



# TPM Service Command Response Buffer Interface Over FF-A

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

*Copyright © 2025 Arm Limited or its affiliates. All rights reserved.*

# TPM Service Command Response Buffer Interface Over FF-A

## Release information

Date	Version	Changes
2025/Apr/07	v1.0 EAC	<ul style="list-style-type: none"><li>• Fixed the endianness of the TPM service UUID in the SMC template for DIRECT_REQ and DIRECT_REQ2</li><li>• Defined 64-bit function IDs in the SMC template</li><li>• Clarified that CRB sequence diagram assumes TPM is in Ready state and does not show Idle</li><li>• Clarified that TPM needs to maintain synchronization between it's internal state and CRB</li><li>• Updated versions of specifications in the References</li><li>• Clarified that a TPM service can support long running commands, because it can be configured to be preemptible and does not have to run with interrupts disabled</li></ul>
2024/Jan/16	v1.0 BET	<ul style="list-style-type: none"><li>• Editorial edits and clean-up</li><li>• Introduced encodings for FIDs and for all function status; also introduced the TPM service UUID;</li><li>• Completed the "Client notification interface", which facilitates the TPM service notifications feature;</li><li>• Introduced the new TPM service function "finish_notified()" used for client notification handling (also known as "finish()");</li><li>• Added the specification of the TPM service CRB interface and its usage;</li><li>• 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);</li><li>• Added information about the assumptions of this software architecture and the Trusted Computing Base through the chapter "Implementation considerations";</li><li>• Added information about compatibility with the legacy Arm TPM Start Method through a section in the Appendix.</li></ul>
2023/Jun/28	v1.0 ALP	<ul style="list-style-type: none"><li>• First release.</li></ul>

## Arm Non-Confidential Document License (“License”)

This License is a legal agreement between you and Arm Limited (“Arm”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License (“Document”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

“**Subsidiary**” means any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“Licensee”) is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

- (i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;
- (ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and
- (iii) sell, supply and distribute products which have been created under the License granted in (i) above.

**Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.**

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm’s view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party’s products or services within this document is not an express or implied approval or endorsement of the use thereof.

THE DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This License may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this License and any translation, the terms of the English version of this License shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. No license, express, implied or otherwise, is granted to Licensee under this License, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at <http://www.arm.com/company/policies/trademarks> for more information about Arm's trademarks.

The validity, construction and performance of this License shall be governed by English Law.

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

## Contents

# TPM Service Command Response Buffer Interface Over FF-A

	TPM Service Command Response Buffer Interface Over FF-A . . . . .	ii
	Release information . . . . .	ii
	Arm Non-Confidential Document License ("License") . . . . .	iii
<b>Preface</b>		
	Conventions . . . . .	viii
	Typographical conventions . . . . .	viii
	Numbers . . . . .	viii
	Additional reading . . . . .	ix
	Feedback . . . . .	x
	Feedback on this book . . . . .	x
<b>Chapter 1</b>	<b>Introduction</b>	
	1.1 Technical background . . . . .	13
	1.2 Overview of this specification . . . . .	14
<b>Chapter 2</b>	<b>The TPM service</b>	
	2.1 TPM service command-response buffers . . . . .	18
	2.2 TPM start method . . . . .	19
	2.3 TPM service notifications . . . . .	20
	2.4 TPM service interface versioning . . . . .	21
	2.4.1 Client-service compatibility . . . . .	21
	2.4.2 Extending the TPM service function interface . . . . .	21
<b>Chapter 3</b>	<b>Accessing the TPM service</b>	
	3.1 Discovery of the TPM service . . . . .	23
	3.2 Accessing the TPM service CRBs . . . . .	24
	3.3 Accessing the TPM service functions . . . . .	25
<b>Chapter 4</b>	<b>TPM service CRB interface</b>	
	4.1 CRB interface structure and general operation . . . . .	29
	4.2 CRB interface operation on a DRTM event . . . . .	30
	4.3 The <code>start</code> notifier function . . . . .	31
	Long-running synchronous command-processing within the TPM service . . . . .	31
<b>Chapter 5</b>	<b>Client notification interface</b>	
	5.1 Client participation . . . . .	35
	5.2 TPM service participation . . . . .	36
<b>Chapter 6</b>	<b>TPM service function ABI</b>	
	6.1 <code>get_interface_version</code> . . . . .	38
	6.2 <code>get_feature_info</code> . . . . .	39
	6.3 <code>start</code> . . . . .	40
	6.4 <code>register_for_notification</code> . . . . .	41
	6.5 <code>unregister_from_notification</code> . . . . .	42
	6.6 <code>finish_notified</code> (finish) . . . . .	43
	6.7 All function status . . . . .	44
<b>Chapter 7</b>	<b>Implementation considerations</b>	
	7.1 Assumptions of this software architecture . . . . .	45

7.2	The Trusted Computing Base (TCB) . . . . .	46
	The TCB of the TPM service . . . . .	46
	The TCB of client usage of the TPM service . . . . .	46

## Glossary

## Part A Appendix

### Chapter A1 Compatibility with the legacy Arm TPM Start Method

# 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 PC Client Platform TPM Profile Specification for TPM 2.0.* (version 1.05, revision 14) Trusted Computing Group.
- [2] *DEN0077A Arm Firmware Framework for Arm A-profile.* (version 1.3) Arm.
- [3] *Trusted Platform Module Library Specification.* (family “2.0,” level 00, revision 01.83) Trusted Computing Group.
- [4] *TCG ACPI Specification.* (version 1.4, revision 15) Trusted Computing Group.
- [5] *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 EAC).
- 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.

