiveness of the system:

- Interruptions by the system. The TPM service allows the framework to pause the command processing directly and to yield to the relevant FF-A partition. The client may receive an FFA_INTERRUPT status, which requires the client to resume the TPM service using FFA_RUN;
- Voluntary pauses. The TPM service pauses using FFA_YIELD. The client is required to resume the TPM service using FFA_RUN;

This document does not make special TPM service function ABI provisions to support the FF-A Managed Exit procedure, that is, interruptions by the system featuring TPM service's management. In particular, this document does not provide a TPM service function ABI for re-entry after a TPM service's managed exit.

Chapter 5
TPM service notifications

TPM service instances that are able to process commands asynchronously may signal completion of processing using FF-A Notifications. For example, a Secure World TPM service that processes commands by offloading them to a TPM device with interrupt support for signalling completion of processing may relay the signal back to the command initiator by sending it an FF-A notification. By definition, a TPM service processes commands asynchronously when it returns from the FF-A Start Method call successfully even if the TPM operation is still in progress.

TPM service notifications are an FF-A signalling method whereby client FF-A partitions can be notified. If the TPM service supports asynchronous command processing, client FF-A partitions may register for being notified of TPM events using an FF-A direct message that indicates the `register_for_notification` TPM service function call.

TPM events occur in connection to TPM operations initiated by an FF-A Start Method call. The CRB Interrupt Control register [2] controls which TPM events should generate a notification. The CRB Interrupt Status register provides the TPM event activation status, to be read and cleared by the FF-A partition receiving the notification.

An instance of the TPM service notifications mechanism is represented in the following figures. Figure 5.1 shows the sequence of events and FF-A messages for a client of a TPM service registering for notifications. Figure 5.2 shows the same for receiving a notification when a TPM operation completes.

