TPM Service Command Response Buffer Interface Over FF-A

Document number	DEN0138
Document quality	BET
Document version	v1.0 BET
Document confidentiality	Non-confidential

TPM Service Command Response Buffer Interface Over FF-A

Release information

Date	Version	Changes
2024/Jan/16	v1.0 BET	• Introduced encodings for FIDs and for all function status; also introduced the TPM service UUID; • Completed the "Client notification interface", which facilitates the TPM service notifications feature; • Introduced the new TPM service function "finish_notified()" used for client notification handling (also known as "finish()"); • Added the specification of the TPM service CRB interface and its usage; • Updated the function access through FFA_MSG_SEND_DIRECT_REQ2, in accordance with the up-to-date FF-A specification release (FF-A v1.2 ALP1); • Added information about the assumptions of this software architecture and the Trusted Computing Base through the chapter "Implementation considerations"; • Added information about compatibility with the legacy Arm TPM Start Method through a section in the Appendix.
2023/Jun/28	v1.0 ALP	• 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 informative and normative sections, as follows:

- Chapters 1,2,7, the Glossary and the Appendix provide auxiliary information;
- Chapter 3 - 6 specify the TPM service interface.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

`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.

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.

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

[4] *Trusted Platform Module Library Specification*. (family "2.0", level 00, revision 01.59) Trusted Computing Group.

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

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

[7] *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, create a ticket at https://support.developer.arm.com.

As part of the ticket, include:

- The title (TPM Service Command Response Buffer Interface Over FF-A).
- The number (DEN0138 v1.0 BET).
- The section name to which your comments refer.
- The page numbers to which your comments refer.
- The rule identifiers to which your comments refer, 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, platform implementors may wish to grant some TPM access to the Normal World, while reserving the privileged access for a Secure principal. In other use cases indirection through firmware helps circumvent platform limitations, enables firmware implementation of the TPM, or enables abstraction of a platform TPM device. Some of this TPM access indirection through firmware may already be supported with the *Arm-SMC* TPM Start Method standardised by the TCG ACPI specification [1].

This document specifies the *FF-A* TPM Start Method and frontend TPM 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) [3].

The FF-A TPM Start Method has uses similar to those of the Arm-SMC Start Method, however, adoption 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 difficulty.

1.1 Technical background

The TPM interface for software

The Trusted Platform Module (TPM) is a platform device designed to translate trust in the platform manufacturer into trust in the system itself. A TPM interface for software is the *Command-Response Buffer* address-mapped interface (CRB interface).

Broadly, the Command-Response Buffer 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, or expects a related CRB interrupt;
4. TPM processes the command from the Command Buffer and writes the result to the Response Buffer;
5. TPM signals that command processing is complete by writing the status bit that the software is polling;
6. If software configured the CRB interrupt, the TPM notifies the software that command processing is complete by raising the interrupt.

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]

In this way, the TPM interface for software provides for restricting the TPM functionality that is available to untrusted software. The TPM hardware-level interface is often protected from most of the platform.

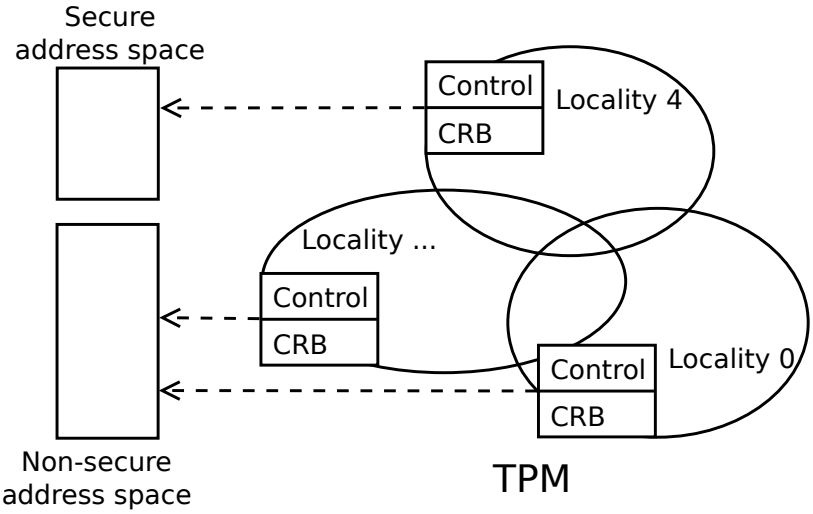

Figure 1.1: Logical representation of TPM localities as sets of TPM capabilities, and example TPM CRB localities in the address space as gateways to TPM localities.

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

1.2 Overview of this specification

The FF-A TPM Start Method is an FF-A direct message sent to a **TPM service**. The purpose of the TPM service is to receive TPM commands and return TPM responses, which are TCG standard [4].

The features that the TPM service must have are described in Chapter 2. The TPM service consists of a set of functions, and input-output memory. The functions are specified by this document, whereas the input-output memory's function and structure is standardised by TCG specifications as *TPM CRB localities* [2][5].

The TPM service CRB localities are described in Section 2.1. The complete TPM service function ABI is specified by Chapter 6.

One of the TPM service functions is the FF-A TPM Start Method described in Section 2.2. 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][5]. In addition, the TPM service may send the client an FF-A Notification of the event – notifications are described in Section 2.3.

A client must determine compatibility with the TPM service, and must know the FF-A ID of the TPM service and the memory address of the CRB locality required. Section 2.4 describes the function ABI version compatibility and extension scheme. Chapter 3 specifies how to access the TPM service CRBs and how to use FF-A direct messages to access its functions.

Chapter 4 specifies the structure of the TPM service CRBs, which form the TPM service CRB interface. Chapter 5 specifies the interface through which the service may send a notification to a client. The complete TPM service function interface is specified by Chapter 6.

Chapter 7 collects assorted implementation considerations.

An overview of the TPM2 CRB protocol over the FF-A is depicted in Figure 1.2, below.