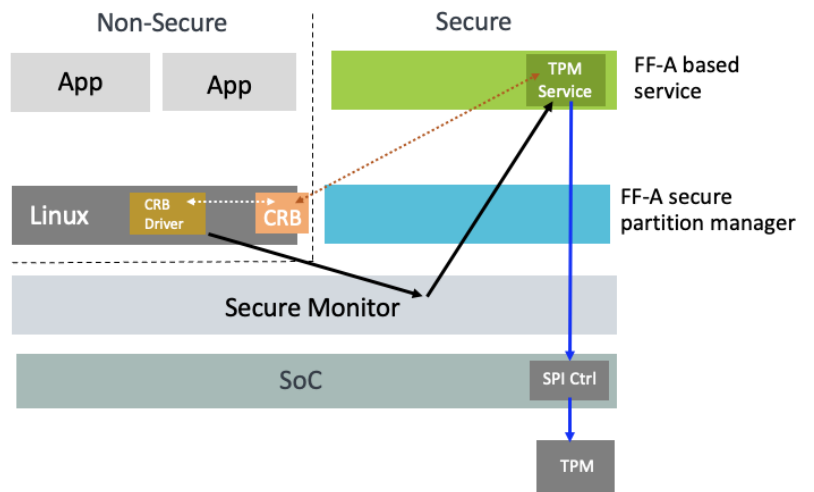
---

# Chapter 1

## Introduction

Arm platform implementors may wish to implement a TPM where access to the TPM is mediated through firmware in TrustZone using a TCG-compliant TPM interface. See example in [Figure 1.1](#) below. This approach provides several benefits:

- For discrete TPM chips, abstracts bus specific details such as bus type (SPI, I2C), bus controller, and chip selects away from the OS. This allows a single TPM driver in the OS to support many implementations.
- Allows firmware to deal with platform limitations, such the TPM and flash devices sharing a SPI bus where access must be arbitrated
- Allows TPM implementations such as a firmware TPM, where the underlying implementation of the TPM is transparent to the OS
- Allows protection of Locality 4 from the Normal World



**Figure 1.1: TPM access mediated through firmware**

A TPM client and a firmware-based TPM service interact using a TCG compliant Command Response Buffer (CRB) interface in shared memory. The architecture and semantics of the CRB are defined in the TCG PC-Client-Platform TPM Profile (PTP) specification [1]. This document assumes a TPM service based on the Arm Firmware Framework for A-profile (FF-A) [2]. With this approach the CRB shared memory region is implemented in RAM, and a signaling mechanism is needed to enable the TPM client and TPM service to notify each other when updates to the CRB occur.

This document specifies:

- An FF-A based ABI to allow a TPM client to send notifications to a TPM service and receive notifications from a TPM service
- Considerations for implementing an FF-A based TPM service interface that is compliant with the CRB defined in the TCG PTP
- Considerations for implementing a TPM client interacting with an FF-A based TPM service compliant to this document

## 1.1 Technical background

### The TPM interface for software

A Trusted Platform Module or TPM is a system component that can be used to help establish, enforce, and attest to the integrity of a system. A standard interface to a TPM defined by TCG is the CRB interface [1].

The following is the general usage model of the CRB for sending TPM commands:

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. If a CRB is implemented in RAM an out-of-band signaling mechanism is required to notify the TPM that the control bit has been updated.
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
7. Software reads the TPM response from the Response Buffer

The complete TPM interface consists of multiple instances of the CRB 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. <sup>1</sup>

Figure 1.2 shows the relationship between TPM localities and software components that map and use a CRB corresponding to those localities.

