GLOBALPLATFORM®
THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

GlobalPlatform Technology

TEE System Architecture v1.3

White Paper

May 2022 – GPD_SPE_009

 |

Contents

Figures

Tables

1 Introduction

Devices, from smartphones to servers, offer a Regular Execution Environment (REE), providing a hugely extensible and versatile operating environment. This brings flexibility and capability, but leaves the device vulnerable to a wide range of security threats. The Trusted Execution Environment (TEE) is designed to reside alongside the REE and other Execution Environments and to provide a safe area of the device to protect assets and execute trusted code.

This document explains the hardware and software architectures behind the TEE. It introduces TEE management and explains concepts relevant to TEE functional availability in a device.

At the highest level, a TEE that meets the GlobalPlatform TEE Protection Profile ([TEE PP]) is an environment where the following are true:

- Authenticity: All code executing inside the TEE has been authenticated.
- Integrity: Unless explicitly shared with entities outside the TEE, the ongoing integrity of all TEE assets is assured through isolation, cryptography, or other mechanisms.
- Data Confidentiality: Unless explicitly shared with entities outside the TEE, the ongoing confidentiality of the contents of all TEE data assets – including keys – is assured through isolation or other mechanisms such as cryptography.
- TA Code Confidentiality: TEE capabilities, such as isolation or cryptography, can be used to provide confidentiality of the TA code asset.
- Security: The TEE resists known remote and software attacks, and a set of external hardware attacks.
- Debug and Trace: Both code and other assets are protected from unauthorized debug tracing and control operations being performed through the device's debug and test features.

Note: The architectural concepts and principles in this document do not and should not dictate any particular hardware or software implementation and are broad enough to cover many possible implementations as long as the security principles are adhered to. Hence, any hardware or software architectural diagram in this document is provided as an example and for reference only.

Since release of the first version of this document, many of the requirements to fulfil the goal of being a GPD TEE have become available in specific specification documents. It is not the role of this high-level architecture document to duplicate those detailed requirements, and so many of the statements of this document are intentionally reduced from normative language to informative language.

If you are implementing technology related to this architecture, and you think this document is not clear on something:

1. **Check with a colleague.**

And if that fails:

2. **Contact GlobalPlatform at TEE-issues-GPD_SPE_009_v1.3@globalplatform.org**

1.1 Audience

This document is intended primarily for the use of developers of:

- Trusted Execution Environments
- Trusted Hardware
- Trusted OSes
- Trusted boot loaders
- Trusted Applications that make use of Trusted Execution Environments
- Client Applications that use the services of Trusted Applications by means of the TEE Client API (described in [TEE Client API])

1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://globalplatform.org/specifications/ip-disclaimers/. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

1.3 References

The table below lists references applicable to this specification. The latest version of each reference applies unless a publication date or version is explicitly stated.

Table 1-1: References

Standard / Specification	Description	Ref
GPD_SPE_010	GlobalPlatform Technology TEE Internal Core API Specification	[TEE Core API]
GPD_SPE_007	GlobalPlatform Technology TEE Client API Specification	[TEE Client API]
GPD_SPE_021	GlobalPlatform Technology TEE Protection Profile	[TEE PP]
GPD_SPE_025	GlobalPlatform Technology TEE TA Debug Specification	[TEE TA Debug]
GPD_SPE_024	GlobalPlatform Technology TEE Secure Element API	[TEE SE API]
GPD_SPE_020	GlobalPlatform Technology Trusted User Interface API	[TEE TUI API]
GPD_SPE_055	GlobalPlatform Technology TEE Trusted User Interface Low-level API	[TEE TUI Low]

Standard / Specification	Description	Ref
GPD_SPE_042	GlobalPlatform Technology TEE TUI Extension: Biometrics API v1.0	[TEE TUI Bio]
GPD_SPE_120	GlobalPlatform Technology TEE Management Framework including ASN.1 Profile [Initially published as TEE Management Framework]	[TMF]
GPD_SPE_123	GlobalPlatform Technology TEE Management Framework: Open Trust Protocol (OTrP) Profile	[TMF OTrP]
GPD_SPE_124	GlobalPlatform Technology TMF: Open Trust Protocol (OTrP) Mapping	[TMF OTrP Mapping]
GPD_SPE_100	GlobalPlatform Technology TEE Sockets API Specification	[TEE Sockets]
GPD_GUI_069	GlobalPlatform Technology TEE Initial Configuration	[TEE Init Config]
GPD_GUI_089	GlobalPlatform Technology TMF Initial Configuration	[TMF ASN.1 Config]
GPD_GUI_125	GlobalPlatform Technology TMF: OTrP Profile Initial Configuration	[TMF OTrP Config]
GPD_SPE_075	GlobalPlatform Technology Open Mobile API Specification	[Open Mobile]
GP_REQ_025	GlobalPlatform Technology Root of Trust Definitions and Requirements	[RoT Req]
GP_TEN_053	GlobalPlatform Technology Cryptographic Algorithm Recommendations	[Crypto Rec]
GP_PRO_023	GlobalPlatform Technology TEE Certification Process	[TEE Cert Proc]
ISO/IEC 15408	Information technology – Security techniques – Evaluation criteria for IT security	[ISO 15408]
OMTP ATE TR1	Open Mobile Terminal Platform (OMTP) Advanced Trusted Environment TR1 v1.1 https://www.gsma.com/newsroom/resources/omtp-documents-1-1-omtp-advanced-trusted-environment-omtp-tr1-v1-1/	[OMTP ATE TR1]
RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]
RFC 8174	Amendment to RFC 2119	[RFC 8174]
TCG Glossary	Trusted Computing Group Glossary, https://trustedcomputinggroup.org/wp-content/uploads/TCG-Glossary-V1.1-Rev-1.0.pdf	[TCG_G]

Standard / Specification	Description	Ref
BSI-CC-PP-0084	Common Criteria Protection Profile Security IC Platform Protection Profile with Augmentation Packages	[PP-0084]
embedded MultiMedia Card Product Standard	EMBEDDED MULTIMEDIACARD(e•MMC) e•MMC/CARD PRODUCT STANDARD, HIGH CAPACITY, including Reliable Write, Boot, Sleep Modes, Dual Data Rate, Multiple Partitions Supports, Security Enhancement, Background Operation and High Priority Interrupt https://www.jedec.org/document_search?search_api_views_fulltext=MMC	[eMMC]

1.4 Terminology and Definitions

The following meanings apply to SHALL, SHALL NOT, MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY in this document (refer to [RFC 2119] as amended by [RFC 8174]):

- **SHALL** indicates an absolute requirement, as does **MUST**.
- **SHALL NOT** indicates an absolute prohibition, as does **MUST NOT**.
- **SHOULD** and **SHOULD NOT** indicate recommendations.
- **MAY** indicates an option.

Note that as clarified in the [RFC 8174] amendment, lower case use of these words is not normative.

Table 1-2: Terminology and Definitions

Term	Definition
Access Control Rules (ACR)	A set of rules defining which components or software can access what assets when an isolation boundary is in place.
Application Programming Interface (API)	A set of rules that software programs can follow to communicate with each other.
Biometrics API	An extension of the TEE Trusted User Interface Low-level API that supports the discovery and identification of all biometric capabilities and the use of biometric functionality supported by hardware, entirely protected inside the TEE.
Bus Manager	The component that instigates an action on a bus connection. Typically, the actions will either be to send data to another component or to request data from another component. Some components act as both Bus Manager and Bus Subordinate. For example, a cache will receive requests for data on its Bus Subordinate port and if it doesn't hold that data will then use its Bus Manager port to request that data from other Bus Subordinates to which it is connected. Also known as *Bus Master or Bus Requestor*.

Term	Definition
Bus Subordinate	The component that reacts to an action on a bus connection. Typically, the response to actions will be either to act on data provided by the requesting component or to return data back to the requesting component. Some components act as both Bus Manager and Bus Subordinate. For example, a cache will receive requests for data on its Bus Subordinate port and if it doesn't hold that data will then use its Bus Manager port to request that data from other Bus Subordinates to which it is connected. Also known as *Bus Slave or Bus Completer*.
Client Application (CA)	An application running outside of the Trusted Execution Environment making use of the TEE Client API to access facilities provided by Trusted Applications inside the Trusted Execution Environment. Contrast *Trusted Application*.
Computing Engine	In the context of this document, any system of hardware that will consistently change in a deterministic manner from one state to another. That deterministic manner is governed by the Computing Engine's current state, instruction code, and/or data. A platform has one Computing Engine. This definition does not limit Computing Engines to CPUs, although a Computing Engine must have a minimum of circuitry to perform specific tasks. Note: Hardware entropy sources are acted upon as an aspect of data, and so while they themselves are non-deterministic, the resultant value at a given point in time is acted upon by the Computing Engine in a deterministic manner.
Core Migration	The transfer of the task of execution of code from one CPU core to another.
Execution Environment (EE)	An Execution Environment, as defined in [OMTP ATE TR1], is a set of hardware and software components providing facilities necessary to support running of applications. An EE typically consists of the following elements: • A hardware processing unit • A set of connections between the processing unit and other hardware resources • Physical volatile memory • Physical non-volatile memory • Peripheral interfaces
Extended Root of Trust Component (eRoTc)	A Computing Engine, executable code, and/or data (including cryptographic data) whose integrity is verified by either an Initial Root of Trust Component or another Extended Root of Trust Component at any point during the life cycle of the platform, but whose verification measurement is not discoverable.
GPD TEE	A TEE that is compliant with a GlobalPlatform TEE functionality configuration and certified according to the GlobalPlatform TEE Protection Profile ([TEE PP]).
Hardware isolation	In this document, unless stated otherwise for particular assets, hardware isolation of security related assets is considered to include isolation by electronic access control through the TEE system hardware, that can be configured by TEE resident boot or run-time software.
IC package	See *TEE Hosting IC Package*.

Term	Definition
Initial Root of Trust Component (iRoTc)	A Computing Engine, code, and related data (including cryptographic data) that a platform manufacturer provisions and initializes during the manufacturing process and that is a part of the first code executed on the platform.
In-package	There exist a number of physical boundaries relating to the presence of resources used by the TEE. One of those boundaries is defined by the Integrated Circuit package that contains one or more components of the TEE. While one hardware boundary is often described as on-SoC, in reality it is the SoC packaging material that often forms the boundary. It is important to make this distinction between SoC and Package because it enables the use of more than one chip die inside a package, and thus more facilities inside that hardware boundary. These extra facilities would not be considered “on-SoC” but are considered “in-package”. Examples of an IC package can be found on https://en.wikipedia.org/wiki/List_of_integrated_circuit_packaging_types.
Isolator Control Software	In the context of this document, refers to software, such as hypervisors or similar technology, that is used to create isolation between different Execution Environments.
Live Image	Image data captured in real time by the Biometric Sensor as the end user’s biometric trait is presented to it. Live Image is equivalent to biometric sample (ISO Definition 3.3.21).
One Time Programmable (OTP)	A form of memory that can be read many times, but only written once. On a typical IC package implementing a TEE, this can be a very limited resource on the order of a few thousand bits at most. An example of this form of memory is eFuse.
Package	See *TEE Hosting IC Package*.
Platform	An Execution Environment inside a device. SE, TEE, and REE are examples of platforms.
Protection Profile (PP)	A document according to the Common Criteria, as described in [ISO 15408], used as part of the security certification process; defines the specific set of security features required of a technology to claim compliance.
REE Communication Agent	A Regular OS driver that enables communication between REE and TEE. Contrast *TEE Communication Agent*.
Regular Execution Environment (REE)	An Execution Environment comprising at least one Regular OS and all other components of the device (IC packages, other discrete components, firmware, and software) that execute, host, and support the Regular OSes (excluding any Secure Components included in the device). From the viewpoint of a Secure Component, everything in the REE is considered untrusted, though from the Regular OS point of view there may be internal trust structures. (Formerly referred to as a *Rich Execution Environment (REE)*.) Contrast *Trusted Execution Environment*.

Term	Definition
Regular OS	An OS executing in a Regular Execution Environment. May be anything from a large OS such as Linux down to a minimal set of statically linked libraries providing services such as a TCP/IP stack. (Formerly referred to as a *Rich OS* or *Device OS*.) Contrast *Trusted OS*.
Replay Protected Memory Block (RPMB)	A separate partition inside a non-volatile memory component for which access is authenticated and protected against replay attack. See [eMMC] 4.4 and later.
Root of Trust (RoT)	A Computing Engine, code, and possibly data, all co-located on the same platform; provides security services. No ancestor entity is able to provide a trustable attestation (in digest or other form) for the initial code and data state of the Root of Trust.
Secure Element (SE)	A tamper-resistant secure hardware component which is used in a device to provide the security, confidentiality, and multiple application environment required to support various business models. May exist in any form factor, such as embedded or integrated SE, SIM/UICC, smart card, smart microSD, etc.
Security Domain (SD)	An on-device representative of an Authority in the TEE Management Framework security model. Security Domains are responsible for the control of administration operations. Security Domains are used to perform the provisioning of the TEE properties and manage the life cycle of TAs and SDs associated with them.
Service Provider (SP)	The owner or vendor of a combination of CA and/or TA software.
Shared Trusted Peripheral	A peripheral that is shared with other EE and may be Trusted when made available by the Trusted OS for TA use.
System-on-Chip (SoC)	An electronic system all of whose components are included in a single integrated circuit. Contrast *In-package*.
Tamper-resistant secure hardware	Hardware designed to isolate and protect embedded software and data by implementing appropriate security measures. The hardware and embedded software meet the requirements of the latest Security IC Platform Protection Profile ([PP-0084]) including resistance to physical tampering scenarios described in that Protection Profile.
TEE Client API	The API defined in GlobalPlatform TEE Client API Specification ([TEE Client API]); a communications API for connecting Client Applications running in a REE or other Execution Environment with Trusted Applications running inside a TEE.
TEE Communication Agent	Trusted OS driver that enables communication between REE or other Execution Environment, and the TEE. Contrast *REE Communication Agent*.

Term	Definition
TEE Hosting IC Package	An IC package containing one or more integrated circuit chips. This IC package will host the TEE's Roots of Trust. While this is the primary location for TEE computation activities, a TEE may make use of other IC packages in the device. It will be of a material that will resist direct access to the chip (typically epoxy). See also *In-package.*
TEE Internal APIs	A general series of APIs that provide a common implementation for functionality often required by Trusted Applications. Figure 3-2 illustrates currently included APIs.
TEE Internal Core API	A specific set of APIs providing functionality to the Trusted Application, defined in GlobalPlatform TEE Internal Core API Specification ([TEE Core API]). Figure 3-2 illustrates currently included APIs.
TEE Management Framework (TMF)	A security model for administration of Trusted Execution Environments (TEEs) and for administration and life cycle management of Trusted Applications (TAs) and corresponding Security Domains (SDs).
TEE Secure Element API	The API defined in GlobalPlatform TEE Secure Element API specification ([TEE SE API]); specifies an enabling thin layer to support communication to Secure Elements connected to the device within which the TEE is implemented.
TEE Sockets API	The API defined in GlobalPlatform TEE Sockets API Specification ([TEE Sockets]), including annexes published separately; specifies a generic C interface used by a TA to establish and utilize network communications using a socket style approach.
TEE TA Debug API	The API defined in GlobalPlatform TEE TA Debug Specification ([TEE TA Debug]); specifies a set of APIs to support TA development and/or compliance testing of the TEE Internal APIs.
TEE Trusted User Interface API	The API defined in GlobalPlatform TEE Trusted User Interface API specification ([TEE TUI API]).
TEE Trusted User Interface Low-level API	The API defined in GlobalPlatform TEE Trusted User Interface Low-level API ([TEE TUI Low]).
Trusted Application (TA)	An application running inside the Trusted Execution Environment (TEE) that provides security related functionality to Client Applications outside of the TEE or to other Trusted Applications inside the TEE. Contrast *Client Application*.
Trusted Device Driver	A software package, resident in the TEE, that allows communication (directly or indirectly) between a TA and TEE resident hardware.