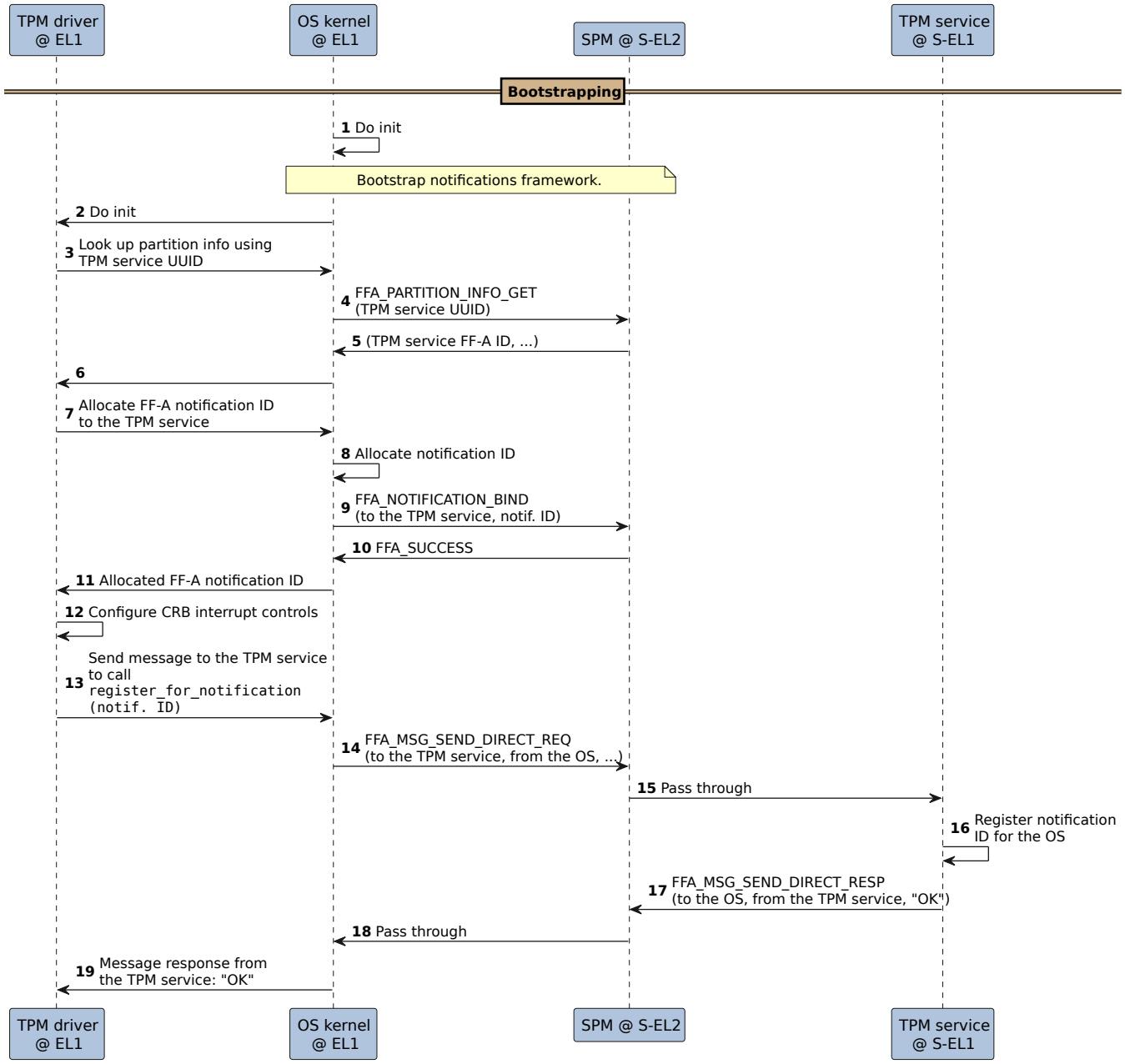

Figure 5.1: Illustration of a client and a TPM service setting up FF-A notifications.