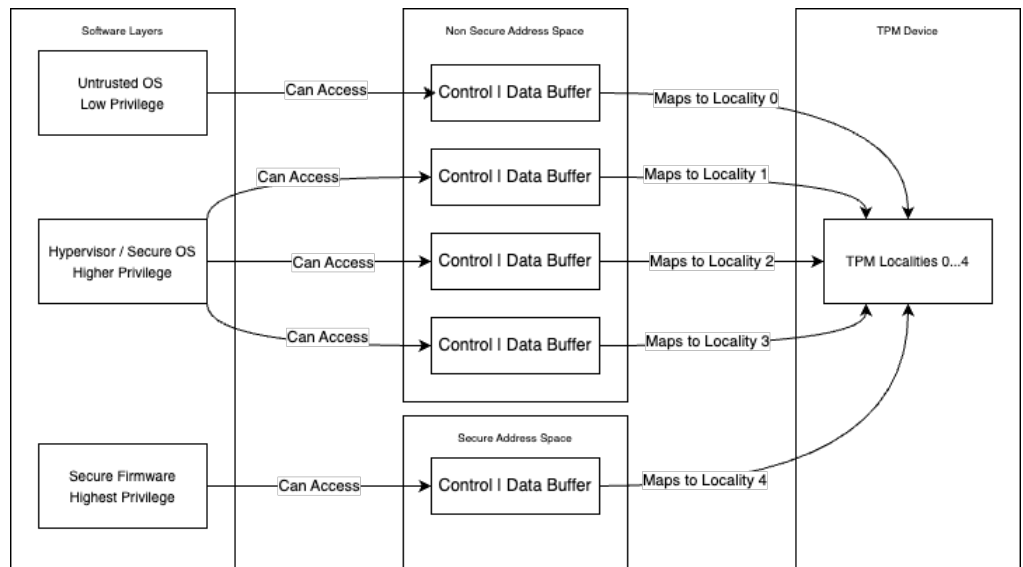


Figure 1.2: Relationship of TPM localities to CRBs and software components

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

## 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 a TPM service is to implement a TCG-compliant TPM [3] which receives TPM commands and returns TPM responses.

The features that the TPM service must have are described in [Chapter 2](#).

- [Section 2.1](#) describes the TPM service CRB interface.
- [Section 2.2](#) describes the FF-A message that a TPM client uses to notify the TPM service of updates to the CRB.
- [Section 2.3](#) describes FF-A notifications that the TPM service can use to notify a TPM client of events.
- 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 clients access the TPM service CRBs and how to use FF-A direct messages to access the TPM service's 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.

[Chapter 6](#) specifies the complete TPM service function interface.

[Chapter 7](#) provides TPM service implementation considerations.

An overview of the TPM2 CRB protocol over FF-A is depicted in [Figure 1.3](#), below.

Note: [Figure 1.3](#) assumes the TPM is in the “Ready” state and does not depict the “Idle” state.

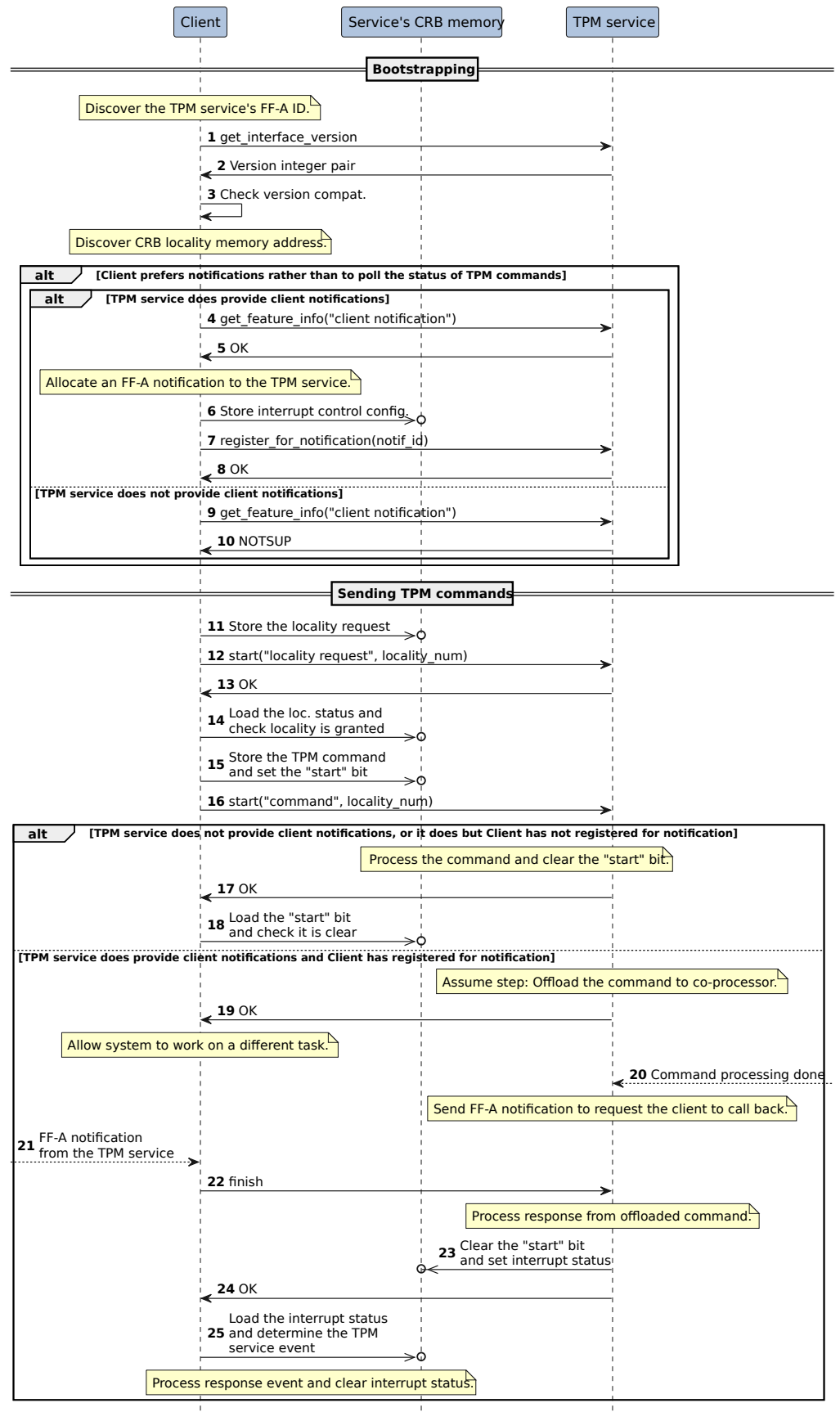


Figure 1.3: Overview of the TPM2 CRB protocol over FF-A.