Term	Definition
Trusted Execution Environment (TEE)	An Execution Environment that runs alongside but isolated from Execution Environments outside of the TEE. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets against a set of defined threats which include general software attacks as well as some hardware attacks, and defines rigid safeguards as to data and functions that a program can access. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly. (For more information on security requirements, see the GlobalPlatform TEE Protection Profile ([TEE PP]) and [OMTP ATE TR1].) Contrast *Regular Execution Environment (REE)*.
Trusted OS	An OS executing in a Secure Component. In the context of TEE, this OS has been designed primarily to enable the TEE using security based design techniques. It provides the TEE Internal APIs to Trusted Applications and a proprietary method to enable the TEE Client API software interface from other EE. Contrast *Regular OS*.
Trusted storage	In GlobalPlatform TEE documents, *trusted storage* indicates storage that is protected to at least the robustness level defined for OMTP Secure storage (in [OMTP ATE TR1] section 5) or relevant parts of the GlobalPlatform TEE Protection Profile ([TEE PP]). It is protected either by the hardware of the TEE, or cryptographically by keys held in the TEE. If keys are used, they are at least of the strength used to instantiate the TEE. A GlobalPlatform TEE trusted storage is not considered hardware tamper resistant to the levels achieved by Secure Elements, but it is bound to the host device.

1.5 Abbreviations and Notations

Table 1-3: Abbreviations and Notations

Abbreviation / Notation	Meaning
ACR	Access Control Rules
API	Application Programming Interface
BIOS	Basic Input/Output System
CA	Client Application
DLM	Debug Log Message
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DRM	Digital Rights Management
EE	Execution Environment
eRoTc	Enhanced Root of Trust Component
eSE	embedded Secure Element
FPGA	Field-programmable Gate Array
GPD TEE	*See definition in Table 1-2.*
HSM	Hardware Security Module
I/O	Input/Output
IC	Integrated Circuit
IoT	Internet of Things
IP	Internet Protocol
IPR	Intellectual Property Rights
iRoTc	Initial Root of Trust Component
LCD	Liquid Crystal Display
MMU	Memory Management Unit
NFC	Near Field Communications
NVM	Non-volatile Memory
OEM	Original Equipment Manufacturer
OMIL	OTrP Mapping Implementation Layer
OMTP	Open Mobile Terminal Platform
OS	Operating System
OTP	One Time Programmable
OTrP	Open Trust Protocol
PCB	Printed Circuit Board

Abbreviation / Notation	Meaning
PLA	Programmable Logic Array
PMR	Post Mortem Reporting
PP	Protection Profile
RAM	Random Access Memory
REE	Regular Execution Environment
ROM	Read Only Memory
RoT	Root of Trust
RPMB	Replay Protected Memory Block
rSD	root Security Domain
SD	Security Domain
SE	Secure Element
SoC	System-on-Chip
SP	Service Provider
TA	Trusted Application
TCG	Trusted Computing Group
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TLS	Transport Security Layer
TMF	TEE Management Framework
TPM	Trusted Platform Module
TUI	Trusted User Interface
UDP	User Datagram Protocol
UEFI	Unified Extensible Firmware Interface

1.6 Revision History

GlobalPlatform technical documents numbered *n*.0 are major releases. Those numbered *n*.1, *n*.2, etc., are minor releases where changes typically introduce supplementary items that do not impact backward compatibility or interoperability of the specifications. Those numbered *n.n*.1, *n.n*.2, etc., are maintenance releases that incorporate errata and precisions; all non-trivial changes are indicated, often with revision marks.

Table 1-4: Revision History

Date	Version	Description
December 2011	1.0	Initial Public Release
January 2017	1.1	This version of the TEE System Architecture has been extended to include the second phase of TEE standardization, which introduced new APIs for supporting tasks such as Trusted User interface, SE and Sockets communications, and remote management for Trusted Applications.
November 2018	1.2	Introduces new TEE APIs: • TEE Trusted User Interface Low-level API • Biometrics API (an extension of TEE TUI Low-level API) • Peripheral API and Event API (initially published in TEE TUI Low-level API; subsequently published in TEE Internal Core API) Introduces GlobalPlatform Root of Trust Definitions and Requirements in the context of TEE processing. Expands high-level security requirements discussion to include the required security assurance level and the activities of the GlobalPlatform TEE Security Evaluation Secretariat. Clarifies minimum memory requirements of GlobalPlatform compliant TEEs.
May 2022	1.3	Introduces the various Trusted Storage types offered by a TEE. Points developers at the GlobalPlatform Cryptographic Algorithm Recommendations ([Crypto Rec]). Introduces the remote management TMF OTrP profile extensions and the mapping between those and the ASN.1 profile. Enhances the discussion of multiple TEEs, including the trust environment built through use of secure hypervisors and supporting hardware. Clarifies some of the isolation activities and control restrictions that occur around the TEE isolation boundary and acknowledges other Execution Environments in a device may have additional security requirements that the TEE will respect. Provides guidance based on the GlobalPlatform TEE Protection Profile ([TEE PP]) with regard to transfer of peripherals to and from a TEE.

2 TEE Device Architecture Overview

A TEE is an Execution Environment providing security features such as isolated execution, integrity of Trusted Applications (TAs), and integrity and confidentiality of TA assets.

A GPD TEE is defined as one that meets both the following criteria:

- GlobalPlatform functional qualification
  - The TEE SHALL support at least the initial TEE configuration ([TEE Init Config]), which currently consists of being compliant with:

    GlobalPlatform TEE Client API Specification ([TEE Client API])

    GlobalPlatform TEE Internal Core API Specification ([TEE Core API])
  - If the TEE is claimed to fully support other GlobalPlatform TEE specifications, it SHALL do so in a functionally compliant manner.
- GlobalPlatform security certification
  - The TEE SHALL meet the security standard defined by the GlobalPlatform TEE Protection Profile ([TEE PP]).
  - If the TEE is claimed to fully support other GlobalPlatform TEE specifications, it SHALL do so in a security certified manner.
  - Note that the TEE SHALL provide isolation from other environments in the device (including other TEEs). Anything that is not so isolated SHALL be considered part of the TEE.

For a particular device, proof of meeting the above criteria is obtained from relevant and approved certification and compliance laboratories. More information on this can be found on the GlobalPlatform website.

Note:

- The presence of a GPD TEE on a device does not restrict the presence of other Trusted Execution Environments that are not GlobalPlatform compliant.
- A GPD TEE can have better security and/or more capabilities than those required by GlobalPlatform.

The remainder of this chapter describes the general device architecture associated with the TEE and provides a high-level overview of the security requirements of a TEE.

There is no mandated implementation architecture for the described components, and they are used here only as logical constructs within this document.

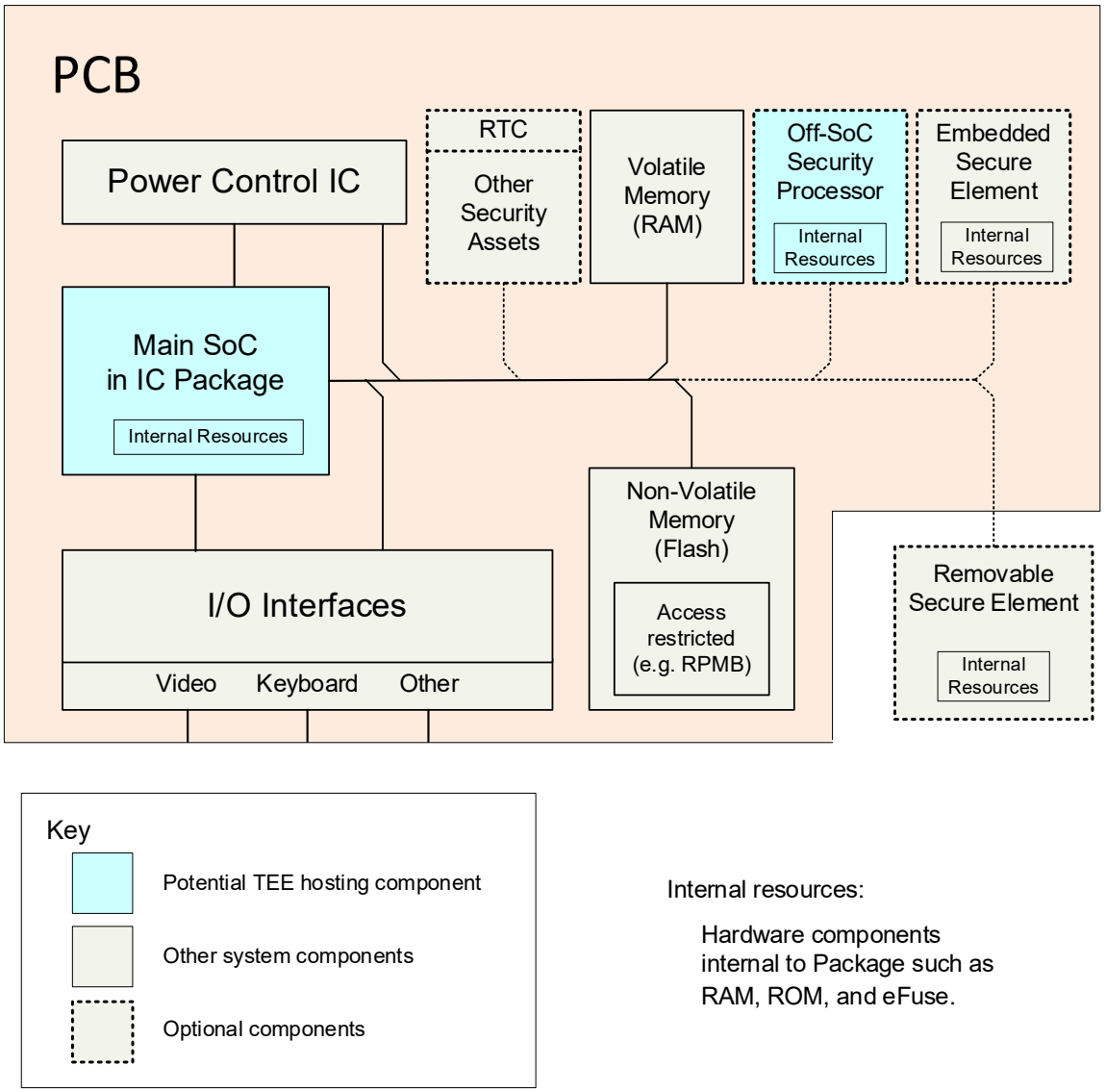
Best effort has been made to make this document align with the required functionality compliance and security certification specifications from GlobalPlatform,

Compliance and certification are performed against the normative configurations and protection profiles.

2.1 Typical Chipset Architecture

Figure 2-1 depicts the board level chipset architecture of a typical mobile device. The chipset hardware consists of a Printed Circuit Board (PCB) that connects a number of components such as SoC processing units, RAM, flash, etc.

Figure 2-1: Chipset Architecture



2.2 Hardware Architecture

The REE and the TEEs utilize a number of resources such as processing core(s), RAM, ROM, cryptographic accelerators, etc. Figure 2-1 above provides a simplified example of the resources that can exist at a device level. Figure 2-2 on page 24 provides an example of the resources that can be associated with a TEE Hosting IC Package such as the package containing the device's main SoC in Figure 2-1.

At any given time, each resource is controlled by the REE or a TEE. Control of part or all of some resources can be transferable between the two environment types. When resources are controlled by a specific TEE, they are isolated from other Execution Environments unless access is explicitly authorized by that controlling TEE. A controlling TEE considers any of its own TEE resources that it does not share to be a trusted resource. These trusted resources are accessible only by other trusted resources and thereby make up a closed system that is protected from other Execution Environments.

Depending on its policies, another Execution Environment may permit some of its resources to be accessible by the TEE without specific permission, whereas the opposite SHALL NOT hold. The other Execution Environment SHALL access TEE resources only with specific permission.

The TEE SHOULD respect the security and access control policies of other Execution Environments and minimize the capability of a TEE unintentionally impacting those other Execution Environments.

GlobalPlatform does not mandate the security requirements of the other parts of the overall device, because those requirements are device specific.

In general terms, the TEE offers an execution space that provides a higher level of security than a Regular OS; although the TEE is not as secure as an SE, the security it offers is sufficient for most applications.

2.2.1 TEE High Level Security Requirements

The high-level security requirements of a TEE can be stated as follows:

- The primary purpose of a TEE is to protect its assets from attacks via the REE and other Execution Environments.
  - This is achieved through hardware mechanisms that those other environments cannot control.
- The TEE is protected against some physical attacks (see [TEE PP]).
  - Typically, this protection will be at a lower level than that provided to dedicated tamper resistant technology.
  - Intrusive attacks that physically break the IC package boundary are normally out of scope of TEE protection.
  - With regard to particular modes of attack such as side channel resistance, etc., see [TEE PP] Annex A.
- System components (such as debug interfaces) capable of accessing assets in a TEE are disabled or are controlled by an element that is itself a protected asset of that TEE.
  - This requirement places no restrictions on system components that cannot access unshared assets of the TEE (e.g. those enabling debug of the REE).
- The Trusted OS run-time environment is instantiated from a Root of Trust (RoT) inside the TEE through a secure boot process using assets either:
  - cryptographically bound to the TEE,
  - or housed in the TEE and isolated from the REE and other TEEs.
  - The integrity and authenticity gained through secure boot:

    Extends throughout the lifetime of the TEE.

    Is retained through any state transitions in the system such as power transitions or core migration.
- The TEE provides Trusted Storage of data and keys.
  - A TEE's Trusted Storage is always bound to that particular TEE on a particular device, such that no unauthorized internal or external attacker can access, copy, or modify the data contained.

    The strength of this binding protection is at least equal to that of the TEE.
  - The Trusted Storage provides a minimum level of protection against rollback attacks.

    The protection levels required against rollback attacks are defined in [TEE Core API] section 5.2.

    The actual physical storage may be in the REE or other Execution Environments. Such storage is vulnerable to bulk rollback and deletion from outside of the TEE. Bulk rollback or deletion is rollback or deletion of the entire TEE trusted store as a block. It SHALL NOT be possible for an unauthorized entity to roll back or delete part of the Trusted Storage.