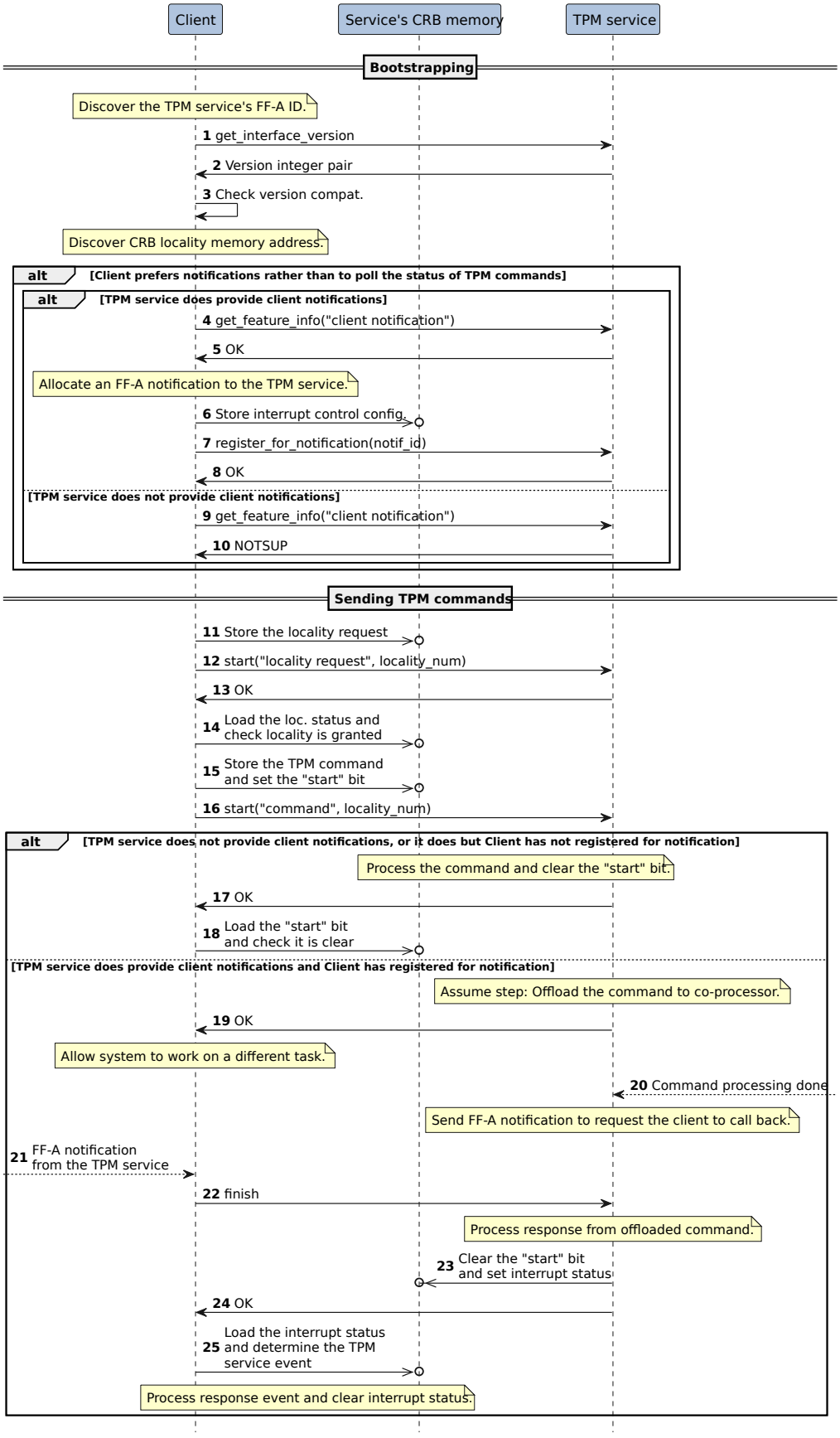

Figure 1.2: Overview of the TPM2 CRB protocol over FF-A.

Chapter 2
The TPM service

The TPM service is responsible with receiving TPM commands and returning TPM responses. It consists of a set of functions and standard TPM2 CRB localities [2][5], for TPM command input and TPM response output. The functions are accessed through the FF-A direct messaging ABI, while the CRBs are accessed through RAM. The TPM service is firmware contained within an FF-A Partition. Figure 2.1 illustrates the TPM service interface.

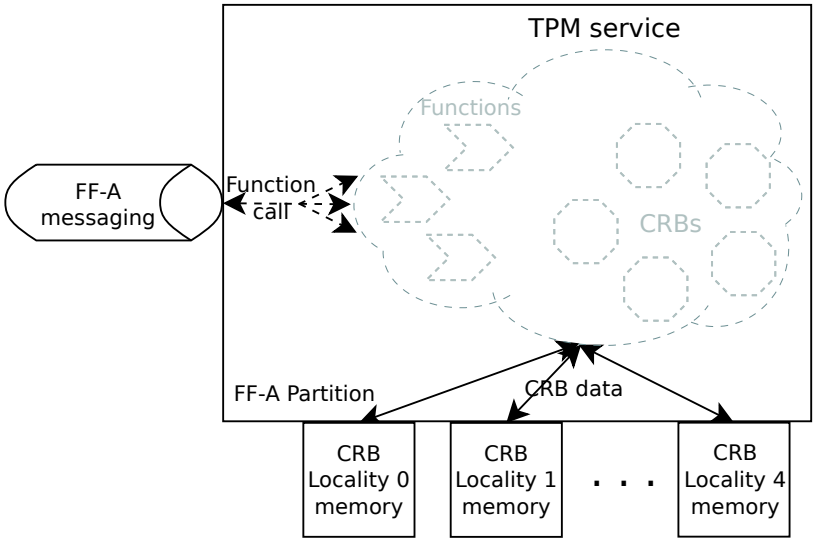

Figure 2.1: Representation of the TPM service interface defined by this specification.

This choice of TPM service interface enables the TPM service to be compatible for the TCG Software Stack [6] while leveraging the Arm FF-A, since the interface consists of standard TPM CRB localities and protocol, based on FF-A.

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.

The TPM service must execute within an FF-A partition, but *where* in the FF-A runtime, that is, under which FF-A partition manager the TPM service FF-A partition executes is IMPLEMENTATION DEFINED.