## Chapter 2

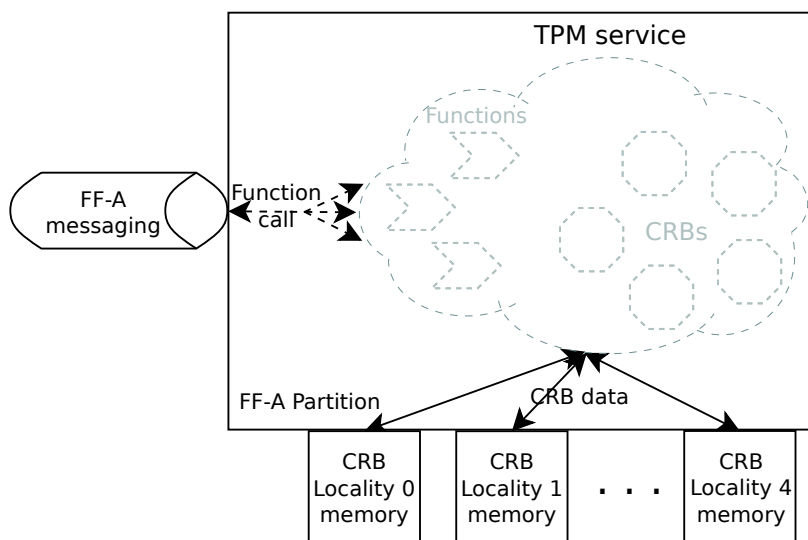
# The TPM service

An FF-A based TPM service is firmware contained within an FF-A partition and is responsible for receiving TPM commands and returning TPM responses. TPM clients interact with the service through two distinct interfaces:

- A TCG compliant Command Response Buffer (CRB) interface in shared memory. The architecture and semantics of the CRB are defined in the TCG PC-Client-Platform TPM Profile (PTP) specification [1].
- A set of service functions accessed through the FF-A direct messaging ABI.

[Figure 2.1](#) illustrates the TPM service interface.





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

The TPM service carries out its functions in an IMPLEMENTATION DEFINED way. TPM service implementations might consist of a front-end tasked with read-writing the TPM service access interface and with driving a back-end, 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 run-time, 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 TCG-compliant Command-Response Buffer interface [1].

The TPM service provides a CRB structure in a fixed RAM region for each TPM locality. Clients send TPM commands and the TPM service returns TPM responses by accessing the CRB structures in memory in accordance with the standard defined in the TCG PTP [1]. A summary of the CRB protocol is provided in the [Introduction](#).

A firmware implementation allocates the RAM region for each CRB structure using knowledge of the platform memory map. This allocation of physical RAM regions will typically be done statically prior to run-time. Knowledge of the physical addresses of the CRB is required for the TPM service FF-A partition and client FF-A partitions to declare access requirements to the RAM regions, enabling 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

When the TPM CRB is implemented in RAM a signaling mechanism is needed for clients to notify the TPM service of updates made to fields in the CRB. This signaling mechanism is called the “Start Method” in the TCG ACPI specification [4]. 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.

## 2.3 TPM service notifications

TPM service notifications are an FF-A signaling 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 [1] in the CRB. When it receives a notification, the client determines the TPM service event from the Interrupt Status register in the CRB.

In addition to signaling 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 [2] when the TPM service does asynchronous processing. For example, a Secure World TPM service that processes commands by offloading them to a TPM device with interrupt support for signaling completion of processing may need to coordinate with the Primary Scheduler before processing TPM device interrupts.

The TPM service notifications signaling 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' version and the second is called the 'minor' version. The interface in this version of the specification is **version (1, 0)**.

The following rules apply to the version numbering:

- Different major revision values indicate possibly incompatible functions. A newer major revision might:
  - Introduce new functions
  - Deprecate older functions
  - Change behavior of existing functions
- For two revisions, A and B, for which the major revision values are identical, if the minor revision value of revision B is greater than the minor revision value of revision A, then every function in revision A must work in a compatible way with revision B. However, revision B can have a higher function count than revision A.

Hence the TPM service is compatible with a client if and only if its major version is equal to ( $=$ ) the client's and its minor version is greater or equal to ( $\geq$ ) 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
<code>0x1fxx_xxxx</code>	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
<code>0x15xx_xxxx</code>	IMPLEMENTATION DEFINED function success status.
<code>0x9exx_xxxx</code>	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 [1] 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 [4]. 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.

---

<b>TPM service UUID:</b>	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 [2].