    A separate class of Trusted Storage is available on some devices where a limited amount of memory is available with improved protections (e.g. RPMB partition or TEE directly managed flash). Such storage has the basic properties of Trusted Storage mentioned above but in addition SHALL NOT be vulnerable to bulk rollback, deletion, or other modification by actors in other Execution Environments.

  - On devices supporting the TMF remote management methods (described in [TMF] & [TMF OTrP]), a class of trusted storage is provided that enables remote entities to securely provision the Trusted Application with keys and/or data. This method uses a separate Trusted Storage partition, enables the TA to choose how to interact with this remotely provisioned data, and prevents that remotely provisioned data from interfering with the TA's other storage classes.

- Software outside the TEE is not able to call directly to functionality exposed by the TEE Internal APIs or the Trusted Core Framework.
  - The non-TEE software goes through protocols such that the Trusted OS or Trusted Application verifies the acceptability of the TEE operation that non-TEE software has requested.

The GlobalPlatform TEE Protection Profile ([TEE PP]) specifies the typical threats that the hardware and software of the TEE needs to withstand. It also details the security objectives that are to be met in order to counter these threats and the security functional requirements that a TEE SHOULD comply with. A security assurance level of EAL2+ has been selected; the focus is on vulnerabilities that are subject to widespread, software-based exploitation.

The GlobalPlatform TEE Security Evaluation Secretariat manages the GlobalPlatform TEE Certification Scheme (ref [TEE Cert Proc]). Under this scheme, providers of TEE products are able to submit their products to this GlobalPlatform Secretariat for independent evaluation of their conformance to the organization's TEE Protection Profile.

2.2.2 Roots of Trust and TEE

The TEE MAY offer at least four different types of RoT services on a device:

- RoT Security Services used during initialization of a Trusted OS
- RoT Security Services offered to Trusted Application on the TEE platform
  - E.g. secure storage offered to TAs by the TEE
- RoT Security Services offered to remote entities (off device) by the TEE platform
  - E.g. GlobalPlatform Trusted Management Framework
- RoT Security Services that are built on Trusted Applications alongside initial boot REE software
  - E.g. a firmware Trusted Platform Module (TPM), as defined in the Trusted Computing Group Glossary ([TCG_G]), running in the TEE providing services to the REE boot

Section 5.2 clarifies RoT services used by a TEE during different potential initialization sequences.

For more detail about Roots of Trust and their use in the TEE context, see GlobalPlatform Root of Trust Definitions and Requirements ([RoT Req]).

2.2.3 TEE Resources

A TEE may be implemented using some or all of three classes of resources. Here we describe their relationship to the IC package that holds the TEE core functionality and to which other parts are bound.

In-package resource

These resources are implemented in the same package as the TEE roots of trust, and so are protected by the IC packaging from a range of physical attacks. In-package communication channels between these resources do not need to be encrypted as they are considered physically secure.

Off-package, cryptographically protected resource

These off-package resources potentially include:

non-volatile memory areas	(often used for TEE_STORAGE_PRIVATE and TEE_STORAGE_PERSO)
access controlled, non-volatile memory areas	(required by TEE_STORAGE_PROTECTED)
volatile memory areas	(used for fast data caches by software in the TEE)

For these memory areas, the security is fulfilled by using proven cryptographic methods (see [TEE PP]). Only the TEE SHALL be able to decrypt the plaintext from the encrypted content stored in these locations. These resources are not protected by being in the same package as the TEE Hosting IC Package, and so the ciphertext can be intercepted while transiting the device PCB.

Exposed or partially exposed resources

TEE-controlled trusted areas of device components external to the TEE Hosting IC Package can contain data not guarded by a proven cryptographic method (see [TEE PP]). This is needed to:

- Enable trusted DRAM-based buffers where the code and data is in the clear but is protected from attack by unauthorized software while being manipulated (e.g. to protect TLS or DRM stream buffers).
- Provide space for a trusted screen frame store.

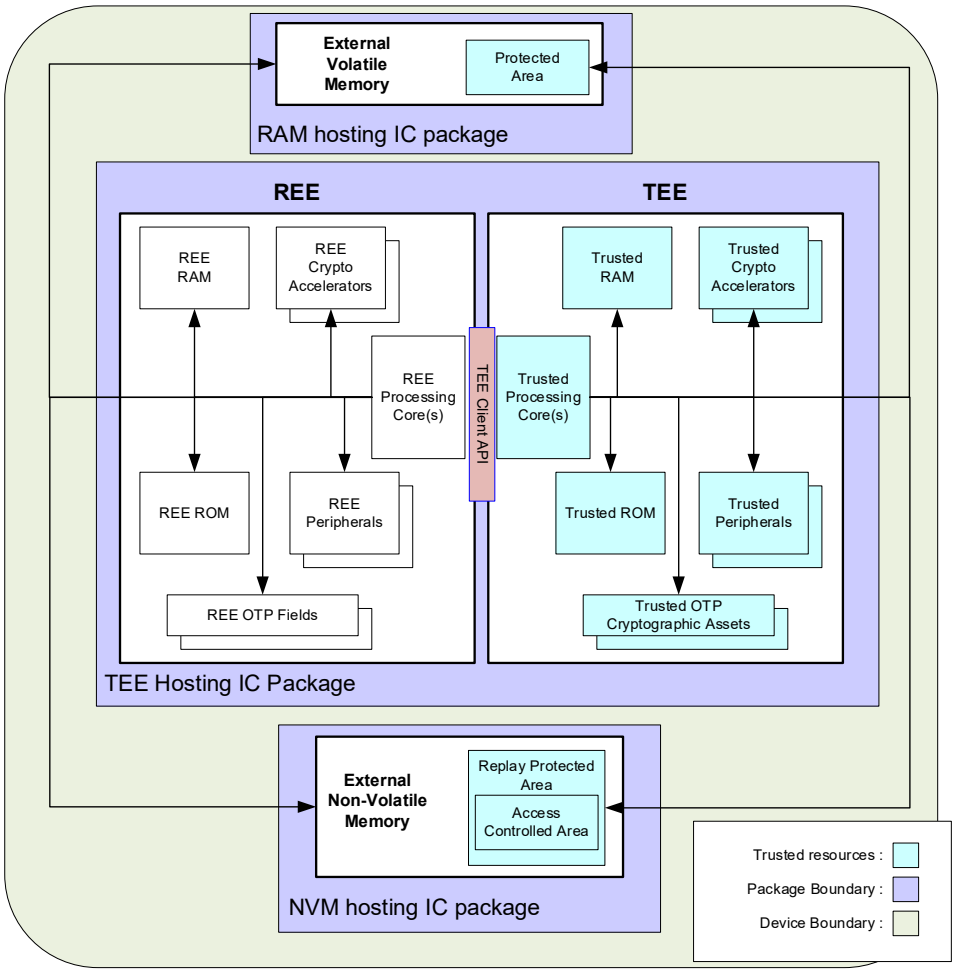
  Neither of the above use cases necessarily requires encrypted RAM storage, just isolation from the REE and other environments.
- Use keyboards and other I/O that are not accessible to the REE but are not guarded from physical attack.

2.2.4 REE and TEE Resource Sharing

The following discussion is simplified to consider the presence of only one TEE and the REE. A TEE is similarly isolated in component ownership and resource sharing from other environments such as SEs and other TEEs.

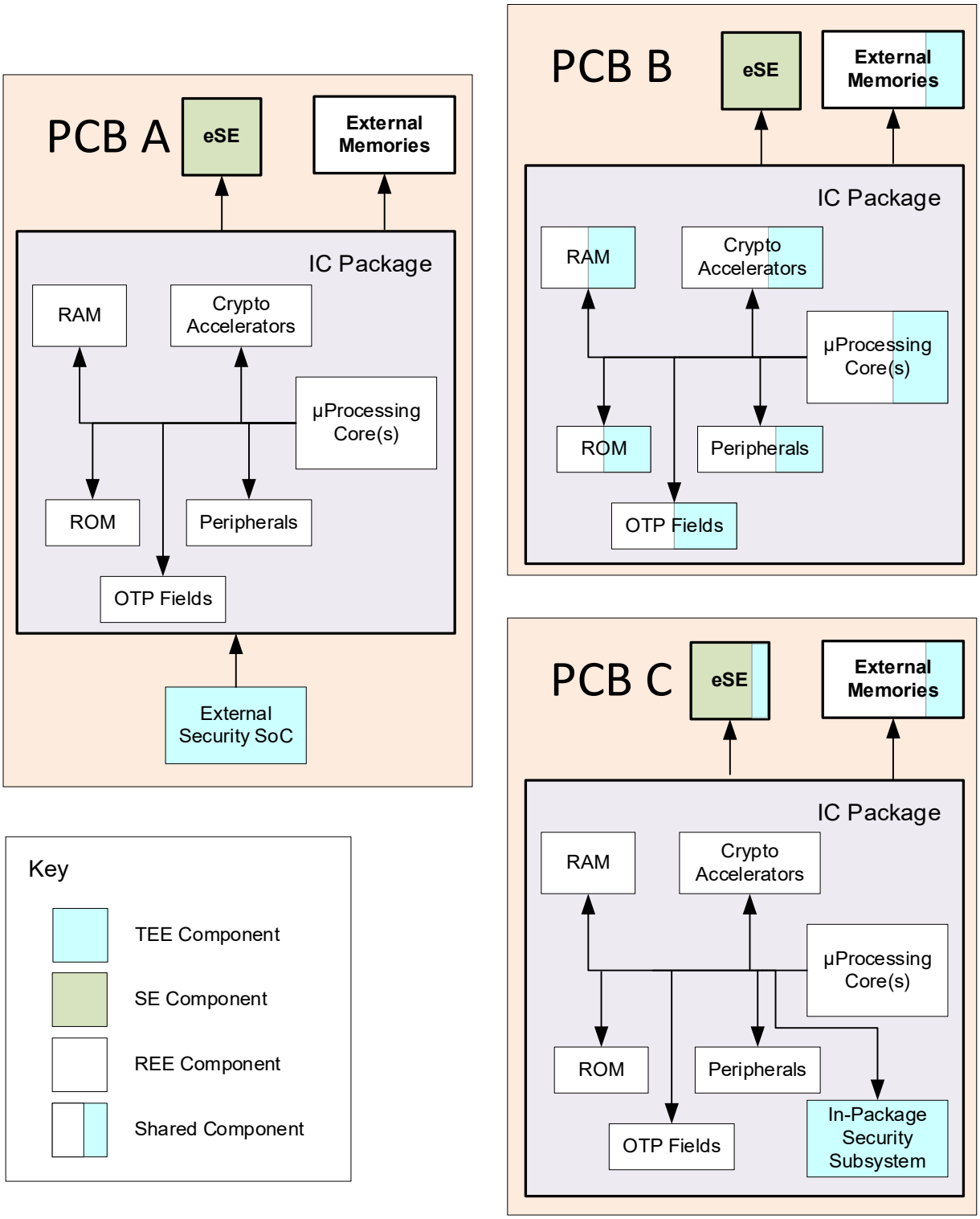
The REE has access to the untrusted resources, which can be implemented either inside the TEE Hosting IC Package or in other components on the PCB. The REE cannot access the trusted resources. This access control is enforced through physical isolation, hardware isolation, or cryptographic isolation methods. The only way for the REE to get access to trusted resources is via API entry points or services exposed by the TEE and accessed through, for example, the TEE Client API. This does not preclude the capability of the REE passing buffers to the TEE (and vice versa) in a controlled and protected manner.

Figure 2-2: Hardware Resource Ownership



Note that the architectural view of TEE and REE illustrated in Figure 2-2 does not dictate any specific physical implementation. Possible implementations include and are not limited to those illustrated in Figure 2-3. Some capabilities MAY not be supportable by all implementations. For example, PCB A in Figure 2-3 cannot support the Trusted User Interface.

Figure 2-3: Example Hardware Realizations of TEE



PCB A Shows a TEE housed in a separate security SoC with its own internal RAM, ROM, cores, and other peripheral components.

PCB B Shows a TEE that uses some form of internal IC Package filter to isolate sections of RAM, ROM, core(s), etc., for its own use.

PCB C Shows a TEE that uses some form of internal IC Package separation to isolate a sub-system with its own internal RAM, ROM, cores, and other peripheral components. As an additional optional factor, it is also making use of an eSE and external memory to hold some materials.

2.2.4.1 Isolation of Trusted Resources

In some systems, a TEE trusted resource cannot be physically isolated from the other Execution Environments and is accessed via a memory bus that is shared with the other Execution Environments. For example, PCB B in Figure 2-3 uses a common memory bus for all REE and TEE components. Depending on the security requirements for the trusted resource, the TEE must implement either hardware isolation or cryptographic isolation of these trusted resources.

Cryptographic isolation is typically used for off-package trusted resources; see TEE Resources (section 2.2.3).

Hardware isolation is typically used for separating resources within the TEE Hosting IC Package. This isolation is implemented by filtering address accesses on the bus, to ensure that only access requests originating from the TEE are permitted to reach the trusted resource.

Where this is done, it is necessary to control access from all Bus Managers that are controlled by other Execution Environments, not just processing cores. See Isolator type #B in section 3.6.4.1 for more detail. Note that the hardware components that manage or implement the isolation of a TEE must also be trusted resources of that TEE.

If a TEE's hardware isolation is configurable, only that TEE may configure it:

- An isolator that protects a resource that is private to a particular TEE should be configured when that TEE boots, prior to enabling other Execution Environments to boot. See also section 5.2.1, Typical Boot Sequence.
- An isolator that protects a configurable resource that is shared with other Execution Environments (for example, a display that is used by the TEE for a Trusted UI) must be reconfigured each time the resource is transferred between the TEE and another Execution Environment. This type of Shared Trusted Peripheral is depicted in Figure 3-1. See also section 5.4, Transfer of Hardware Components to and from the TEE.
- In some cases, a commonly trusted component of the relevant group of TEEs in the device will provide this isolation before a particular TEE takes over ownership of the resource.

3 TEE Software Interfaces

The TEE is a separate Execution Environment that runs alongside the REE and other environments and provides security services to those other environments and to applications running inside those environments. The TEE exposes sets of APIs to enable communication from the REE and other APIs to enable Trusted Application software functionality within the TEE.

This chapter describes the general software architecture associated with the TEE, the interfaces defined by GlobalPlatform, and the relationship between the critical components found in the software system.