2.1 TPM service command-response buffers

The TPM service receives TPM commands and returns TPM responses through a standard TPM2 Command-Response Buffer (TPM2 CRB) interface specified by TCG [2][5].

The TPM service provides one CRB structure in a fixed RAM region, for each TPM service locality. Clients transmit TPM commands and the TPM service returns TPM responses by driving the CRB structures in memory in accordance with the TPM2 CRB protocol, which is summarised in the Introduction.

Firmware implementation lays the RAM region for each CRB structure using knowledge of the platform memory map, fixing the RAM regions so that the CRB localities' physical addresses are known ahead of run-time. This is required for the TPM service FF-A partition and client FF-A partitions to declare access requirements to the CRB structures, for the FF-A framework to ensure run-time access.

The TPM service CRB interface is specified by Chapter 4 *TPM service CRB interface*.

2.2 TPM start method

In the TPM2 CRB protocol, the TPM Start Method defines how clients of a TPM can start the processing of a TPM command. The TPM service must support the standard CRB protocol and therefore must feature a TPM Start Method. The TPM service start method is the FF-A TPM Start Method specified in this document.

The FF-A TPM Start Method is an FF-A direct message that is sent to the TPM service and that encodes the `start` TPM service function call. The `start` function is specified by Section 4.3 *The `start` notifier function*, and the `start` function ABI is specified in the TPM service function ABI section.

Compared to the Arm-SMC Start Method, 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. Also, the FF-A Start Method has additional parameters that simplify TPM service's task of scanning its CRB interface and improve the function's virtues – e.g. the FF-A Start Method can return an error status for added integrity.

2.3 TPM service notifications

TPM service notifications is an FF-A signalling method whereby client FF-A partitions can be notified about TPM service events and therefore avoid polling the TPM service CRB interface. Client notification is an optional feature of the TPM service.

If the TPM service has the client notification feature, client FF-A partitions may register for being notified of TPM service events using an FF-A direct message that encodes the `register_for_notification` TPM service function call.

TPM service events occur in connection to TPM operations initiated by an FF-A Start Method call. The client controls which TPM service CRB events generate a notification by configuring the Interrupt Control register [2] in the TPM service CRB interface. When it receives a notification, the client determines the TPM service event from the Interrupt Status register in the TPM service CRB interface.

In addition to signalling TPM service CRB events, the TPM service may also signal a call-back request through a client notification. The client's call-back enables coordination between the TPM service and the Primary Scheduler [3] when the TPM service processes commands asynchronously using a co-processor. 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 need to coordinate with the Primary Scheduler before processing TPM device interrupts.

The TPM service notifications signalling method is specified by Chapter 5 *Client notification interface*.

2.4 TPM service interface versioning

2.4.1 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 encodes 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 (>) the other 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 through an interface that is 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 (≥) the client's.

2.4.2 Extending the TPM service function interface

Future versions of this specification 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 implementation-defined extension of the TPM service function interface.

Table 2.1: 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 this specification.

Table 2.2: 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 this specification.

Chapter 3
Accessing the TPM service

Clients access the TPM service's CRBs [2][5] in memory, as specified by Section 3.2 *Accessing the TPM service CRBs*. The TPM service functions are accessed through FF-A direct messages and return FF-A direct responses, as specified by Section 3.3 *Accessing the TPM service functions*.

3.1 Discovery of the TPM service

OS-level discovery of TPM service presence may occur through a mechanism unrelated to the FF-A, for example, through the Start Method parameter of the TPM2 ACPI table [1]. However, clients are required to discover additional TPM service information through FF-A, as this section describes.

Clients must know the TPM service's FF-A partition ID in order to transmit TPM commands to it, or to access any of its functions. The TPM service's FF-A partition ID may be discovered through an FFA_PARTITION_INFO_GET call that provides the TPM service's UUID as argument.

TPM service UUID: 17b862a4-1806-4faf-86b3-089a58353861

Under FF-A v1.2 or later, the message addressing mode required by the TPM service must be determined at the time of service discovery; it is a TPM service implementation choice whether the service is addressed messages by only the FF-A partition ID through FFA_MSG_SEND_DIRECT_REQ, or by both the FF-A partition ID and the service UUID through FFA_MSG_SEND_DIRECT_REQ2.

3.2 Accessing the TPM service CRBs

Discovery of the CRB localities

High-privilege firmware defines platform-dependent CRB localities' physical addresses (PAs) such that they are known ahead of run-time, as specified in Section 2.1 *TPM service command-response buffers*. This means that the CRB localities may generally be discovered ahead of run-time.

The mechanism behind other firmware's discovery of the CRB localities is IMPLEMENTATION DEFINED. Apart from such IMPLEMENTATION DEFINED internal discovery mechanisms, firmware must continue to support its external discovery interfaces, for example the standard OS-facing TPM2 ACPI table or the Device Tree.

Sharing and accessing the CRBs

Client FF-A partitions must know the PA spaces and the PAs of the CRB localities ahead of run-time, to declare them as memory regions in their FF-A partition manifest so that the FF-A partition manager ensures the access. Also the TPM service must declare the CRB localities as memory regions in its FF-A partition manifest, for this purpose.

The clients and the TPM service must access the CRBs with the following memory attributes:

- the *Device* memory type; and
- the *nGnRnE* device memory attributes.

Note that Device-nGnRnE are also the default memory access attributes when address translation is disabled.

The clients and the TPM service must access the CRB localities in the correct PA space, as known ahead of run-time or implied through the discovery mechanism.

3.3 Accessing the TPM service functions

This section specifies the Arm-SMC template that clients must use to access the TPM service's functions. 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 is an FF-A direct message response from the TPM service. The TPM service must send the FF-A direct message response back to the client using the SMC response template specified by Table 3.3.

If the FF-A direct message delivery fails or its processing stalls, the SMC response is an error status returned by the FF-A framework. If the FF-A direct message processing is paused by the TPM service, the SMC response is an FF-A status returned by the TPM service. In either case, the SMC response is of the form shown in Table 3.4 and is specified in the FF-A specification [3].