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.	ID of FFA_MSG_SEND_DIRECT_REQ <ul style="list-style-type: none"> <li>• 0x8400006f (32-bit version)</li> <li>• 0xc400006f (64-bit version)</li> </ul>
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 run-time FF-A ID; variable, Section <a href="#">3.1 Discovery of the TPM service</a>   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 <a href="#">Chapter 6 TPM service function ABI</a> .

**Table 3.3: SMC response template for TPM service function calls.**

Register	Return argument	Value
w0	FF-A function status.	ID of FFA_MSG_SEND_DIRECT_RESP – the message was delivered successfully and a message response has been returned. <ul style="list-style-type: none"> <li>• 0x84000070 (32-bit version)</li> <li>• 0xc4000070 (64-bit version)</li> </ul>
w1	Sender and receiver FF-A partition IDs.	Bit[31:16]: TPM service's run-time 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

Register	Return argument	Value
w4 - w7	TPM service function status parameters.	Specified by <a href="#">Chapter 6 TPM service function ABI</a> .

**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.	<ul style="list-style-type: none"> <li>0x84000060, that is, ID of FFA_ERROR – message has failed to reach the TPM service; or</li> <li>0x84000062, that is, ID of FFA_INTERRUPT – message processing has been interrupted and must be resumed through FFA_RUN; or</li> </ul> <p>For function calls that the TPM service may pause as per function return parameter specification in <a href="#">Chapter 6 TPM service function ABI</a>:</p> <ul style="list-style-type: none"> <li>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.</li> </ul>
w1 - w7	FF-A function status details.	Available in the FF-A specification [2].

## 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 [2].

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, <i>l</i> assigned by FF-A, the client can obtain <i>l</i> it <i>l</i> through FFA_ID_GET. <i>l</i> Bit[15:0]: TPM service's run-time FF-A ID; variable, Section <a href="#">3.1 Discovery of the TPM service</a> <i>l</i> describes how it may be discovered. <i>l</i>
x2	TPM service UUID Lo part.	<p>The TPM service UUID fields spanning bytes 0 - 7 specified by RFC 4122 [5].</p> <p>Bits[31:0]: 0xa462b817.</p> <p>Bits[47:32]: 0x0618.</p> <p>Bits[63:48]: 0xaf4f.</p>

Re gist er	Argument	Value
x3	TPM service UUID Hi part.	The TPM service UUID fields spanning bytes 8 - 15 specified by RFC 4122 [5]. Bits[15:0]: 0xb386. Bits[63:16]: 0x613835589a08.
w4 - w7	TPM service function params.	Specified by <a href="#">Chapter 6 TPM service function ABI</a> .
x8 - x17	Reserved for future ABI versions.	0

**Table 3.6: SMC response template for TPM service function calls through FFA\_MSG\_SEND\_DIRECT\_REQ2.**

Re gist er	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 run-time FF-A ID.    Bit[15:0]: client's FF-A ID, as l 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 <a href="#">Chapter 6 TPM service function ABI</a> .
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 definition of the CRB in the TCG PTP specification [1]. The TPM service CRB interface consists of five CRB locality interfaces CRB Locality 0 - 4, one for each TPM service locality. The TCG PTP defines the structure and function of these interface.

### CRB interface layout in memory

The TPM service CRB interface resides in RAM. For each locality, the TPM service must provide one CRB structure in a fixed RAM region. For localities 0-3 the fixed RAM regions should form a contiguous region.

Firmware 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 and for the FF-A framework to ensure run-time access.

Firmware must ensure that the TPM service's CRB Locality 4 is protected from untrusted software entities such as the OS [1]. The way in which the TPM service meets this TCG requirement [1] is IMPLEMENTATION DEFINED. Firmware implementation may, for example, leverage TrustZone memory protection by allocating 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 interface structure

The CRB structure must consist of all the fields defined in the TCG PTP [1]—locality control, CRB control, data buffer. Note, the definition of Locality 4 CRB control fields is different than 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 semantics of the CRB [1] in their interaction with the fields in the CRB interface. 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.

When Clients make locality requests such as 'request access' and 'relinquish' the client updates the locality control register and then invokes the *start notifier function* with the locality qualifier.

### Service interaction with the CRB interface

The service must maintain the CRB interface state in accordance with the TPM2 CRB protocol [1].

The TPM service must not rely on the CRB interface to store internal state. 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 in RAM, which can be modified without TPM service's knowledge.

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 the TCG PTP [1]. Generally, a system-wide DRTM event signifies establishment of trustworthy system state.

### 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 made a change to a field in the CRB and it is ready 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 the CRB update immediately, within the FF-A direct message processing. However, for TPM commands 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 Start bit in the TPM\_CRB\_CTRL\_START\_x register. 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 Start bit or through an FF-A Notification from the TPM service when the service supports it. <sup>1</sup>

### 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 secure partition can be configured to be preemptible if it is not necessary to run with interrupts disabled. The TPM service allows the FF-A framework to pause the command processing directly and to yield to the relevant FF-A partition. The TPM client may receive an FFA\_INTERRUPT return status from the 'start' function, which requires the client to resume the TPM service using FFA\_RUN;
- Voluntary pauses. The TPM service pauses using FFA\_YIELD opportunistically. The TPM client receives the FFA\_YIELD return status from the 'start' function and is required to resume the TPM service using FFA\_RUN. This requires the TPM service to determine when to yield. 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.