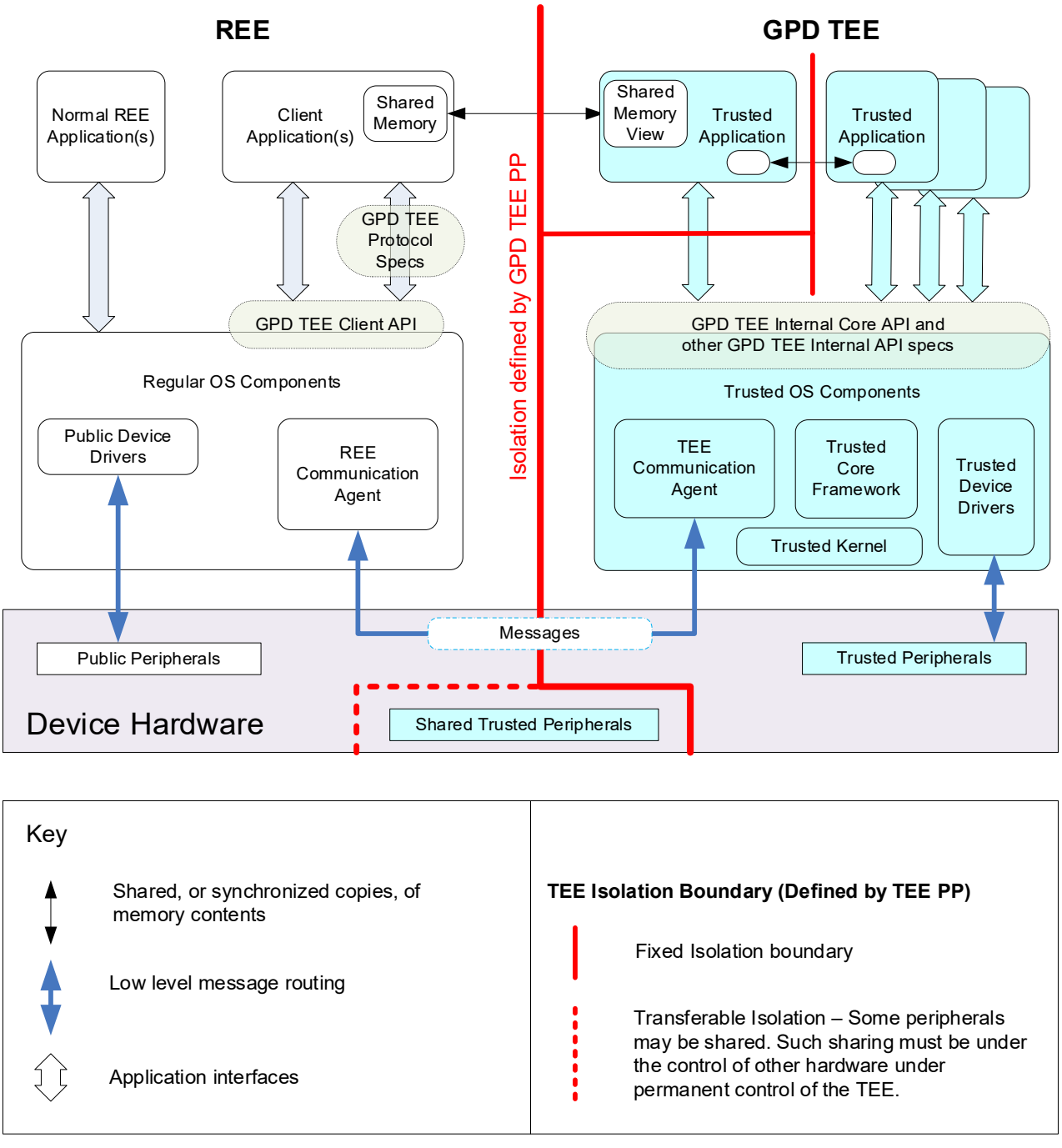
There is no mandated implementation architecture for these components, and they are used here only as logical constructs within this document.

The components and logical constructs can also reside in different architectural locations than those shown as long as they meet the restrictions specified by the GlobalPlatform TEE Protection Profile ([TEE PP]) and GlobalPlatform API specifications for a GlobalPlatform TEE.

3.1 The TEE Software Architecture

Figure 3-1 outlines the relationship between the major software systems components.

Figure 3-1: TEE Software Architecture



The goal of the TEE Software Architecture is to enable Trusted Applications (TAs) to provide isolated and trustworthy capabilities, which can then be used through Client Applications (CAs).

Please note:

- Just as there are many hardware solutions to implementing a TEE (see Figure 2-3), there can also be many software configurations of a TEE (or even TEEs) in a device. The following sections discuss some possible configurations.
- For simplicity, subsequent graphics show only the fixed isolation boundary discussed in Figure 3-1. However, Shared Trusted Peripherals (as illustrated and described in Figure 3-1) are possible in all configurations.

3.2 Components of a GPD TEE

3.2.1 REE Interfaces to the TEE

Within the REE, the architecture identifies an optional protocol specification layer, an API, and a supporting communication agent.

- The REE Communication Agent provides REE support for messaging between the Client Application and the Trusted Application.
- The TEE Client API is a low-level communication interface designed to enable a Client Application running in the Regular OS to access and exchange data with a Trusted Application running inside a Trusted Execution Environment.
- The TEE Protocol Specifications layer exposed in the REE offers Client Applications a set of higher-level APIs to access some TEE services. TEE TA Debug API ([TEE TA Debug]), the TMF ASN.1 Profile (described in [TMF]), and TMF OTrP Profile ([TMF OTrP]) currently use this stack layer. TA developers can develop additional proprietary TEE APIs at the TEE Protocol Specifications layer.

3.2.2 Trusted OS Components

Within the TEE, the architecture identifies two distinct classes of software: the hosting code provided by Trusted OS Components, and Trusted Applications, which run on top of that code.

Trusted OS Components

- The Trusted Core Framework provides OS functionality to Trusted Applications.
  - The Trusted Core Framework is part of the TEE Internal Core API, discussed in section 3.5.1.
- The Trusted Kernel provides scheduling and other OS management functions for both Trusted Applications and the Trusted Core Framework.
- Trusted Device Drivers provide a communications interface to trusted peripherals that are dedicated to the TEE.
  - Trusted Device Drivers can be an integral part of the Trusted Kernel or can be modular components, depending on the architecture of the Trusted Kernel.
- The TEE Communication Agent is a special case of a Trusted OS component. It works with its peer, the REE Communication Agent, to safely transfer messages between CA and TA.

As stated at the start of this chapter, the descriptions and diagrams are examples of potential architectural arrangement. It should be noted that potentially some of the functionality shown here and elsewhere as coming from the Trusted OS Components may actually be provided by libraries statically linked to the TA itself, or from dynamically loaded components that effectively reside in what might be thought of as the TA space.

3.2.3 Trusted Applications (TAs)

Trusted Applications communicate with the rest of the system via APIs exposed by Trusted OS components.

- The TEE Internal APIs define the fundamental software capabilities of a TEE.
- Other non-GlobalPlatform internal APIs can be defined to support interfaces to further proprietary functionality.

When a Client Application creates a session with a Trusted Application, it connects to an instance of that Trusted Application. A Trusted Application instance has physical memory address space which is separated from the physical memory address space of all other Trusted Application instances.

A session is used to logically connect multiple commands invoked in a Trusted Application. Each session has its own state, which typically contains the session context and the context(s) of the task(s) executing the session.

It is up to the Trusted Application to define the combinations of commands and their parameters that are valid to execute.

TAs can start execution only in response to an external command, session start, or instance creation. They make their own choice as to when to return from that command. Typical TAs follow a short command response life cycle, but complex TAs can iterate for long periods while processing input and output events such as TUI.

GlobalPlatform compliant TEEs validated against one GlobalPlatform configuration (such as [TEE Init Config]) require a minimum amount of memory to enable testing of that TEE. As such, a TEE that has passed GlobalPlatform compliance has at least this minimum memory capability. As each TEE implementation can use different build systems, and TAs are defined in terms of source code, that amount of memory is target dependent but as general guidance it is possible to state the following:

- A compliant TEE SHALL be able to host TAs that use up to:
  - Heap per TA: 5 Kbytes
  - Stack per TA: 336 Bytes
  - Binary TA code: 65 Kbytes ELF format file for one TA
- A compliant TEE SHALL be able to host two TAs at the same time to pass some tests:
  - Binary TA code: 57 Kbytes ELF format file for TA 1
  - Binary TA code: 44 Kbytes ELF format file for TA 2

As a reference, a stub TA with no calls to TEE functionality, using the same build method as applied above:

  - Binary TA code: 22 Kbytes ELF format file

This information is provided to give the reader an indication of memory resources a minimal TEE system SHALL provide. In reality, these are minimums and GlobalPlatform compliant TEEs are usually capable of hosting far larger TAs and providing far larger stack and heap space. Contact your TEE implementers for details.

3.2.4 Shared Memory

One feature of a TEE is its ability to enable the CA and TA to communicate large amounts of data quickly and efficiently via memory area accessible to both the TEE and the REE (or other Execution Environment). The API design allows this feature to be implemented by the Communication Agents (Figure 3-1) either as memory copies or as directly shared memory. The protocols for how to make use of this ability are defined by the TA designer, and enabled by the TEE Client API and TEE Internal Core API.

Care has to be taken with the security aspects of using shared memory, as there is potential for a Client Application or Trusted Application to modify the memory contents asynchronously with the other parties acting on that memory.

3.2.5 TA to TA Communication

A TA can communicate to another TA inside the same TEE. This uses the same process used by the CA to communicate to the TA, but a trustworthy flag allows the receiving TA to be assured that communication has not been exposed outside the TEE. This simplifies determining whether to trust the communication content and also the metadata associated with the content, such as the identity of the calling TA.

A communication session from a TA in another TEE will be indicated by the trustworthy flag as having originated outside of the TEE.

- If one TA has mitigating factors, then the other TA may choose to trust the first TA more than it would a REE CA.
- If there are no mitigating factors, then a TA in another TEE in the same device should be treated as though it is a REE CA, because the receiving TA's TEE has no reason to trust the calling TA's TEE.

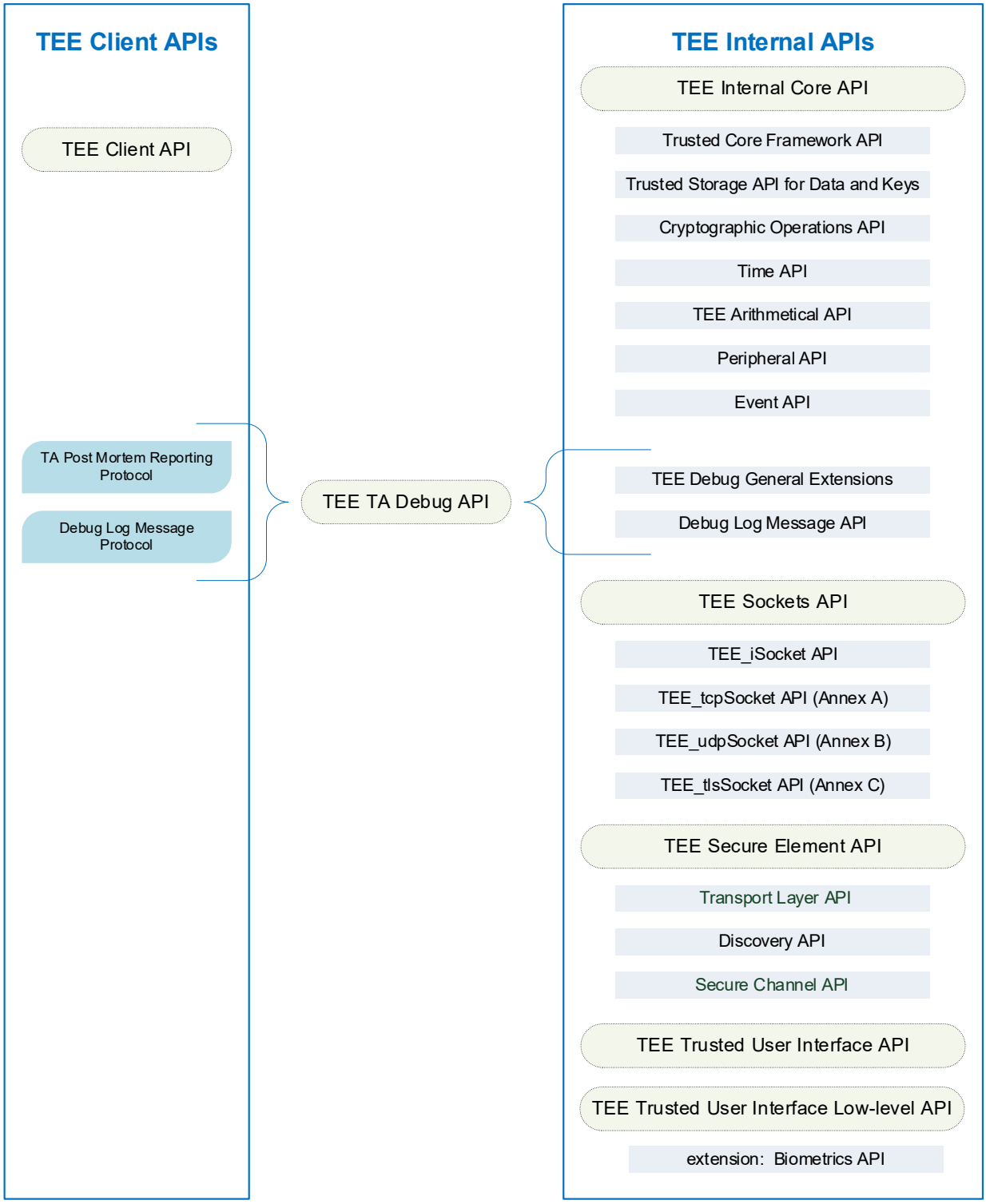
Mitigating factors might be knowledge by the TA that the message routing is isolated and that the other TEE was provided by a trustworthy source.

Future releases will address mitigating factors. Until the GlobalPlatform specifications are updated to provide further information, it will be up to the TA (and hence the TA designer) as to how it considers mitigating factors that it may be aware of.

3.3 Relationship between TEE APIs

Figure 3-2 outlines the relationships between the various APIs and released specification documents.

Figure 3-2: TEE APIs



3.4 The TEE Client API Architecture

GlobalPlatform specifies the TEE Client API in the GlobalPlatform TEE Client API Specification ([TEE Client API]). The TEE Client API concentrates on the interface to enable efficient communications between a Client Application and a Trusted Application.

Higher level standards and protocol layers (known as TEE Protocol Specifications and functional APIs) can be built on top of the foundation provided by the TEE Client API – for example, to support common tasks such as trusted storage, cryptography, and run-time installation of new Trusted Applications.

Within the REE, this architecture identifies three distinct classes of component:

- The Client Applications, which make use of the TEE Client API
- The TEE Client API library implementation
- The REE Communication Agent, which is shared amongst all Client Applications, and which handles communications between the REE and the TEE

The REE implementer can choose to expose the TEE Client API to the user layer, the privileged layer, or both. If exposed in the privileged layer, then drivers or any other privileged components can be considered to take the place of Client Applications. The API is typically blocking on a per thread basis, but can be called asynchronously from multiple threads.

A typical application will use the TEE Client API to:

- Establish communications with its chosen TEE
- Establish a session with a Trusted Application
- Set up shared memory, if it wants to expose that memory to the TA
- Send TA specific commands to invoke a trusted service provided by that TA
- Then cleanly shut down communications

More information on the TEE Client API can be found in [TEE Client API].