Under run-time framework version FF-A v1.2 or later, the TPM service discovery may indicate that the service must be addressed by the service UUID in addition to the FF-A partition ID. For such TPM services, clients must access the TPM service functions as specified in When the TPM service functions must be accessed through FFA_MSG_SEND_DIRECT_REQ2.

Under run-time framework version prior to FF-A v1.2, or under a later framework version when the TPM service is addressed only by the FF-A partition ID, clients must access the TPM service functions as specified by Table 3.2.

Table 3.2: 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]: client's FF-A ID; variable, assigned by FF-A, the client can obtain it through FFA_ID_GET. Bit[15:0]: TPM service's runtime FF-A ID; variable, Section 3.1 *Discovery of the TPM service* 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 6 *TPM service function ABI*.

Table 3.3: 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]: client's FF-A ID, as received in the FFA_MSG_SEND_ _DIRECT_REQ.
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 6 *TPM service function ABI*.

Table 3.4: SMC response form for delivery error or incomplete processing of FF-A direct message.

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 through FFA_RUN; or For function calls that the TPM service may pause as per function return parameter specification in Chapter 6 *TPM service function ABI*: • since FF-A v1.2, `0x8400006c`, that is, ID of FFA_YIELD – message processing has been paused by the TPM service, and must be resumed through FFA_RUN.
w1 - w7	FF-A function status details.	Available in the FF-A specification [3].

When the TPM service functions must be accessed through DIRECT_REQ2

Accessing the TPM service functions through FFA_MSG_SEND_DIRECT_REQ2 requires the TPM service UUID as argument, in addition to the FF-A partition ID. The function access is still an FF-A direct message exchange as described in the previous section.

In this TPM service addressing mode, to send an FF-A direct message that calls a TPM service function, the client must use the SMC template specified by Table 3.5. Both the client and the service must be in the AArch64 execution mode for the SMC, in order to access the 64-bit parameter registers.

The SMC response when the FF-A message delivery fails or its processing stalls is an FF-A status of the form shown in Table 3.4, and is specified by the FF-A specification [3].

If the FF-A message delivery succeeds, the TPM service must respond to the function call using the SMC template specified by Table 3.6.

Table 3.5: SMC template for TPM service function calls through FFA_MSG_SEND_DIRECT_REQ2.

Register	Argument	Value
w0	FF-A function ID.	`0xc400008d`, that is, ID of FFA_MSG_SEND_DIRECT_REQ2.
w1	Sender and receiver FF-A partition IDs.	Bit[31:16]: client's FF-A ID; variable, assigned by FF-A, the client can obtain it through FFA_ID_GET. Bit[15:0]: TPM service's runtime FF-A ID; variable, Section 3.1 *Discovery of the TPM service* 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 [7]. Bits[31:0]: `0x17b862a4`. Bits[47:32]: `0x1806`. Bits[63:48]: `0x4faf`.
x3	TPM service UUID Hi part.	The TPM service UUID fields spanning bytes 8 - 15 specified by RFC 4122 [7]. Bits[15:0]: `0x86b3`. Bits[63:16]: `0x89a58353861`.
w4 - w7	TPM service function params.	Specified by Chapter 6 *TPM service function ABI*.

Register	Argument	Value
x8 - x17	Reserved for future ABI versions.	0

Table 3.6: SMC respone template for TPM service function calls through FFA_MSG_SEND_DIRECT_REQ2.

Register	Argument	Value
w0	FF-A function ID.	`0xc400008e`, that is, ID of FFA_MSG_SEND_DIRECT_RESP2.
w1	Sender and receiver FF-A partition IDs.	Bit[31:16]: TPM service's runtime FF-A ID. Bit[15:0]: client's FF-A ID, as received in the FFA_MSG_SEND_ _DIRECT_REQ2.
x2 - x3	Reserved for FF-A specification.	0
w4 - w7	TPM service function status parameters.	Specified by Chapter 6 *TPM service function ABI*.
x8 - x17	Reserved for future ABI versions.	0

Chapter 4
TPM service CRB interface

This chapter specifies the address-mapped *TPM service Command Response Buffer interface* (TPM service CRB interface), and the client-service behaviour required by this interface. The high-level command-response buffers feature of the TPM service is described in Section 2.1 *TPM service command-response buffers*.

4.1 CRB interface structure and general operation

The TPM service CRB interface must be in accordance with the TPM2 CRB interface specified by TCG [2]. The TPM service CRB interface consists of five CRB locality interfaces CRB Locality 0 - 4, one for each TPM service locality. A CRB locality interface consists of the Locality Support area, the Control Area and the Buffer area. The TCG specification takes precedence with respect to the structure and function of these interface areas. Usage of the CRB interface must be in accordance with the TPM2 CRB protocol implied from the TCG specifications [2][5].

CRB interface layout in memory

The TPM service CRB interface resides in RAM. The TPM service must provide one CRB structure in a fixed RAM region, for each TPM service locality; the fixed RAM regions need not form a contiguous region.

Firmware implementation must fix the RAM region for each CRB structure within the platform memory map, so that the CRB localities' physical addresses are known ahead of run-time. This is required for the TPM service FF-A partition and client FF-A partitions to declare access requirements to the CRB structures, for the FF-A framework to ensure run-time access.

Firmware implementation must ensure that the TPM service's CRB Locality 4 is protected from untrusted software entities such as the OS [2]. The way in which the TPM service meets this TCG requirement [2] is IMPLEMENTATION DEFINED. Firmware implementation may, for example, leverage TrustZone memory protection by laying the TPM service's CRB Locality 4 in a Secure memory region, if the TPM service is implemented in an FF-A Partition in the Secure World.

The remainder of this section specifies the CRB locality interface structure and the particulars of CRB interface operation in RAM.

CRB locality interface structure

The CRB structure must consist of the standard Locality Support area, the Control Area and the Buffer area [2], and must be arranged in this order in a contiguous region of RAM.

The Locality Support area differs at CRB Locality 4 in comparison with the other localities CRB Locality 0 - 3. Specifically, CRB Locality 4 must contain the standard Locality Control Register for Locality 4, whereas CRB Locality 0 - 3 must contain the Locality Control Register for Localities 0 - 3.