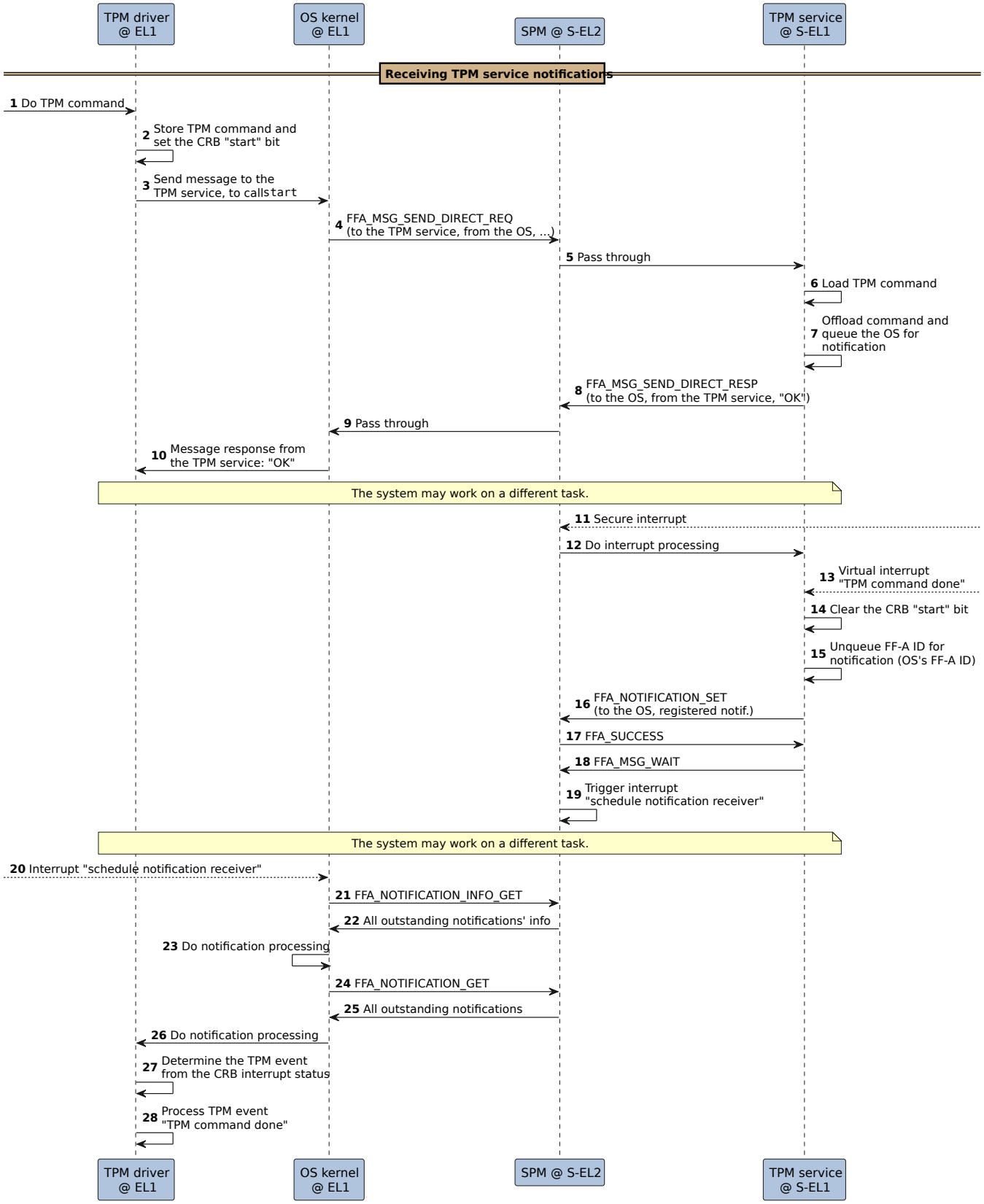

Figure 5.2: Illustration of a client and a TPM service using FF-A notifications.

Chapter 6
TPM service interface versioning

6.1 Client-service compatibility

Clients of the TPM service must check whether the TPM service is compatible with their supported version of the interface before using the TPM service's interface. A client can find out the version of the TPM service's interface using an FF-A direct message that indicates the `get_interface_version` TPM service function call.

A version of the interface is represented by a pair of integers, where the first integer is called the 'major' element and the second is called the 'minor' element. The interface in this version of the specification is version (1, 0).

Future versions of the interface will be represented by pairs of integers with the following significance:

- If a major element is equal to ($=$) another, then a minor element greater than ($>$) another represents a version of the interface that is newer than the other;
- A major element will be equal to ($=$) a major element representing an older interface only if a TPM service adhering to the newer interface continues to carry out an identical function for function calls that are in accordance with the older interface specification.

Hence the TPM service is compatible with a client if and only if its major version element is equal to ($=$) the client's and its minor version element is greater or equal to ($\geq$) the client's.

6.2 Extending the TPM service interface

Future versions of the interface may introduce new function ABIs, while particular client-service implementations may wish to include implementation-defined function ABIs. This section specifies integer space allocations to facilitate the extension of the TPM service interface.

Table 6.1: Allocation of the function ID space.

Function ID space (32-bit integer)	**Allocation**
`0x1fxx_xxxx`	IMPLEMENTATION DEFINED function IDs.
The remainder	Reserved for future versions of the interface.

Table 6.2: Allocation of the function status code space.

Function status code space (32-bit integer)	**Allocation**
`0x15xx_xxxx`	IMPLEMENTATION DEFINED function success status.
`0x9exx_xxxx`	IMPLEMENTATION DEFINED function error status.
The remainder	Reserved for future versions of the interface.

Chapter 7
Accessing the TPM service

The TPM service functions are accessed through FF-A direct messages and return FF-A direct responses, as specified by Section 7.2 *Accessing the TPM service functions*. The TPM service's CRB localities [2][3] are accessed in memory.

7.1 TPM service discovery

The TPM service's FF-A partition ID must be discovered in order to address it FF-A direct messages and access its functions. The discovery is done with an FFA_PARTITION_INFO_GET call that provides the TPM service's UUID as argument.

Provisional

TPM service UUID: UUID is reserved for the Beta quality.

Under FF-A v1.2 or later, the message addressing mode required by the TPM service must also be determined at the time of service discovery; it is an implementation choice whether the TPM service is addressed messages by only the FF-A partition ID, or by both the FF-A partition ID and the TPM service UUID.

CRB memory discovery

The mechanism for discovering the CRB-locality memory regions is IMPLEMENTATION DEFINED. The TPM service and associated firmware might support a standard discovery interface to the OS, such as the TPM2 ACPI table, as well as use an own mechanism for providing the information to other firmware components.

7.2 Accessing the TPM service functions

The Arm-SMC template for accessing the TPM service's functions is specified by Table 7.2. The SMC is a call to send an FF-A direct message to the TPM service.

If the message is delivered successfully, the SMC response follows the template specified by Table 7.3. The SMC response is an FF-A direct message response from the TPM service.

If the FF-A direct message delivery or processing fails, the SMC response is of the form shown in Table 7.4, specified in the FF-A specification.

Under FF-A v1.2 or later, the TPM service discovery may indicate that the service must be addressed messages by both the FF-A partition ID and the service UUID. Function accesses for such TPM services are through a different FF-A direct message primitive, as specified in When the TPM service is addressed messages by the UUID in addition to the FF-A partition ID.