3.5 The TEE Internal API Architecture

GlobalPlatform specifies a series of APIs to provide a common implementation for functionality typically required by many Trusted Applications. The TEE Internal Core API is specified in the GlobalPlatform TEE Internal Core API Specification ([TEE Core API]). The TEE Internal Core API concentrates on the various interfaces to enable a Trusted Application to make best use of the standard TEE capabilities. Additional low-level functionality is provided by optional TEE Internal APIs such as the TEE Secure Element API, TEE Sockets API, and TEE TA Debug API.

An introduction to each of these APIs is provided in this document. For clarity, the latest versions of each of these documents should be read in place of these introductions.

Higher level standards and protocol layers can be built on top of the foundation provided by the TEE Internal APIs – for example, to support common tasks such as creating a trusted password entry screen for the user, confidential data management, financial services, and Digital Rights Management.

Within the TEE, this architecture currently identifies three distinct classes of component:

- Trusted Applications, which make use of TEE Internal APIs
- The TEE Internal API library implementations
- Trusted OS Components, which are shared amongst all Trusted Applications, and which provide the system level functionality required by the Trusted Applications

3.5.1 The TEE Internal Core API

The TEE Internal Core API provides several subsets of functionality to the Trusted Application.

Table 3-1: APIs within TEE Internal Core API

API Name	Description
Trusted Core Framework API	This API provides integration, scheduling, communication, memory management, and system information retrieval interfaces.
Trusted Storage API for Data and Keys	This API provides Trusted Storage for keys and general data.
Cryptographic Operations API	This API provides cryptographic capabilities.
Time API	This API provides support for various time-based functionality to support tasks such as token expiry and authentication attempt throttling.
TEE Arithmetical API	This API provides arithmetical primitives to create cryptographic functions not found in the Cryptographic Operations API.
Peripheral API	This API enables a Trusted Application to interact with peripherals via the Trusted OS. Initially defined in GlobalPlatform TEE Trusted User Interface Low-level API ([TEE TUI Low]), then defined in [TEE Core API] beginning with v1.2.
Event API	This API supports the event loop, which enables a TA to enquire for and then process messages from types of peripherals including pseudo-peripherals. Initially defined in [TEE TUI Low], then defined in [TEE Core API] beginning with v1.2.

More information on the TEE Internal Core API can be found in [TEE Core API].

3.5.1.1 Trusted Storage API

This API provides Trusted Storage for keys and general data.

There are now potentially three separate areas of storage a TA can access through this API:

Table 3-2: Storage Areas

Storage Areas	Description
TEE_STORAGE_PRIVATE	Only accessible by the TA. This resource may be implemented with different levels of rollback protection. Required by all GlobalPlatform compliant implementations.
TEE_STORAGE_PROTECTED	Only accessible by the TA. This resource may only be implemented with the highest level of rollback protection available in the TEE. However, it may be a smaller resource and not available at all to some TAs and on some devices. Optional feature added in [TEE Core API] v1.3.
TEE_STORAGE_PERSO	This resource may be implemented with different levels of rollback protection. This resource is only written to by the managing remote entities through the TEE Management Framework ([TMF]). It enables remote personalization of a TA. Only readable by the TA. If the TA requires written data to be stable, then it should copy that data into TEE_STORAGE_PRIVATE or TEE STORAGE_PROTECTED, which no other entities can write into. Required by all TEE Management Framework specification compliant implementations.

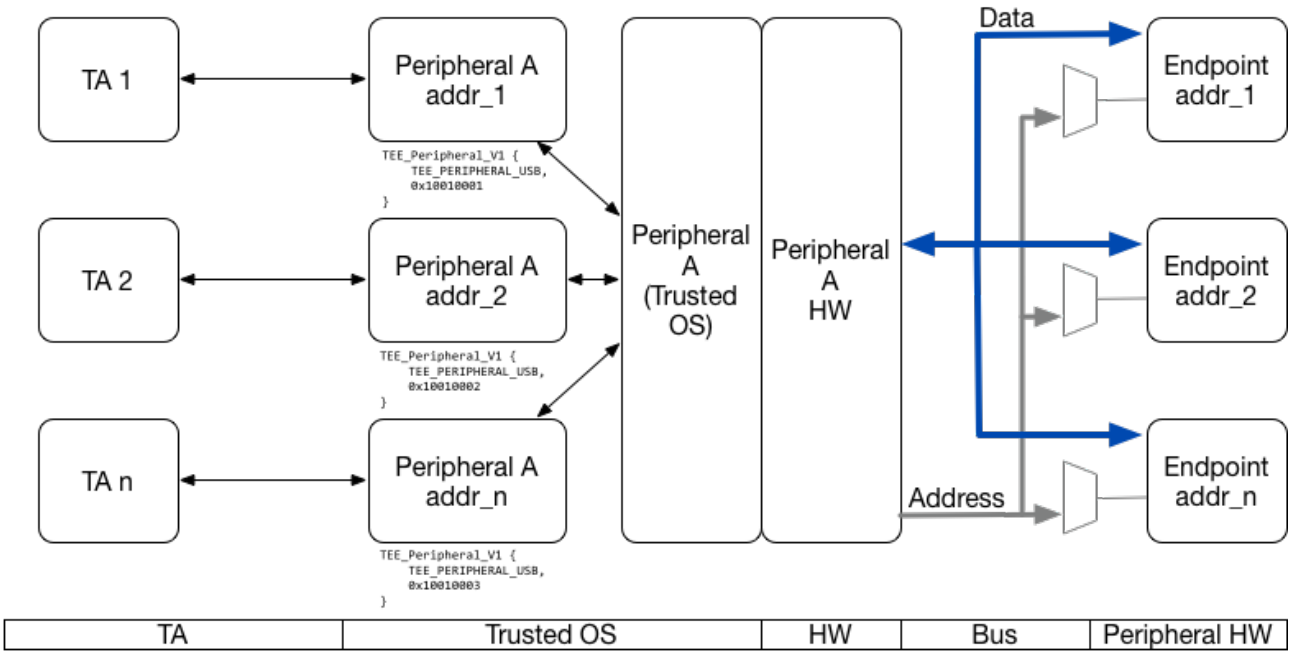
3.5.1.2 Peripheral and Event APIs

The optional Peripheral and Event APIs support asynchronous interfacing for a TA to TEE internal and external events, alongside a generic interface to peripherals.

Some peripherals offer multiple channels, addressing capability, or other mechanisms which have the potential to allow access to multiple endpoints. It can be convenient in some scenarios to assign different logical endpoints to different TAs, while supporting a model of exclusive access to the peripheral per TA.

One approach, shown in Figure 3-3, is to implement a separate driver interface for each of the multiple endpoints. For example, a driver for an I^2C interface can support separate endpoints for each I^2C address, while itself being the exclusive owner of the I^2C peripheral. As with any other information asset, the Protection Profile ([TEE PP]) implies that such drivers SHALL ensure that information leakage between the TA clients of the different endpoints is prevented.

Figure 3-3: Example of Multiple Access to Bus-oriented Peripheral



3.5.2 The TEE Sockets API

The GlobalPlatform TEE Sockets API Specification ([TEE Sockets]) provides a common modular interface for the TA to communicate to other network nodes, acting as a network client.

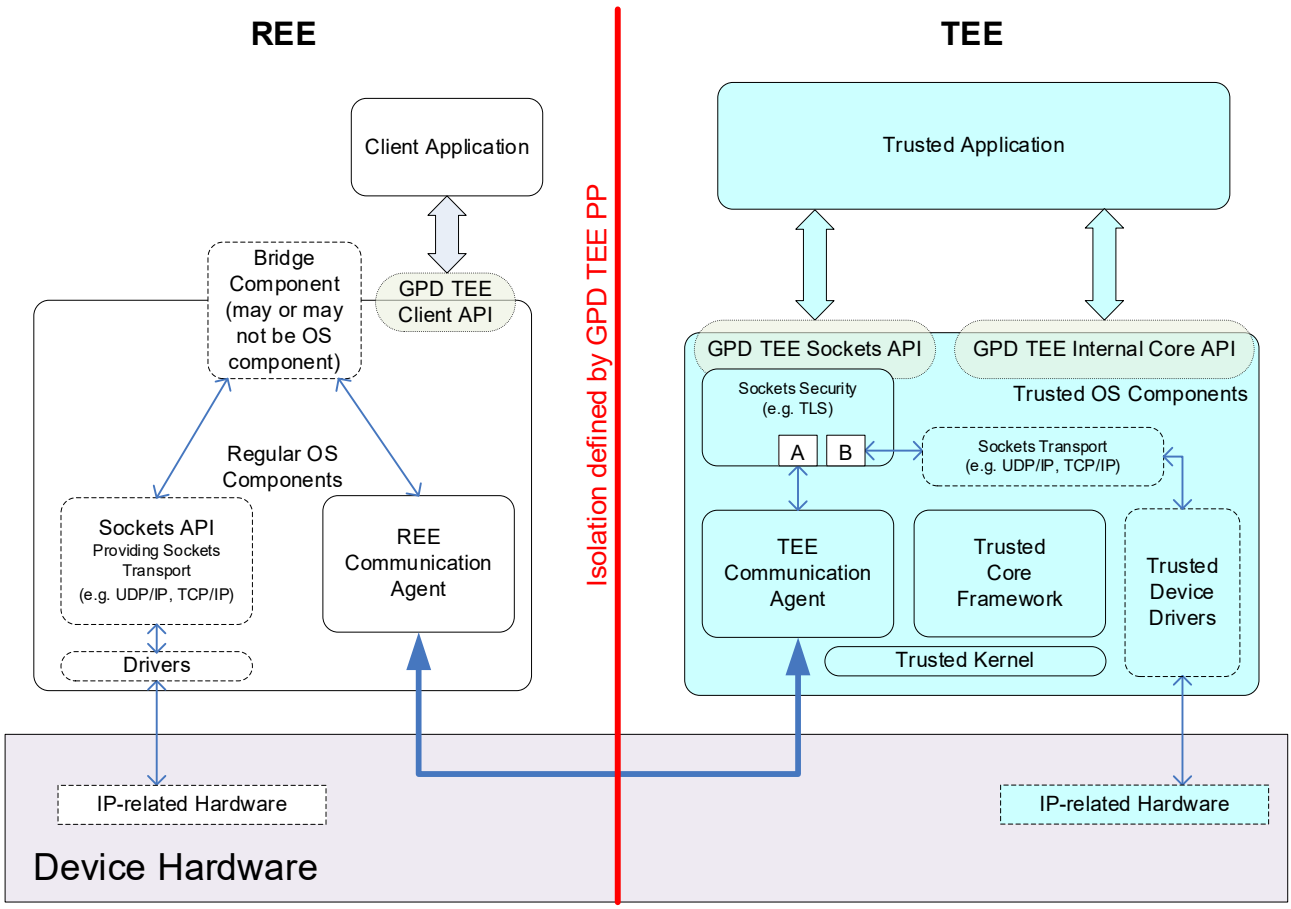
The TEE Sockets API is the general API for accessing and handling client sockets of various kinds.

TEE Sockets API Annex A specifies the TEE_iSocket interface for Transmission Control Protocol (TCP).

TEE Sockets API Annex B specifies the TEE_iSocket interface for User Datagram Protocol (UDP).

TEE Sockets API Annex C specifies the TEE_iSocket interface for Transport Layer Security (TLS).

Figure 3-4: Example TEE Sockets API Architecture



The above diagram shows two routing options (A) and (B) inside the TEE. These are options because only the security layer has to reside inside the TEE. It is expected that a real implementation would need only one of these options (A) or (B). Typically functionality such as UDP/IP and TCP/IP can be placed in the REE without security risks, so placing Sockets Transport in the TEE is optional as well.

More information on the TEE Sockets API can be found in the GlobalPlatform TEE Sockets API Specification ([TEE Sockets]).

3.5.3 The TEE TA Debug API

The TEE TA Debug API provides services that are designed to support TA development and/or compliance testing of the TEE Internal APIs.

The Post Mortem Reporting (PMR) service supports compliance testing and TA debug. This service provides a method for a TEE to report to clients the termination status of TAs that enter the Panic state. Without this capability it is not possible to certify correct functionality of the TEE Internal APIs, as the Panic state is used to report various error conditions that need to be tested.

The Debug Log Message (DLM) service is useful in a TA debug scenario. This service provides a method for a TA to report simple debug information on authorized systems. It can report to client applications, off-device hardware, or both.

More information about the TEE TA Debug API Architecture can be found in the GlobalPlatform TEE TA Debug Specification ([TEE TA Debug]).

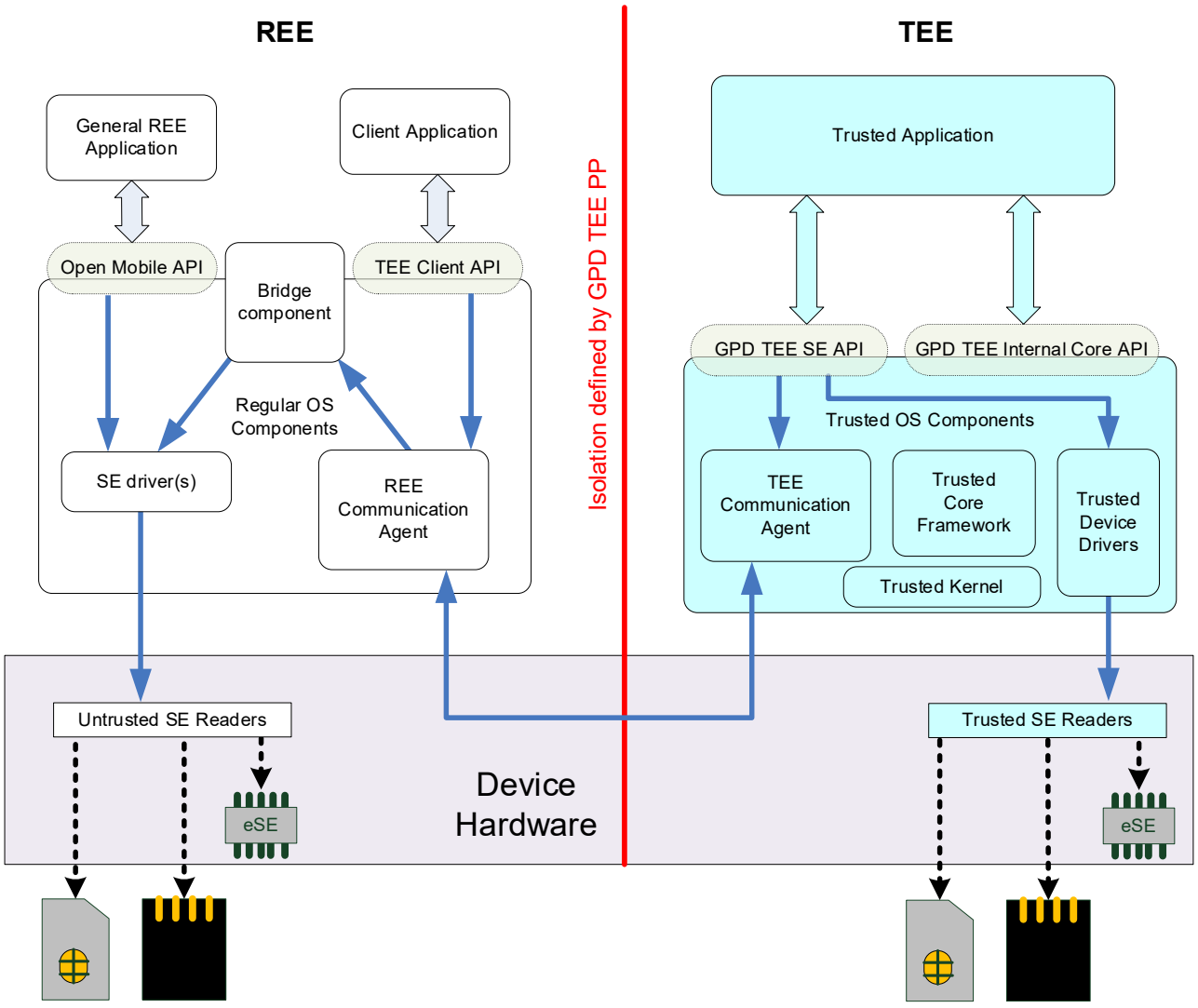
3.5.4 The TEE Secure Element API

The TEE Secure Element API is an enabling thin layer that supports communication to Secure Elements (SEs) connected to the device within which the TEE is implemented. This API defines a transport interface based on the GlobalPlatform Open Mobile API Specification ([Open Mobile]).