<sup>1</sup>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 of TPM service notifications. A high-level description of the TPM service notifications feature is 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 [2]. 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 interrupt event that occurred in connection to a client's FF-A TPM Start Method invocation. The *TPM interrupt events* are standard CRB state transitions that occur at the TPM service CRB interface. Thus, the client notification concept corresponds to CRB interrupts as defined in the TCG PTP [1].

An example 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.



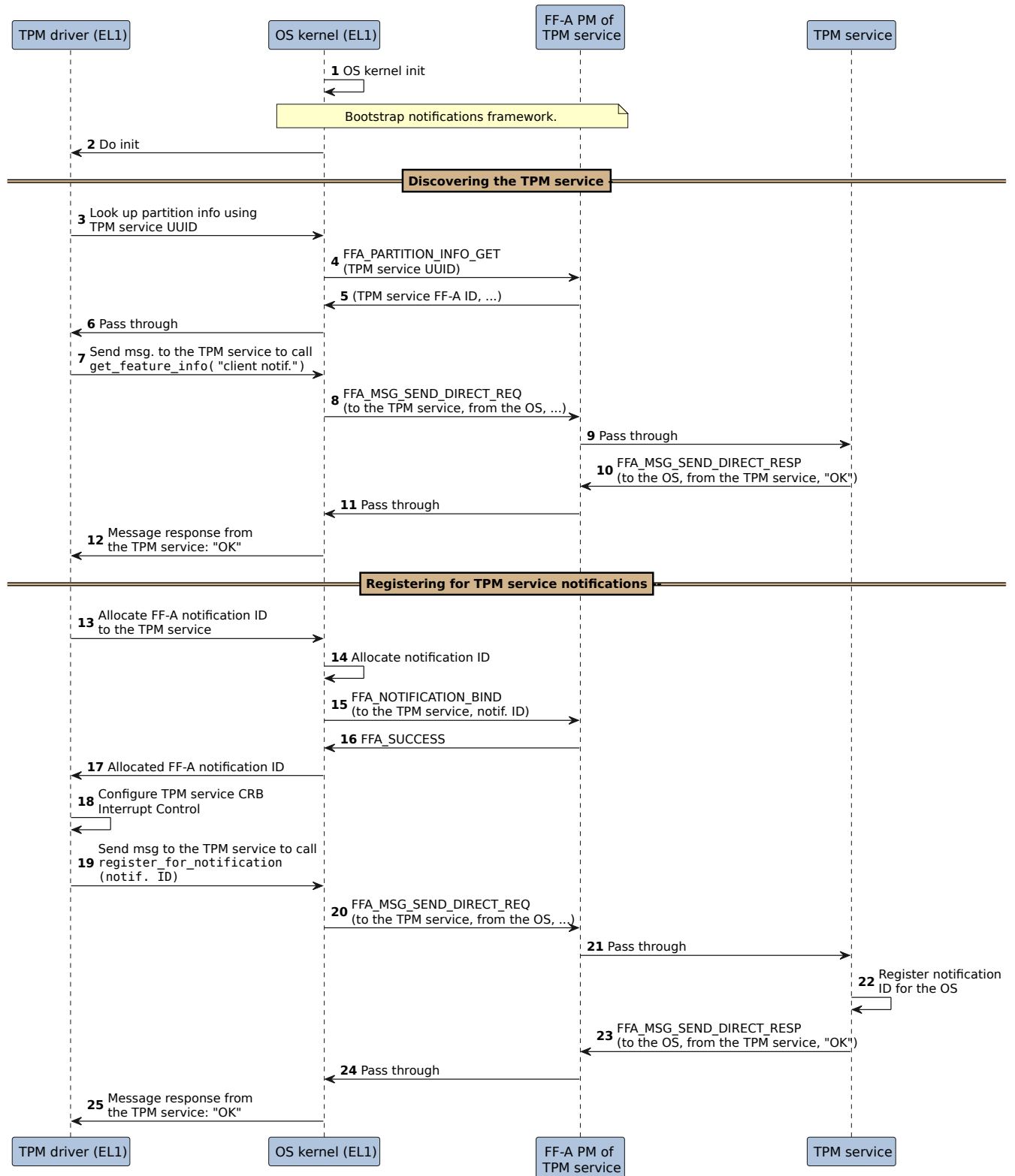


Figure 5.1: Illustration of a Linux client and a TPM service setting up FF-A Notifications