Accessing the CRB interface

Clients discover and access the CRB interface as specified by Section 3.2 *Accessing the TPM service CRBs*.

Client interaction with the CRB interface

Clients must follow the TPM2 CRB protocol in their interaction with the CRB interface [2][5]. A client's update of the TPM service CRB interface is detected by the TPM service only when the client invokes the FF-A TPM Start Method – clients must invoke the *start* notifier function after updating the CRB interface. The FF-A TPM Start Method invocation is required since the CRB interface is regular RAM.

Clients should ensure that their CRB interface updates can be observed only in the right order by the TPM service, which may observe them from a different CPU.

Service interaction with the CRB interface

The service must maintain the CRB interface state in accordance with the TPM2 CRB protocol [2][5]. The TPM service must not rely on the CRB interface to store internal state, as the CRB interface is regular RAM which can be modified by clients.

The service should ensure that its updates to the CRB interface can be observed only in the right order by the client, which may observe them from a different CPU.

4.2 CRB interface operation on a DRTM event

A TPM DRTM event occurs when a client completes a DRTM sequence at CRB Locality 4, as specified by TCG [2]. Generally, a system-wide DRTM event signifies establishment of trustworthy system state.

As part of processing a DRTM sequence, the TPM service must synchronise the CRB interface's state to the TPM service's internal valid state. This must restore the CRB interface from any malicious or erroneous alterations done by previous clients, so that subsequent clients need not trust them. This requirement is due to the TPM service CRB interface being regular RAM, which can be modified without TPM service's knowledge.

4.3 The `start` notifier function

The TPM service must support a `start` function that a client can use to notify the TPM service that the client has prepared a TPM command in a CRB locality, for processing. The `start` function ABI is specified in Chapter 6 *TPM service function ABI*.

A `start` function call means that the calling client allows the TPM service to process a TPM command immediately, within the 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. As per the standard TPM2 CRB protocol, the client must wait for this indication before reading the TPM command response.

The client may determine when the TPM command processing completes by polling the CRB Start bit, or through an FF-A Notification from the TPM service when the service supports it.[1]

Long-running synchronous command-processing within the TPM service

A `start` function call that returns successfully only when the TPM operation completes is said to trigger *synchronous command-processing* within the TPM service.

The TPM service must not allow synchronous command-processing to consume too many CPU cycles at a time, as that may affect the system's responsiveness or induce an operating system panic. The TPM service may use one of the following FF-A features to prevent synchronous command-processing from consuming too many CPU cycles at a time:

- 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 opportunistically. The client receives the FFA_YIELD request and is required to resume the TPM service using FFA_RUN. This usage of FFA_YIELD for pausing direct message processing is specified in FF-A v1.2 or later.

This document does not make TPM service function ABI provisions to support TPM service pauses through other methods. In particular, this document does not provide for pausing with an FF-A Managed Exit completed through an FF-A direct message response, in a TPM service implemented in a S-EL1 FF-A partition.

[1] The client notification interface is specified in Chapter 5 *Client notification interface*.

Chapter 5
Client notification interface

This chapter specifies the client–service interface and behaviour that realise the TPM service notifications signalling method. The high-level TPM service notifications feature is described in Section 2.3 *TPM service notifications*.

A *client notification* is an FF-A Notification sent by the TPM service to a client FF-A partition [3]. Client notification is a feature that the TPM service may optionally provide, and that clients may request when the TPM service provides it.

A client notification represents a TPM service event that occured in connection to a client's FF-A TPM Start Method invocation. The *TPM service events* are standard CRB state transitions that occur at the TPM service CRB interface. Thus, the client notification concept corresponds to the standard CRB interrupt [2].

An instance of the client notification mechanism is represented in the following figures. Figure 5.1 shows the sequence of events and FF-A messages for client registration for TPM service notifications. Figure 5.2 shows the sequence for receiving a notification when a TPM operation completes. The figures illustrate the end-to-end client notification mechanism only for completeness, client and service implementation typically does not involve FF-A framework implementation but rather it assumes and builds upon an FF-A framework implementation.