SEs can be connected in a shared way via the REE or exclusively to the TEE.

- An SE connected exclusively to the TEE is accessible by a TA without using any resources from the REE. Thus the communication is considered trusted.
- An SE connected to the REE is accessible by a TA using resources within the REE. It is recommended that the Secure Channel API be used to protect the communication between the TA and the SE against attacks in the REE.

Figure 3-5: Typical Device with Multiple SE Readers



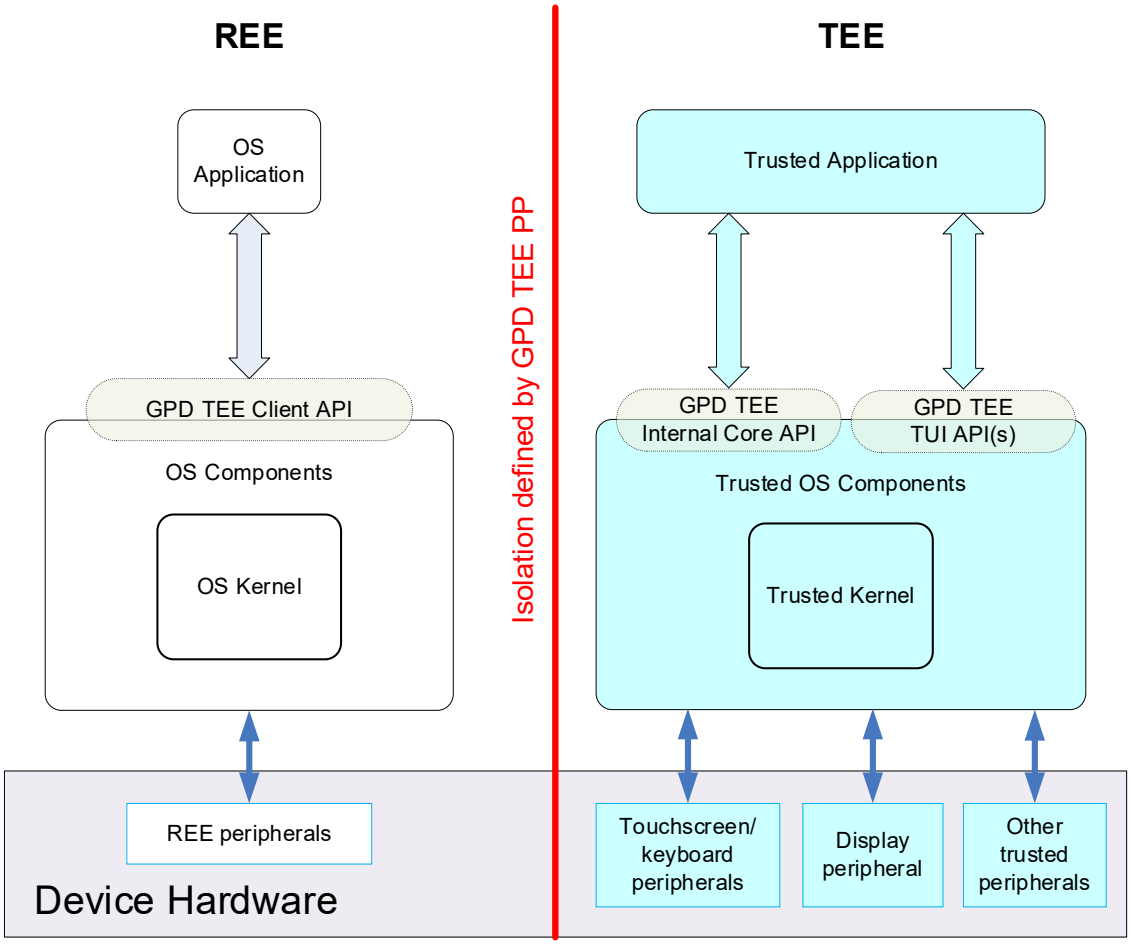
More information about the TEE Secure Element API can be found in the GlobalPlatform TEE Secure Element API specification ([TEE SE API]).

3.5.5 The TEE Trusted User Interface API

The Trusted User Interface API permits the display of screens to the user while achieving three objectives:

- Secure display – Information displayed to the user cannot be accessed, modified, or obscured by any software within the REE or by an unauthorized application in the TEE.
- Secure input – Information entered by the user cannot be derived or modified by any software within the REE or by an unauthorized application in the TEE.
- Security indicator – The user can be confident that the screen displayed is actually a screen displayed by a TA.

Figure 3-6: TEE with TUI Architecture



Remote parties SHOULD NOT treat the user's identification as a trustworthy identification by itself, but only in combination with a factor known only to the TA in the TEE (such as a key).

Use of the TEE TUI also provides a third party the guarantee of non-interference. The remote party can have confidence that what the user signs is what they actually saw, and not some information spoofed into the UI, replacing the desired display information.

More information about the TEE Trusted User Interface can be found in the GlobalPlatform TEE Trusted User Interface API specification ([TEE TUI API]) and in the GlobalPlatform TEE Trusted User Interface Low-level API ([TEE TUI Low]).

3.5.6 The Biometrics API – an Extension of TEE TUI Low-level API

Biometric capabilities and their functionality as present in the hardware of the TEE are made available to TAs via the Biometrics API ([TEE TUI Bio]). The biometric capabilities are contained in the Biometric Sub-system, consisting of Biometric Peripherals which use Biometric Sensors.

- A Biometric Sub-system is a component of the TEE, composed of all TEE Biometric Peripherals in the device plus any supporting TEE or REE software and hardware.
- A Biometric Peripheral is a component of the Biometric Sub-system.
- A Biometric Sensor provides the Live Image and possibly other related services.

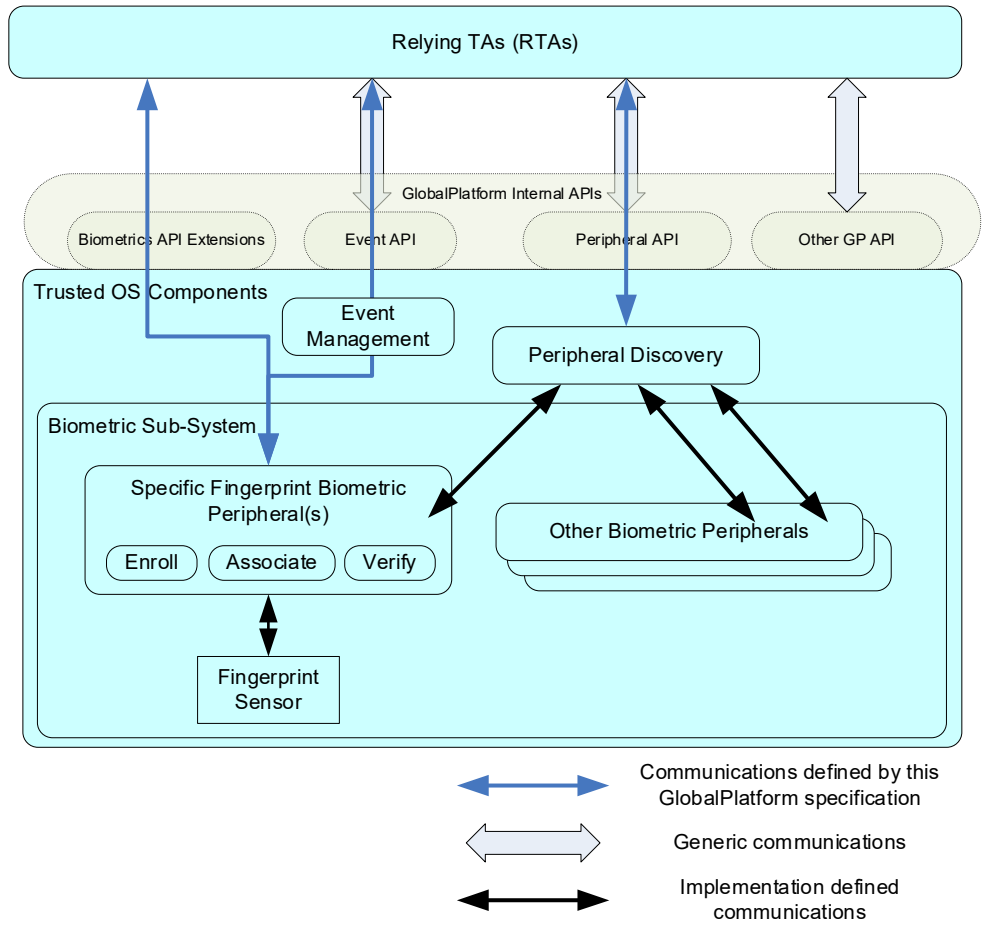
In general, it is an implementation choice as to whether particular functionality is implemented in the generic Biometric Sub-system, the Biometric Peripheral, or the Biometric Sensor. Such decisions SHALL be transparent to the calling TA.

When interacting with the Biometric Sub-system of the TEE, the first step is the discovery of the available biometric capabilities present in the platform. This is performed using the standard discovery mechanisms in the Peripheral API (discussed in section 3.5.1.2).

The relying TA interacts with the Biometric Peripheral; it SHALL NOT be possible to interact directly with the Biometric Sensor.

Once the biometric capabilities are known, the TA can select a Biometric Peripheral and use its service, as shown for the specific case of fingerprint biometrics in Figure 3-7.

Figure 3-7: Architecture Overview – Multiple Biometrics

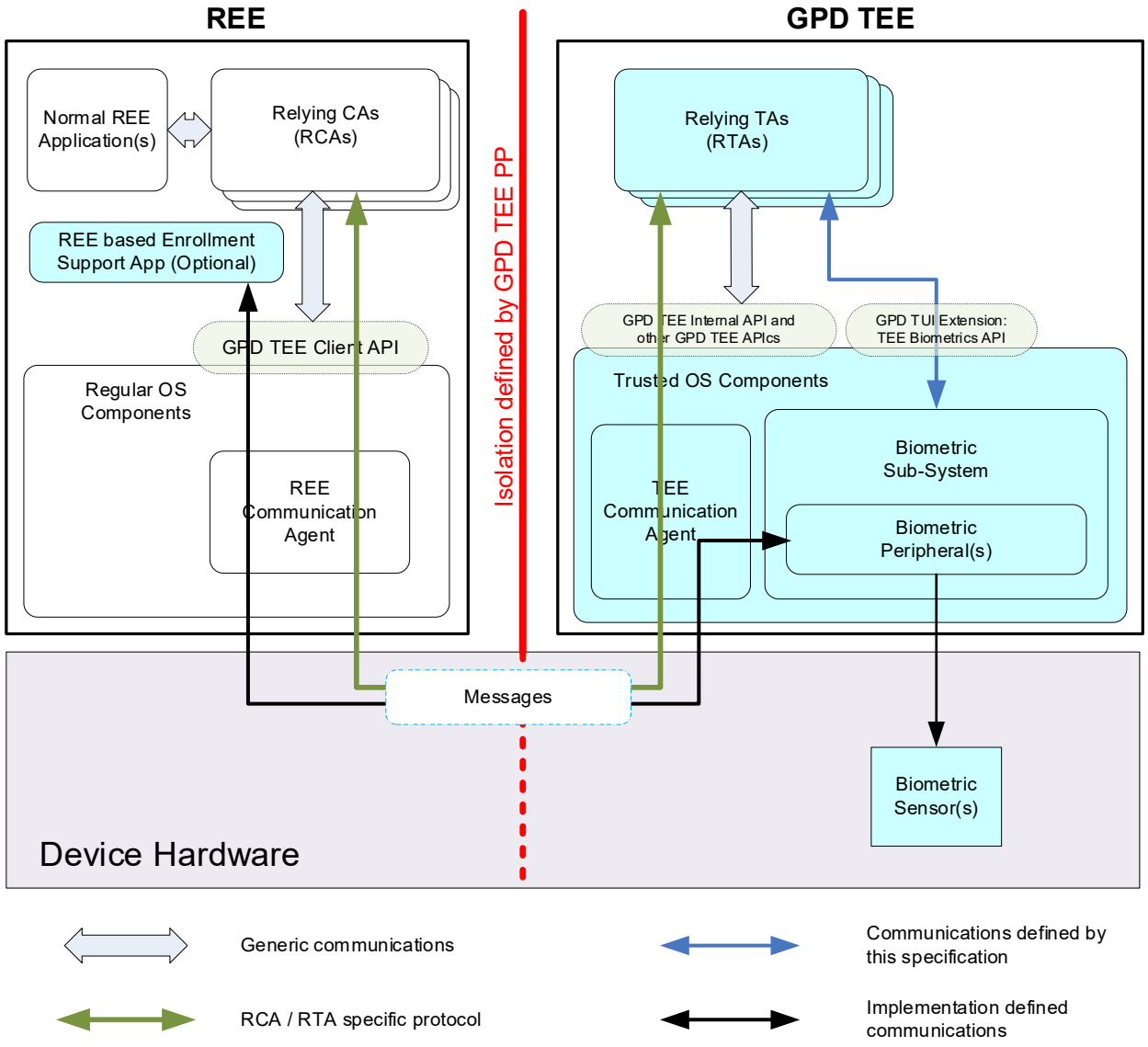


The Biometric Sub-system is integrated into the TEE and provides a service through the established interfaces. It MAY utilize TEE secure storage, along with REE and SE capabilities as appropriate and available on any specific platform. Figure 3-8 shows the general positioning of the Biometric Sub-system, the Biometric Peripheral, and the Biometric Sensor in a conceptual TEE architecture.