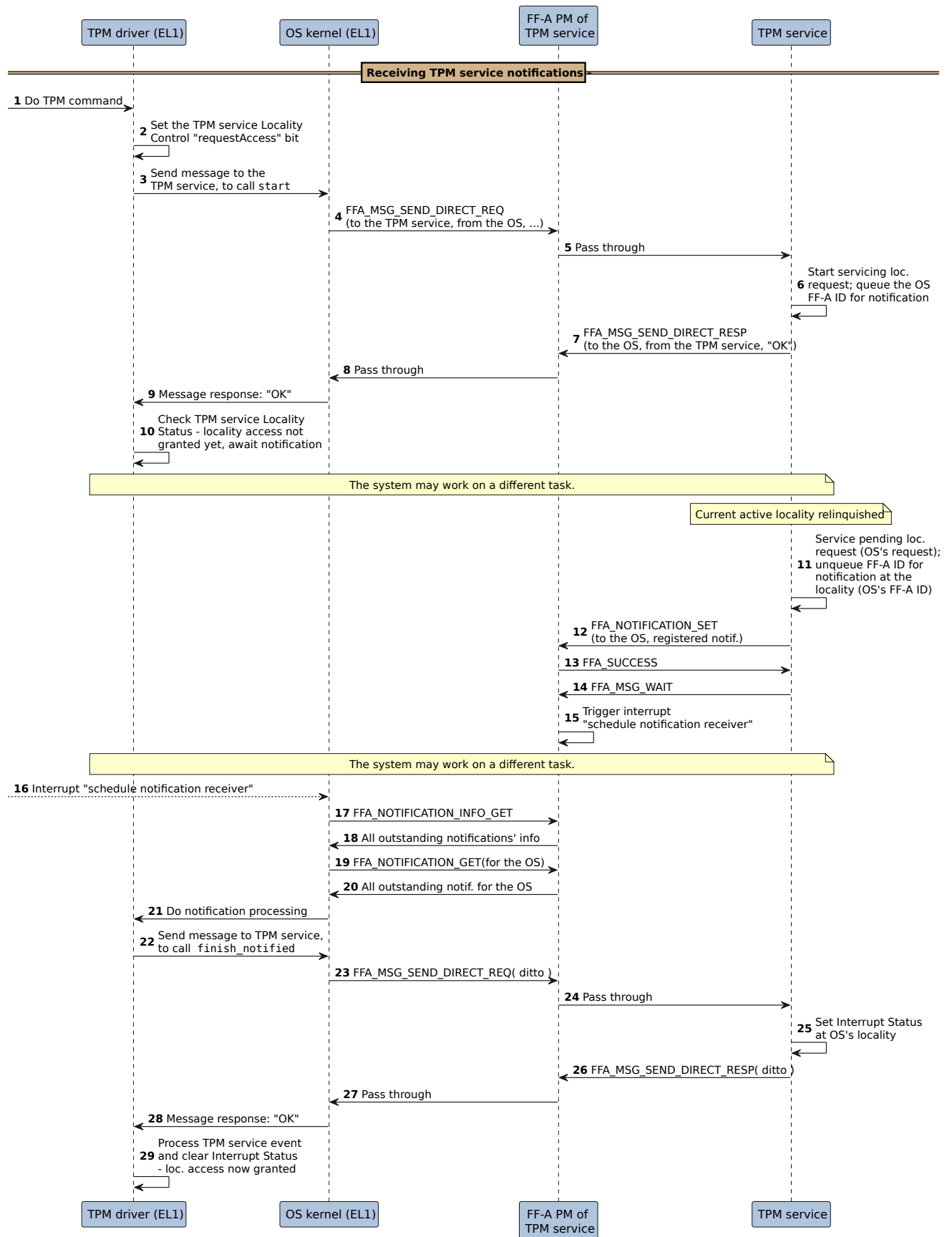


Figure 5.2: Illustration of a Linux client and a TPM service using FF-A Notifications.

## 5.1 Client participation

This section specifies the client responsibilities when using 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 interrupt 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 interrupt event occurred through the Interrupt Status register in the CRB interface. The client may determine which TPM interrupt event occurred separately at each CRB locality it uses. After the client has dealt with the TPM interrupt 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 interrupt 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 interrupt 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 run-time FF-A partition is destroyed.

## 5.2 TPM service participation

This section specifies the TPM service responsibilities when implementing 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 interrupt 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 interrupt 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 set the CRB Interrupt Status register 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

A notification may be sent even if the TPM interrupt 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 interrupt 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 <a href="#">3.3 Accessing the TPM service functions</a> .
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 <a href="#">3.3 Accessing the TPM service functions</a> .
w4	TPM service function status.	<ul style="list-style-type: none"><li>OK_RESULTS_RETURNED – the version of the interface has been returned.</li></ul>
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.**

Re gist er	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section <a href="#">3.3 Accessing the TPM service functions</a> .
w4	TPM service function ID arg.	0x0f00_0101
w5	TPM service feature.	<ul style="list-style-type: none"><li>• 0xfea7_0000 – client notification of TPM service events through FF-A Notification. See <a href="#">Section 2.3</a> for a description of this feature.</li></ul>
w6 - w7	Reserved for future ABI versions.	0

**Table 6.4: SMC response for the *get\_feature\_info* function.**

Re gist er	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section <a href="#">3.3 Accessing the TPM service functions</a> .
w4	TPM service feature status.	Success status: <ul style="list-style-type: none"><li>• OK – the feature is present; or</li></ul> Error status: <ul style="list-style-type: none"><li>• INVARG – the given feature ID is not valid; or</li><li>• NOTSUP – the feature is not present.</li></ul>
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.**