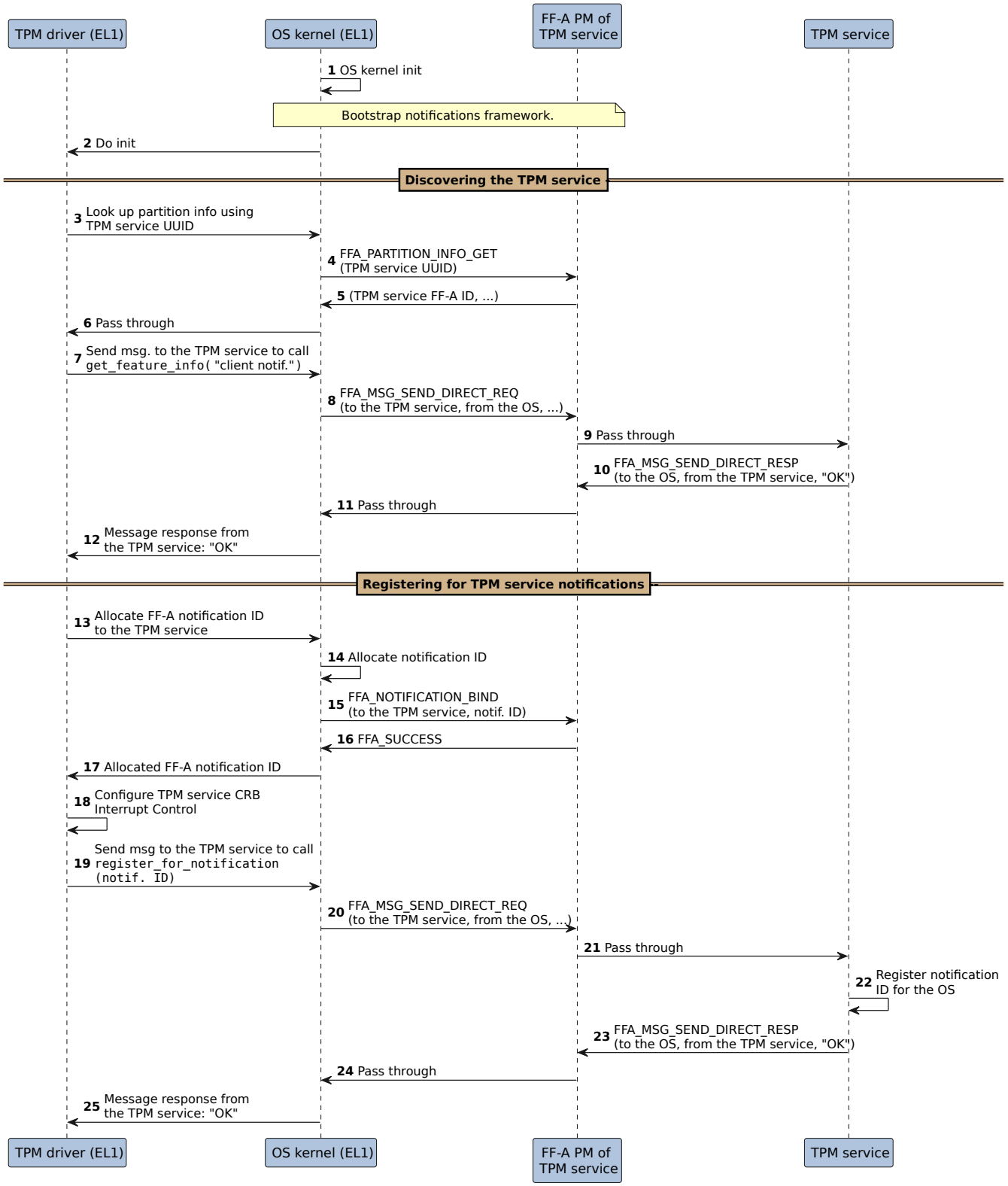

Figure 5.1: Illustration of a specific client and a TPM service setting up FF-A Notifications. (This client is formed of the TPM driver + the OS kernel).

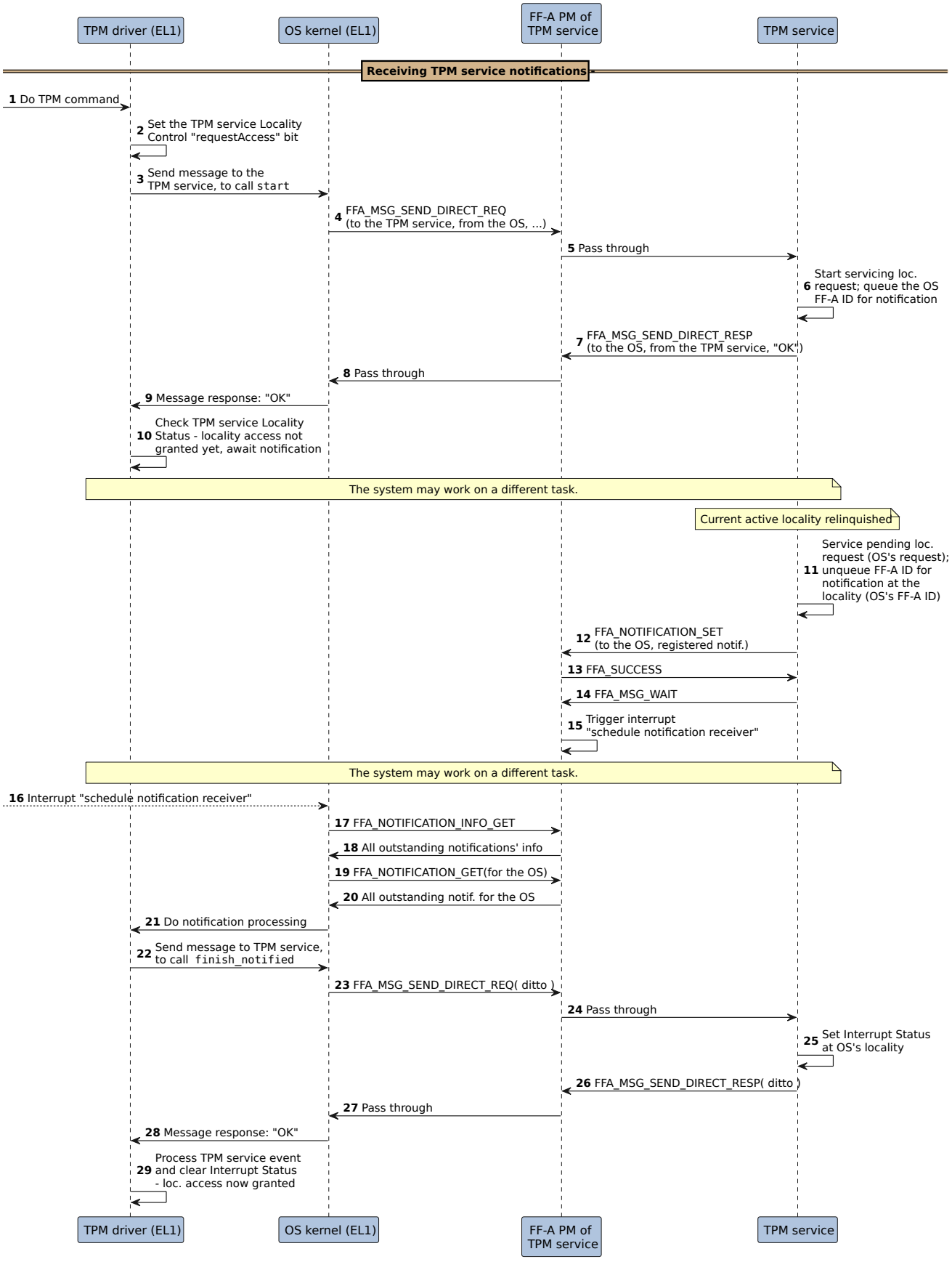

Figure 5.2: Illustration of a specific client and a TPM service using FF-A Notifications.

5.1 Client participation

This section specifies the client responsibilities of the notification interface.

The client may request notifications if the TPM service provides them. To determine whether the TPM service provides notifications, the client must call the get_feature_info function and query for the client notification feature.