Part or all of the Biometric Peripheral MAY optionally be implemented as TAs executing on the TEE, or in one of the available SEs executing as “Match on Card”. In addition, some functionality that is not security-critical MAY be handled by Biometric Sub-system components in the REE. Each variation provides different advantages and limitations; the choice of architecture in this respect is left to the device manufacturer.

Regardless of where the Biometric Sub-system is placed, its execution and all data, whether long term stored or run-time, SHALL be protected using the security criteria of the TEE for Trusted Storage.

Figure 3-8: Architecture Overview – Biometrics



3.5.7 Cryptography and the TEE

The GlobalPlatform TEE Internal Core API Specification ([TEE Core API]) provides a comprehensive set of cryptographic functionality requirements.

The majority of these must be implemented on any TEE, though there are some exceptions to enable regional variations that may not be acceptable in some parts of the world.

Other GlobalPlatform TEE specifications will generally make use of cryptographic capabilities based on those defined in [TEE Core API], and thereby provide consistency, align with additions to the base set, and minimize the overall footprint.

It is not the role of the GlobalPlatform TEE specifications to guide the reader in determining which cryptographic capabilities may be safe for their purposes, and the GlobalPlatform TEE specifications recognize that in some cases the use of weak cryptography by a TA may be better than the use of that same cryptography by an application outside of a TEE.

GlobalPlatform does provide recommendations for best practices and acceptable cryptography usage. These can be found in GlobalPlatform Cryptographic Algorithm Recommendations ([Crypto Rec]), and relevant sections of that document MAY be applied to the protocols and APIs offered by the GlobalPlatform TEE specifications. As always, the developer should refer to appropriate security guidelines.

Future GlobalPlatform TEE configurations may limit the full set of cryptographic options, to enable smaller TEEs to be built for goals such as constrained IoT devices.

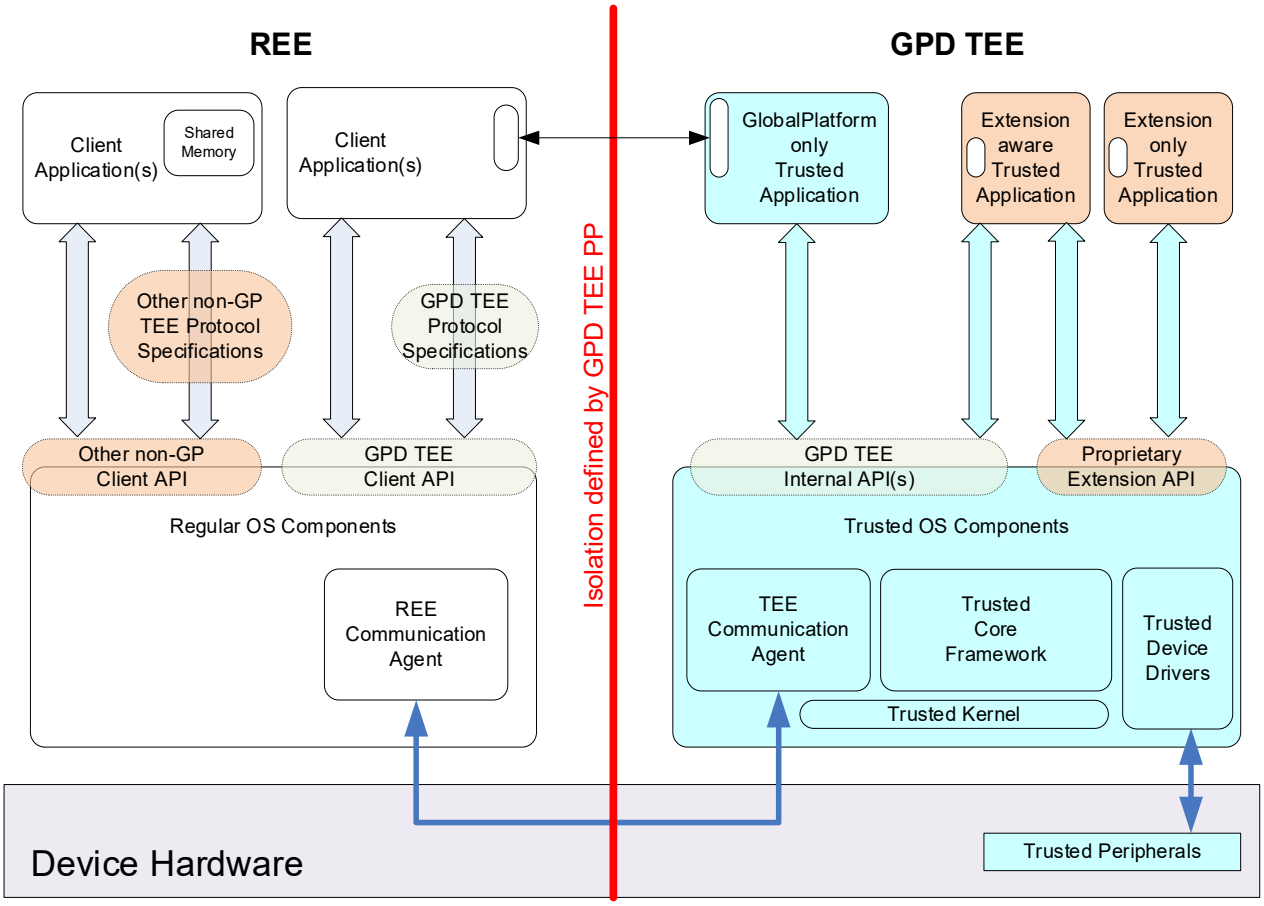
3.6 Variations of TEE Architecture Found on Real Devices

Real devices contain extensions to the basic TEE architecture and can potentially house multiple TEEs. The GlobalPlatform TEE Protection Profile ([TEE PP]) requires that a TEE, including its proprietary extensions, is isolated from other environments including other TEEs.

3.6.1 A GPD TEE Can Have Proprietary Extensions

A compliant GPD TEE can offer additional APIs to Trusted Applications and can offer other access methods to REE applications. This allows flexibility in implementation in special markets, and provides a route for growth of the GlobalPlatform TEE specifications as new APIs are found to be useful and hence adopted by GlobalPlatform as new TEE specifications.

Figure 3-9: Compliant GPD TEE with Proprietary Extensions



Please note:

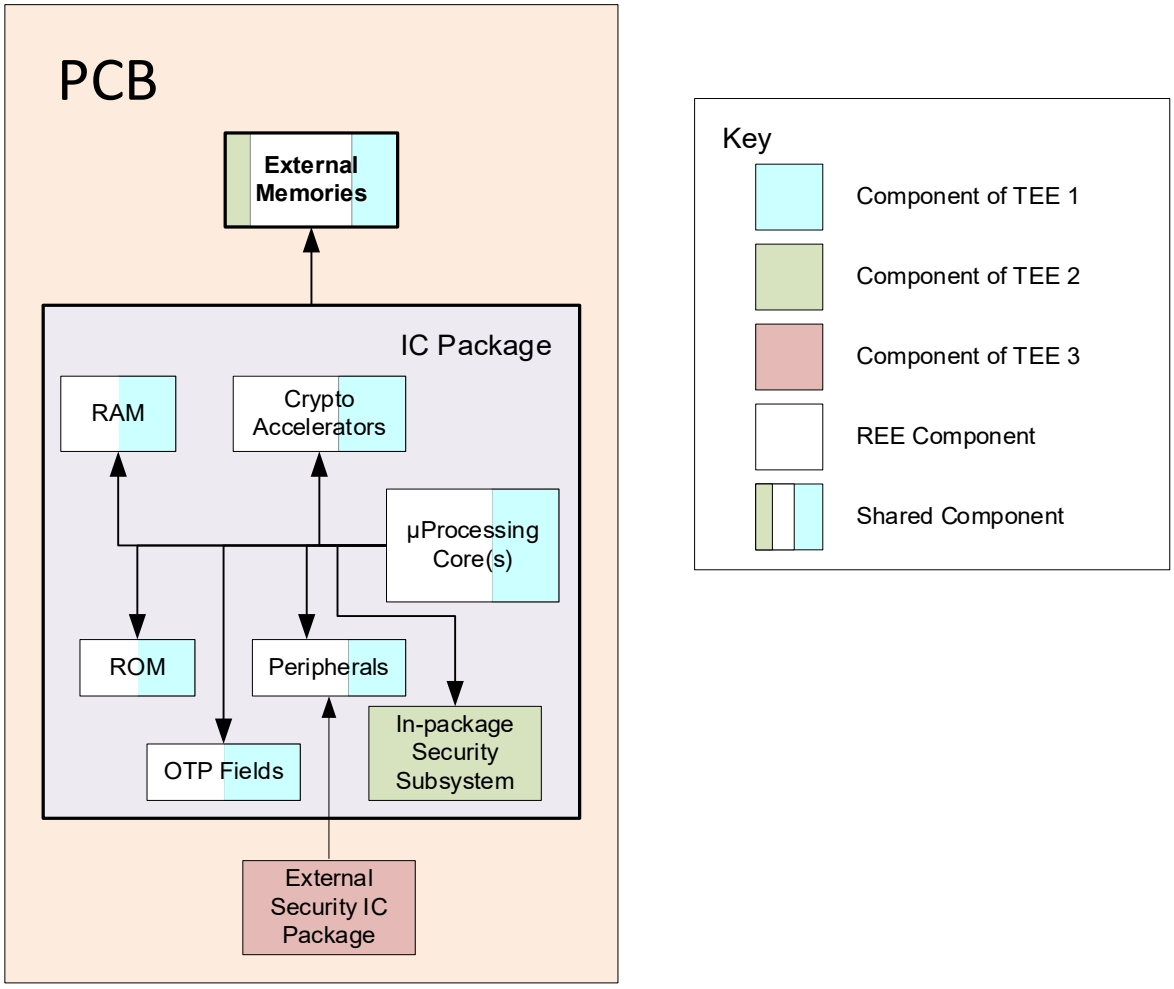
- This is one example configuration of a proprietary extension of a compliant GPD TEE, and other configurations can exist.
- There is no specified limitation on the number of OSes in a REE or the number of TEEs in one device.
- Shared Trusted Peripherals (as illustrated and described in Figure 3-1) are possible in the configuration shown in Figure 3-9.

3.6.2 A Device Can Have Many TEEs

There is no specified limitation on the number of TEEs in a device. The TEE Client API provides a methodology for a REE application to communicate to a specified GPD TEE.

For example, a device can have hardware such as that shown in Figure 3-10. The illustrated device has three TEEs, each created using a different example method. Each will have an independent set of innately trusted components and will be isolated from the other TEEs and the REE, at least to the level of the GlobalPlatform TEE Protection Profile ([TEE PP]).

Figure 3-10: Example of System Hardware with Multiple TEEs



Note that inside one GlobalPlatform TEE Protection Profile boundary there can only be one Trusted OS and hence one set of TEE resources.

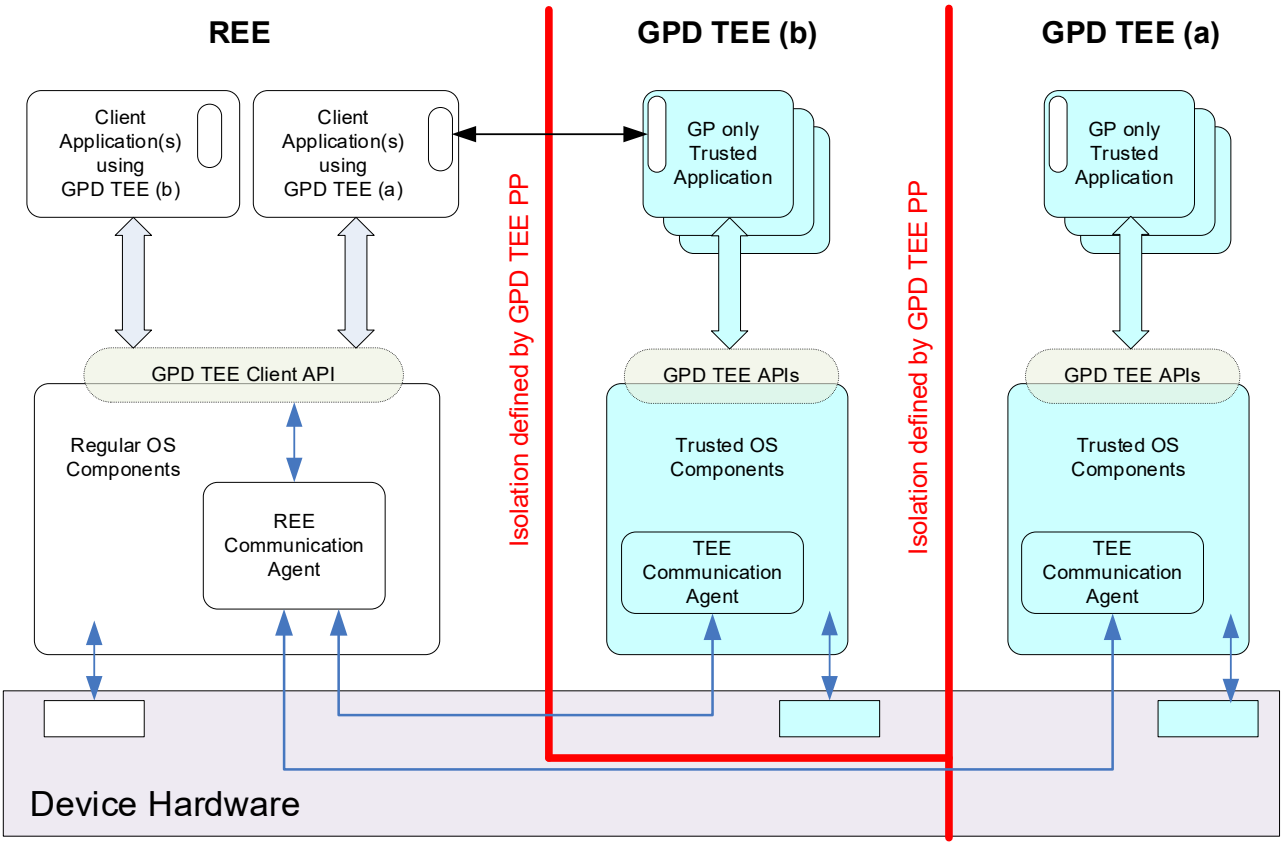
This does not prevent the GPD TEE from sharing resources with other TEEs in much the same way that it can share with the REE. For example, a Trusted User Interface is typically owned in an untrusted mode by the REE and only taken over by the TEE and put in a trusted state when needed. With multiple TEEs, such a Trusted UI would potentially be shared between all TEEs and the REE, with only one having active ownership at one time.

Communications between TEEs SHOULD be treated by a Trusted Application on the initial supposition that the endpoint is untrusted (in the same way that a TA SHOULD be designed to treat anything outside its local TEE). A TA may improve that level of trust if it knows more about the trustworthiness of the link and the other TEE.