Table 7.2: SMC template for TPM service function calls.

Register	Argument	Value
w0	FF-A function ID.	`0x8400006f`, that is, ID of FFA_MSG_SEND_DIRECT_REQ.
w1	Sender and receiver FF-A partition IDs.	Bit[31:16]: own FF-A ID; variable, assigned by FF-A, obtained through FFA_ID_GET. Bit[15:0]: TPM service's runtime FF-A ID; variable, Section 7.1 *TPM service discovery* describes how it may be discovered.
w2	Message FF-A flags.	0 – the direct message type is partition message.
w3	Not used.	0
w4 - w7	TPM service function params.	Specified by Chapter 8 *TPM service function ABI*.

Table 7.3: SMC response template for TPM service function calls.

Register	Return argument	Value
w0	FF-A function status.	`0x84000070`, that is, ID of FFA_MSG_SEND_DIRECT_RESP – the message was delivered successfully and a message response has been returned.
w1	Sender and receiver FF-A partition IDs.	Bit[31:16]: TPM service's runtime FF-A ID. Bit[15:0]: own FF-A ID.
w2	Message FF-A flags.	0 – the direct message response type is partition message.
w3	Not used.	0
w4 - w7	TPM service function status parameters.	Specified by Chapter 8 *TPM service function ABI*.

Table 7.4: SMC response form for an error of FF-A direct message delivery.

Register	Return parameter	Values
w0	FF-A function status.	• `0x84000060`, that is, ID of FFA_ERROR – message has failed to reach the TPM service; or • `0x84000062`, that is, ID of FFA_INTERRUPT – message processing has been interrupted and must be resumed; or • `0x8400006c`, that is, ID of FFA_YIELD – message processing has been paused by the the TPM service itself, and must be resumed.
w1 - w7	FF-A function status details.	Available in the FF-A specification [5].

When the TPM service is addressed by the UUID in addition to the FF-A partition ID

Accessing the TPM service functions with an additional service UUID qualifier only differs in the choice of FF-A messaging ABI, the function access is still an FF-A direct message exchange as described in the previous section.

The SMC template for function calls is specified by Table 7.5. Both the client and the service must be in the AArch64 execution mode for the SMC, so that they can access the 64-bit parameter registers.

The SMC response templates for both successful and failed FF-A direct message delivery are the same as shown in Table 7.3 and, respectively, Table 7.4.

Table 7.5: SMC template for function calls to a TPM service that must be addressed by FF-A partition ID and UUID.

Register	Argument	Value
w0	FF-A function ID.	`0xc400008a`, that is, ID of FFA_MSG_SEND_DIRECT_REQ2.
w1	Sender and receiver FF-A partition IDs.	Bit[31:16]: own FF-A ID; variable, assigned by FF-A, obtained through FFA_ID_GET. Bit[15:0]: TPM service's runtime FF-A ID; variable, Section 7.1 *TPM service discovery* describes how it may be discovered.
x2	TPM service UUID Lo part.	The TPM service UUID fields spanning bytes 0 - 7 specified by RFC 4122 [6]. **Provisional** Bits[63:0]: UUID is reserved for the Beta quality.
x3	TPM service UUID Hi part.	The TPM service UUID fields spanning bytes 8 - 15 specified by RFC 4122 [6]. **Provisional** Bits[63:0]: UUID is reserved for the Beta quality.
w4 - w7	TPM service function params.	Specified by Chapter 8 *TPM service function ABI*.

Chapter 8
TPM service function ABI

8.1 get_interface_version

Returns the version of the interface that is available.

The interface in this version of the specification is version (1, 0), as per Section 6.1 *Client-service compatibility*.

Table 8.1: SMC for the `get_interface_version` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	**Provisional** Function ID encodings are reserved for the Beta quality.
w5 - w7	Reserved for future ABI versions.	0

Table 8.2: SMC response for the `get_interface_version` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function status.	• `OK_RESULTS_RETURNED` – the version of the interface has been returned.
w5	TPM service interface version.	Bits[31:16]: major element of the version integer pair. Bits[15:0]: minor element of the version integer pair.
w6 - w7	Reserved for future ABI versions.	0

8.2 get_feature_info

Returns information on a given feature of the TPM service.