The client may configure which TPM service events generate a notification through the Interrupt Control register in the TPM service CRB interface. The client may configure separately each CRB locality it uses.

To request TPM service notifications, the client must allocate an FF-A Notification ID in the FF-A framework and call the register_for_notification function with the resulting FF-A Notification info as arguments.

While the client is registered for notification, it may receive a TPM service notification in connection to a prior request made through `start` function invocation. On receipt of a notification from the TPM service, the client must call the finish_notified function (`finish`) to allow the TPM service to finalise processing the request.

When the `finish` function call returns, the client may determine which TPM service event occured through the Interrupt Status register in the CRB interface. The client may determine which TPM service event occured separately at each CRB locality it uses. After the client has dealt with the TPM service event, the client must clear the Interrupt Status register. By clearing the Interrupt Status register, the client ensures its content will be unambiguous if a future TPM service event occurs.

The client may receive a TPM service notification regardless of the Interrupt Control register configuration: even if the client has not enabled any TPM service event, the TPM service may still generate a notification to request a `finish` call-back. In this case, when the `finish` call returns, the client observes the Interrupt Status register bits as set even though the corresponding Interrupt Control bits are clear. The client must clear the Interrupt Status as normal, and may choose to take no further action.

To opt out from TPM service notifications once registered, the client may call the unregister_from_notification function. The client must unregister from notification when its runtime FF-A partition is destroyed.

5.2 TPM service participation

This section specifies the TPM service responsibilities of the notification interface.

When it begins processing a valid request from its CRB interface, the TPM service should record the client FF-A partition ID associated with the `start` function call, and the locality. The TPM service should keep the record while the request is being processed.

When a TPM service event occurs, during the processing of a request, the TPM service must send a notification to the client if both of the following conditions are met:

a. The client FF-A partition is registered for notification; and
b. The event is enabled in the Interrupt Control register at the CRB locality.

To send a notification to the client, the TPM service must use FFA_NOTIFICATION_SET with the client notification info received through `register_for_notification` for arguments. If NOTIFICATION_SET fails due to invalid client notification info, the TPM service may take IMPLEMENTATION DEFINED action.

After sending a notification to a client FF-A partition, the TPM service must not send any other notification to it, until after the client FF-A partition calls the finish_notified function (`finish`) and the function call returns.

After sending a notification to a client FF-A partition, the TPM service must service a `finish` function call from that client as follows:

- Set the bit that corresponds to the TPM service event in the Interrupt Status register at the CRB locality; and
- Finish processing the request.

The corresponding Interrupt Status register must not be clear when the TPM service returns from a `finish` function call successfully. The TPM service must not write to the Interrupt Status register at a CRB locality except while it is servicing a `finish` function call for that locality.

Additionally, during the processing of a request, the TPM service may send a notification to the client purely in order to receive a `finish` call-back, if:

a. the TPM service can finish processing the request and trigger a TPM service event in the `finish` call-back, and
b. the TPM service is allowed to send a notification to the client as that client does not already have an outstanding `finish` call-back from a previous notification, and
c. the client FF-A partition is registered for notification,

even if the TPM service event is not enabled in the Interrupt Control register. This allows the TPM service, under the specified conditions, to send a notification in anticipation of an indeterminate TPM service event, in order to continue processing the request and to complete it in the `finish` function call that follows.

Note

The TPM service may use indeterminate notifications to ensure coordination with the client's OS in the asynchronous command-processing, that is, in the case that the TPM service returns from the `start` function call successfully before completing the command processing.

Chapter 6
TPM service function ABI

This chapter specifies the programmatic interface for every TPM service function.

6.1 get_interface_version

Return the version of the interface that is available.

The interface in this version of the specification is **version (1, 0)**, as per Section 2.4.1 *Client-service compatibility*.

Table 6.1: SMC for the `get_interface_version` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 3.3 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	`0x0f00_0001`
w5 - w7	Reserved for future ABI versions.	0

Table 6.2: SMC response for the `get_interface_version` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 3.3 *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

6.2 get_feature_info

Return information on a given feature of the TPM service.

Table 6.3: SMC for the `get_feature_info` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 3.3 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	`0x0f00_0101`
w5	TPM service feature.	• `0xfea7_0000` – client notification of TPM service events through FF-A Notification. See Section 2.3 for a description of this feature.
w6 - w7	Reserved for future ABI versions.	0

Table 6.4: SMC response for the `get_feature_info` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 3.3 *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

6.3 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 6.5: The FF-A Start Method SMC.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 3.3 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	`0x0f00_0201`
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 6.6: The FF-A Start Method SMC response.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 3.3 *Accessing the TPM service functions*. The TPM service may pause this function call; the client must be prepared to resume it.
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

6.4 register_for_notification

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

Client prerequisite: before invoking this function, the calling FF-A partition must set up the FF-A Notification ID such that the TPM service can use it with FFA_NOTIFICATION_SET, as Figure 5.1 shows.

Table 6.7: SMC for the `register_for_notification` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 3.3 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	`0x0f00_0301`
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 [3] 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 6.8: SMC response for the `register_for_notification` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 3.3 *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` – the TPM service does not have the client notification feature; 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 up to a single FF-A partition is supported, only; or • `NOMEM` – the TPM service does not have memory available to perform the registration.
w5 - w7	Reserved for future ABI versions.	0

6.5 unregister_from_notification

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

Table 6.9: SMC for the `unregister_from_notification` function.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 3.3 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	`0x0f00_0401`
w5 - w7	Reserved for future ABI versions.	0

Table 6.10: SMC response for the `unregister_from_notification` function.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 3.3 *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` – the TPM service does not have the client notification feature; or • `DENIED` – the calling FF-A partition is not registered for notification.
w5 - w7	Reserved for future ABI versions.	0

6.6 finish_notified (finish)

Complete command or locality request processing for the calling FF-A partition; reveal the content of the respective TPM service CRB Interrupt Status register.

Client prerequisite: the client must be in receipt of an FF-A Notification from the TPM service before invoking this function – this function is invoked in response to a notification.

Table 6.11: SMC for the `finish_notified` function, also known as `finish`.