Re gist er	Parameter	Values
w0 - w3	FF-A message transport args.	Specified by Section <a href="#">3.3 Accessing the TPM service functions</a> .
w4	TPM service function ID arg.	0x0f00_0201
w5	TPM service <code>start</code> function qualifier.	Bits[31:8]: 0. Bits[7:0] command type: <ul style="list-style-type: none"> <li>0 – the function call notifies the TPM service that a command is ready to be processed; or</li> <li>1 – the function call notifies the TPM service that a locality request is ready to be processed.</li> </ul>
w6	TPM service <code>start</code> function locality qualifier.	Bits[31:8]: 0. Bits[7:0] TPM CRB locality: One of 0 .. 4: <ul style="list-style-type: none"> <li>The locality where the command is, if the function qualifier argument “command type” represents 0; or</li> <li>The locality where the locality request is, if the function qualifier argument “command type” represents 1.</li> </ul>
w7	Reserved for future ABI versions.	0

**Table 6.6: The FF-A Start Method SMC response.**

Re gist er	Return parameter	Values
w0 - w3	FF-A message transport status.	Specified by Section <a href="#">3.3 Accessing the TPM service functions</a> . The TPM service may pause this function call; the client must be prepared to resume it.
w4	TPM service function status.	Success status: <ul style="list-style-type: none"> <li>OK – the TPM service has been notified successfully; or</li> </ul> Error status: <ul style="list-style-type: none"> <li>INVARG – one or more arguments are not valid; or</li> <li>INV_CRB_CTRL_DATA – CRB control data or locality control data at the given TPM locality is not valid; or</li> <li>DENIED – firmware has previously disabled locality requests and command processing at the given locality.</li> </ul>
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 <a href="#">3.3 Accessing the TPM service functions</a> .
w4	TPM service function ID arg.	0x0f00_0301
w5	FF-A Notification type.	Bit[16] notification type qualifier: <ul style="list-style-type: none"> <li>• 0 – notification ID identifies a global FF-A Notification; or</li> <li>• 1 – notification ID identifies a per-vCPU FF-A Notification.</li> </ul> Bits[15:0] destination FF-A vCPU ID: <ul style="list-style-type: none"> <li>• The FF-A vCPU ID that should be passed to FFA_NOTIFICATION_SET [2] when a notification is sent, if Bit[16] is 1; or</li> <li>• 0, if Bit[16] is 0;</li> </ul>
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 <a href="#">3.3 Accessing the TPM service functions</a> .
w4	TPM service function status.	Success status: <ul style="list-style-type: none"> <li>• OK – the calling FF-A partition has been registered for notification.</li> </ul> Error status: <ul style="list-style-type: none"> <li>• INVARG – one or more arguments are not valid; or</li> <li>• NOTSUP – the TPM service does not have the client notification feature; or</li> <li>• ALREADY – the calling FF-A partition is already registered for notification; or</li> <li>• 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</li> <li>• NOMEM – the TPM service does not have memory available to perform the registration.</li> </ul>
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 <a href="#">3.3 Accessing the TPM service functions</a> .
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 <a href="#">3.3 Accessing the TPM service functions</a> .
w4	TPM service function status.	Success status: <ul style="list-style-type: none"><li>OK – the calling FF-A partition has been unregistered from notification.</li></ul> Error status: <ul style="list-style-type: none"><li>NOTSUP – the TPM service does not have the client notification feature; or</li><li>DENIED – the calling FF-A partition is not registered for notification.</li></ul>
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 <a href="#">3.3 Accessing the TPM service functions</a> .
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 <a href="#">3.3 Accessing the TPM service functions</a> . The TPM service may pause this function call; the client must be prepared to resume it.
w4	TPM service function status.	Success status: <ul style="list-style-type: none"><li>OK – command or locality request processing is complete, and Interrupt Status is up-to-date.</li></ul> Error status: <ul style="list-style-type: none"><li>NOTSUP – the TPM service does not have the client notification feature; or</li><li>DENIED – the calling FF-A partition has no outstanding <i>finish</i> callback.</li></ul>

## 6.7 All function status

Table 6.13: TPM service function status codes.

Mnemonic	Description	Value
<b>Error status:</b>		
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
<b>Success status:</b>		
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 bare metal 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

<b>ABI</b>	application binary interface
<b>ACPI</b>	Advanced Configuration and Power Interface
<b>CRB</b>	Command-Response Buffer
<b>DRTM</b>	dynamic Root of Trust for Measurement
<b>FF-A</b>	Firmware Framework for Arm A-profile
<b>nGnRnE</b>	non-Gathering, non-Reordering, No Early Write Acknowledgement
<b>OS</b>	operating system
<b>PM</b>	Partition Manager (FF-A partition manager)
<b>S-EL1</b>	Secure EL1 (Secure Exception Level 1)
<b>SP</b>	Secure Partition (Secure World FF-A partition)
<b>SPM</b>	Secure Partition Manager (Secure World FF-A partition manager)
<b>TCB</b>	Trusted Computing Base
<b>TCG</b>	Trusted Computing Group
<b>TPM</b>	Trusted Platform Module
<b>UUID</b>	Universally Unique Identifier
<b>vCPU</b>	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 [4]. 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 [2] 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.