Table 8.3: SMC for the `get_feature_info` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	**Provisional** Function ID encodings are reserved for the Beta quality.
w5	TPM service feature.	• `FEAT_0` – support for asynchronous command processing and notifications. See Chapter 4 for this feature's specification. **Provisional** Feature ID encoding reserved for the Beta quality.
w6 - w7	Reserved for future ABI versions.	0

Table 8.4: SMC response for the `get_feature_info` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service feature status.	Success status: • `OK` – the feature is present; or Error status: • `INVARG` – the given feature ID is not valid; or • `NOTSUP` – the feature is not present.
w5 - w7	Reserved for future ABI versions.	0

8.3 start

Notifies the TPM service that a TPM command or TPM locality request is ready to be processed, and allows the TPM service to process it.

Table 8.5: The FF-A Start Method SMC.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	**Provisional** Function ID encodings are reserved for the Beta quality.
w5	TPM service `start` function qualifier.	Bits[31:8]: 0. Bits[7:0] command type: • 0 – the function call notifies the TPM service that a command is ready to be processed; or • 1 – the function call notifies the TPM service that a locality request is ready to be processed.
w6	TPM service `start` function locality qualifier.	Bits[31:8]: 0. Bits[7:0] TPM CRB locality: One of 0 .. 4: • The locality where the command is, if the function qualifier argument “command type” represents 0; or • The locality where the locality request is, if the function qualifier argument “command type” represents 1.
w7	Reserved for future ABI versions.	0

Table 8.6: The FF-A Start Method SMC response.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function status.	Success status: • `OK` – the TPM service has been notified successfully; or Error status: • `INVARG` – one or more arguments are not valid; or • `INV_CRB_CTRL_DATA` – CRB control data or locality control data at the given TPM locality is not valid; or • `DENIED` – firmware has previously disabled locality requests and command processing at the given locality.
w5 - w7	Reserved for future ABI versions.	0

8.4 register_for_notification

Register the calling FF-A partition for being sent an FF-A notification when a TPM event occurs.

Table 8.7: SMC for the `register_for_notification` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	**Provisional** Function ID encodings are reserved for the Beta quality.
w5	FF-A notification type.	Bit[16] notification type qualifier: • 0 – notification ID identifies a global FF-A notification; or • 1 – notification ID identifies a per-vCPU FF-A notification. Bits[15:0] destination FF-A vCPU ID: • The FF-A vCPU ID that should be passed to FFA_NOTIFICATION_SET [5] when a notification is sent, if Bit[16] is 1; or • 0, if Bit[16] is 0;
w6	FF-A notification ID.	Bits[31:8]: 0. Bits[7:0]: destination notification ID.
w7	Reserved for future ABI versions.	0

Table 8.8: SMC response for the `register_for_notification` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function status.	Success status: • `OK` – the calling FF-A partition has been registered for notification. Error status: • `INVARG` – one or more arguments are not valid; or • `NOTSUP` – asynchronous command processing and therefore notification about its completion is not supported; or • `ALREADY` – the calling FF-A partition is already registered for notification; or • `DENIED` – an other FF-A partition has already been registered for notifications, if notifying a single FF-A partition is supported only.
w5 - w7	Reserved for future ABI versions.	0

8.5 unregister_from_notification

Unregister the calling partition from being sent an FF-A notification when a TPM event occurs.

Table 8.9: SMC for the `unregister_from_notification` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	**Provisional** Function ID encodings are reserved for the Beta quality.
w5 - w7	Reserved for future ABI versions.	0

Table 8.10: SMC response for the `unregister_from_notification` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 7.2 *Accessing the TPM service functions*.
w4	TPM service function status.	Success status: • `OK` – the calling FF-A partition has been unregistered from notification. Error status: • `NOTSUP` – asynchronous command processing and therefore notification about its completion is not supported; or • `DENIED` – the calling FF-A partition is not registered for notification.
w5 - w7	Reserved for future ABI versions.	0

8.6 All function status

Provisional

Table 8.11: TPM service function status codes.

Mnemonic	Description	Code
Error status:		
NOFUNC	No such function.	Status
NOTSUP	Function not supported.	encodings
INVARG	Invalid argument.	are
INV_CRB_CTRL_DATA	Invalid Command-Response Buffer control data.	reserved
ALREADY	Function is not available in the current state.	for
DENIED	Function is not available in the current state.	the
Success status:		Beta
OK	Function succeeded.	quality.
OK_RESULTS_RETURNED	Function succeeded and results have been returned.	

Glossary

TCG Trusted Computing Group

TPM Trusted Platform Module