Register	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section 3.3 *Accessing the TPM service functions*.
w4	TPM service function ID arg.	`0x0f00_0501`
w5 - w7	Reserved for future ABI versions.	0

Table 6.12: SMC response for the `finish_notified` function, also known as `finish`.

Register	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section 3.3 *Accessing the TPM service functions*. The TPM service may pause this function call; the client must be prepared to resume it.
w4	TPM service function status.	Success status: • `OK` – command or locality request processing is complete, and Interrupt Status is up-to-date. Error status: • `NOTSUP` – the TPM service does not have the client notification feature; or • `DENIED` – the calling FF-A partition has no outstanding `finish` callback.

6.7 All function status

Table 6.13: TPM service function status codes.

Mnemonic	Description	Value
Error status:		
NOFUNC	No such function.	0x8e00_0001
NOTSUP	Function not supported.	0x8e00_0002
INVARG	Invalid argument.	0x8e00_0005
INV_CRB_CTRL_DATA	Invalid Command-Response Buffer control data.	0x8e00_0006
ALREADY	This request has already been carried out.	0x8e00_0009
DENIED	Operation not allowed in the current state.	0x8e00_000a
NOMEM	Not enough available memory.	0x8e00_000b
Success status:		
OK	Function succeeded.	0x0500_0001
OK_RESULTS_RETURNED	Function succeeded and results have been returned.	0x0500_0002

Chapter 7
Implementation considerations

7.1 Assumptions of this software architecture

The components that the TPM service uses to carry out its function are protected from entities other than the TPM service and its TCB.

In particular, if the TPM service carries out its function through a TPM device, this software architecture assumes that the TPM device is protected from entities other than the TPM service and its TCB.

If the TPM service carries out its function through a TPM device, it is controlled exclusively by the TPM service.

If the TPM service carries out its function through a TPM device, it implies that the TPM service features a TPM-device driver. By definition, a device driver assumes complete control over the device and does not allow other software interference with the device.

TCB components such as EL3 firmware also are assumed to access the TPM service rather than control the TPM device directly.

7.2 The Trusted Computing Base (TCB)

This section describes the TCB and helps illustrates how it varies depending on the TPM service and the system implementation.

The TCB of the TPM service

In general, the TPM service must trust the following components:

1. the FF-A Partition Manager;
2. the EL3 firmware;
3. the components that the TPM service uses to carry out its function, and the entities that have access to them.

For example, if

- the TPM service is implemented in a Secure World FF-A partition (an SP); and
- the TPM service carries out its function through a TPM device; and
- the TPM device can be accessed directly from the Normal World;

then the TPM service must trust Normal World components such as EL2 firmware, an EL2 hypervisor, or a baremetal OS kernel, in addition to the SPM and EL3 firmware. This example system implementation has a large TPM service TCB relative to another implementation where the TPM device were in the TrustZone Secure Physical Address Space, protected from the Normal World. All other aspects being equal, the latter example system implementation is more desirable than the former, since a smaller TCB is more desirable.

For another example, if

- the TPM service is implemented in an SP; and
- the TPM service carries out its function through a firmware TPM; and
- the firmware TPM depends on a Normal World driver for secure storage;

then the TPM service must trust Normal World components such as the OS kernel and/or an EL2 hypervisor, in addition to the SPM and EL3 firmware.

The TCB of client usage of the TPM service

In general, clients of the TPM service must trust the following components with the processing of their TPM commands:

1. the TPM service and the TPM service's TCB;
2. the entities that can access the TPM service CRB they are client to;
3. the entities that mediate access to the TPM service function interface.

For example, in the case of a Secure World FF-A partition (an SP) client that uses the TPM service's CRB Locality 0, if

- the CRB Locality 0 is in the Non-secure physical address space;

then the SP must trust Normal World components with the processing of its TPM command, in addition to trusting the SPM; this includes components such as the OS kernel and EL2 firmware.

Glossary

ABI application binary interface

ACPI Advanced Configuration and Power Interface

CRB Command-Response Buffer

DRTM dynamic Root of Trust for Measurement

FF-A Firmware Framework for Arm A-profile

nGnRnE non-Gathering, non-Reordering, No Early Write Acknowledgement

OS operating system

PM Partition Manager (FF-A partition manager)

S-EL1 Secure EL1 (Secure Exception Level 1)

SP Secure Partition (Secure World FF-A partition)

SPM Secure Partition Manager (Secure World FF-A partition manager)

TCB Trusted Computing Base

TCG Trusted Computing Group

TPM Trusted Platform Module

UUID Universally Unique Identifier

vCPU virtual CPU

Part A
Appendix

Chapter A1
Compatibility with the legacy Arm TPM Start Method

The legacy Arm TPM Start Method is one of the TPM Start Methods defined in the TPM2 ACPI specification [1]. It is an Arm SMC used in the CRB protocol for RAM-based CRBs, for a purpose similar to the Arm FF-A TPM Start Method specified by this document. It is identified through an IMPLEMENTATION DEFINED function ID. It is designed to be used by an OS kernel after discovering the function ID through the TPM2 ACPI table, or by Normal World firmware with built-in knowledge of the function ID or of a firmware-specific discovery mechanism.

The legacy Arm TPM Start Method may be added to the TPM service function interface through an IMPLEMENTATION DEFINED TPM service extension within the extension function ID range specified by Section 2.4.2 *Extending the TPM service function interface*. Firmware implementation may add compatibility with the legacy Arm TPM Start Method through a TPM service proxy that services the legacy Arm SMC by invoking the TPM service extension. The TPM service proxy must:

- have knowledge of the TPM service, gained as specified by Section 3.1 *Discovery of the TPM service*, or through an IMPLEMENTATION DEFINED mechanism; and
- invoke the extension TPM service function as specified by Section 3.3 *Accessing the TPM service functions*, except use the ERET conduit [3] rather than the SMC conduit if the proxy resides at EL3.

The caller of the Arm SMC must know or discover the IMPLEMENTATION DEFINED function ID of the Arm TPM Start Method, as before.

Supporting the legacy Arm TPM Start Method in this way, through the TPM service function interface, has the benefit of TPM firmware isolation in an FF-A partition and abstraction of the TPM implementation.