Figure 3-11 shows an example with two GPD TEEs (i.e. two TEEs that are compliant with a GPD TEE functionality configuration and certified according to [TEE PP]). Each GPD TEE exists within its own isolation boundary and does not trust components outside of the boundary. Therefore, from the viewpoint of GPD TEE (a), GPD TEE (b) is assumed to be untrustworthy as it is not part of GPD TEE (a). Likewise, GPD TEE (b) trusts neither the REE nor GPD TEE (a).

If any OS in the device wishes to use a GPD TEE based set of innately trusted components to secure its boot, then it is up to the boot structure of that OS (i.e. its BIOS, UEFI, etc.) to choose which GPD TEE it uses. Different TEEs may be involved in the boot of each Regular OS or other EE in the system.

Figure 3-11: Multiple GPD TEEs in One Device



Please note:

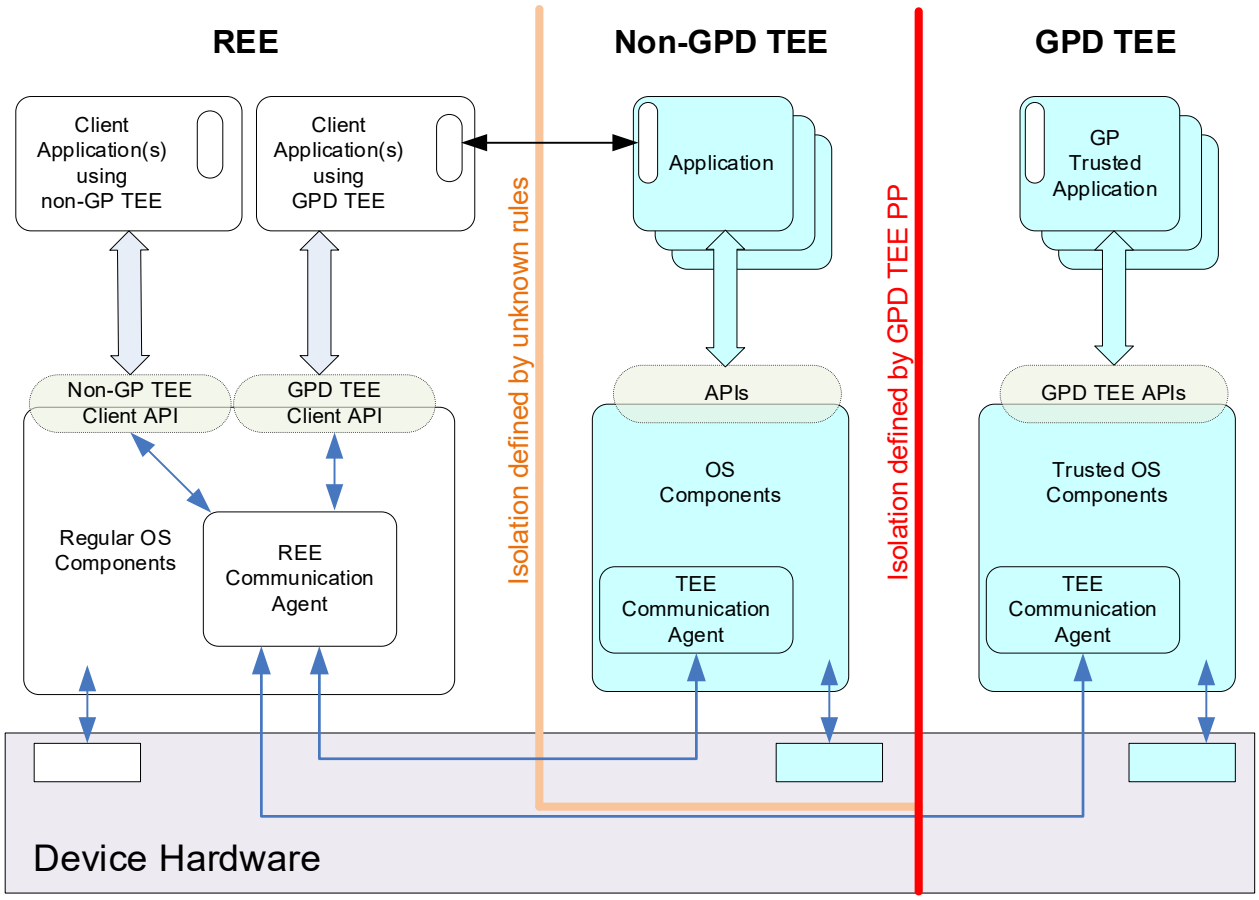
- This is one example configuration of a system with two GPD TEEs, and other configurations can exist.
- There is no specified limitation on the number of TEEs and OSes in the REE in one device.
- Shared Trusted Peripherals (as illustrated and described in Figure 3-1) are possible in the configuration shown in Figure 3-11.

3.6.3 Not All TEEs on a Device Need To Be GlobalPlatform Compliant

A device can even have environments that claim to be TEEs but are not GlobalPlatform compliant TEEs.

Clearly if such an environment does not meet GlobalPlatform specifications then GlobalPlatform cannot make any assertions about that environment; however, the environment does not raise an issue because a compliant GPD TEE SHALL be isolated from it as specified in the GlobalPlatform TEE Protection Profile ([TEE PP]).

Figure 3-12: GPD TEE alongside Unknown TEE



Please note:

- This is one example configuration of an unknown TEE alongside a GPD TEE, and other configurations can exist.
- There is no specified limitation on the number of GPD TEEs, non-GlobalPlatform TEEs, and OSes in the REE in one device.
- Shared Trusted Peripherals (as illustrated and described in Figure 3-1) are possible in the configuration shown in Figure 3-12.

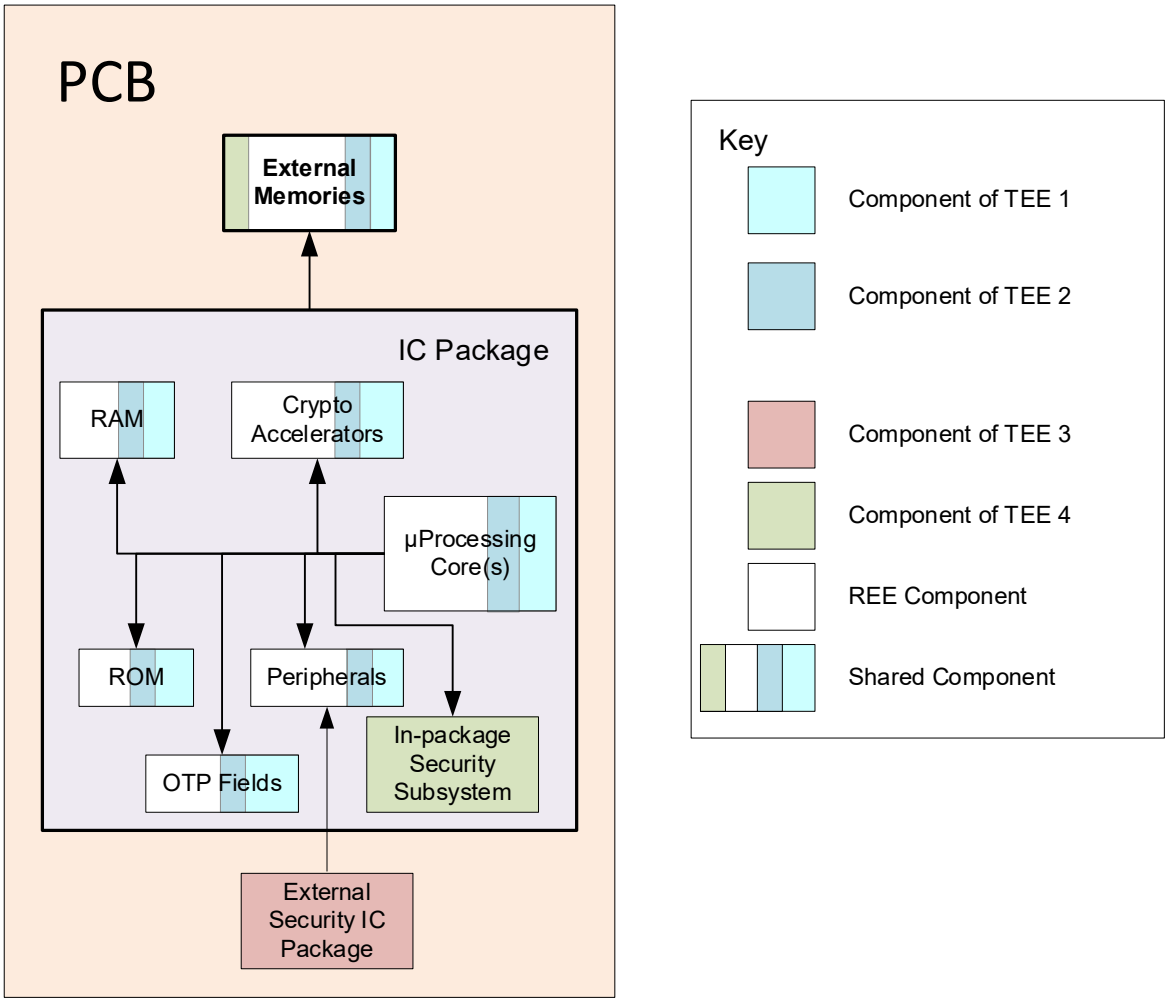
3.6.4 TEEs and TEE-enabling Hypervisors

A device can have multiple TEEs, some of which may be isolated by a suitable combination of hypervisor and hardware.

This section specifically discusses the hypervisor enabled forms of TEE. Section 3.6.4 (including its sub-sections) ignores TEEs in the device that may use other methodologies. For example, it does not describe TEEs that use structures exemplified by TEE 3 and TEE 4 in Figure 3-13. TEE 3 and TEE 4 correspond to PCB A and PCB C, described following Figure 2-3.

Section 3.6.4 (including its sub-sections) does not preclude the use of solutions using any combination of zero, one, or more instances of any of the TEEs shown in Figure 3-13.

Figure 3-13: Many TEEs including Using Hypervisor Separation



The components in Figure 3-13 represent an extension of those in Figure 2-3, introducing the hosting of multiple TEEs in one shared set of processor cores.

Please note: Hypervisor based isolation is, by definition, based on some sharing of resources. The content of section 2.2.4 (REE and TEE Resource Sharing) and section 5.4 (Transfer of Hardware Components to and from the TEE) is relevant when considering this type of system.

3.6.4.1 Hypervisor Isolation from the Hardware Point of View

Some hardware components may be assigned to specific TEEs or other Execution Environments. Some use cases require the ownership of components to be switched between Execution Environments (see sections 2.2.4.1 and 5.4).

One method of assigning such hardware assets in a modern system is using a hypervisor.

Traditional hypervisor hardware support does not meet TEE isolation requirements. To meet the requirements of the GlobalPlatform TEE Protection Profile ([TEE PP]), care must be taken that the hypervisor regulates not only access control to assets managed by the processing cores themselves, but also such access via other Bus Managers in the system.

Hardware debug is also part of the access control regime and must be similarly limited by the isolation technology. We use the term Isolator Control Software to refer to hypervisor or similar software technology used to control isolation between different Execution Environments, and “isolator” to refer to the hardware that may be used by such Isolator Control Software or by the OSes.

The example system illustrated in Figure 3-14 shows potential hardware actors and assets in a two TEE system based on a hypervisor separation. Assets of a given TEE must be isolated from actions of any Bus Manager that is not currently controlled by that specific TEE.

Figure 3-14: Example of Two Hypervisor-separated TEEs

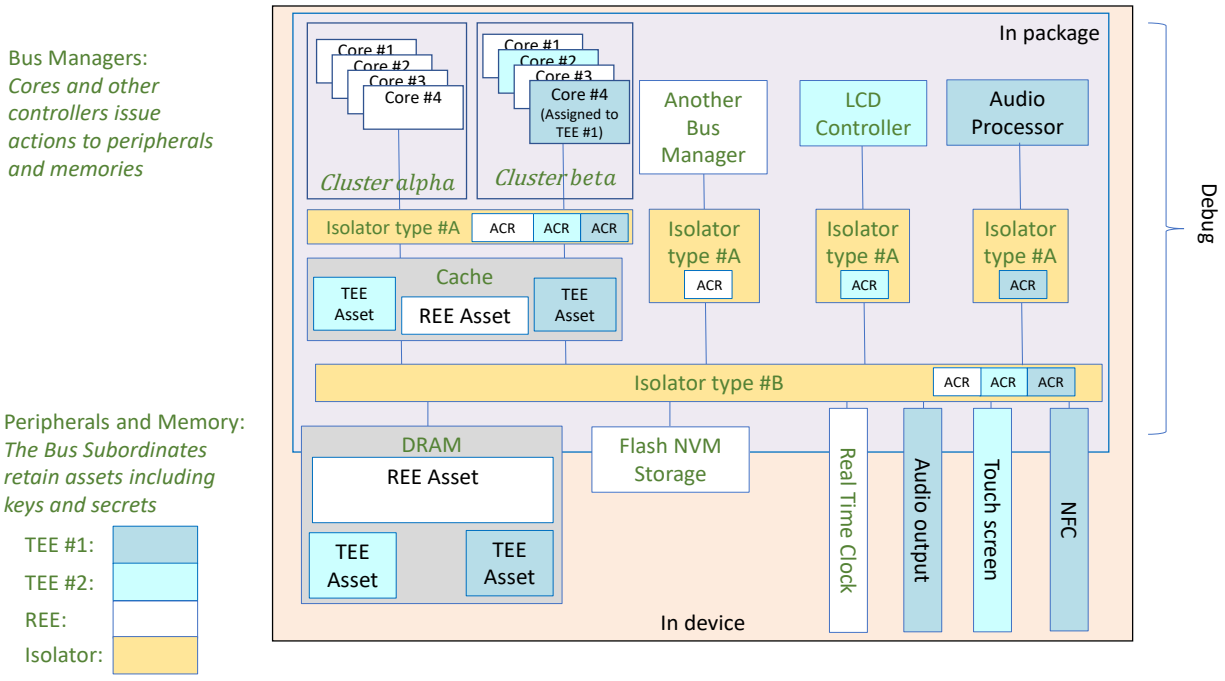


Figure 3-14 depicts an example of a package in a device with multiple clusters of Cores, and multiple other Bus Managers. The presence of these features is typical in modern SoC but as this is an example it does not preclude other arrangements with different numbers of clusters, of cores per clusters, Bus Managers, and assignments of TEE activity to particular cores or Bus Managers.

The above example shows two TEEs.

- TEE #1 that might currently have a TA doing some form of Audio DRM based on an NFC token.
- TEE #2 has a TA using the touchscreen for some tasks.

Clearly one TEE could perform both these tasks but there can be security and control reasons to have multiple TEEs; for example, if one is a locked down device OEM TEE and one is enabling third party TAs.

What is critical here, and should be reflected in real systems, is:

- TEE #1 must not be able to use ANY of its Bus Managers to access TEE #2's assets:

  E.g. TEE #1's audio processor must not be able to access the TEE #2 Asset in DRAM.

- TEE #2 must not be able to use ANY of its Bus Managers to access TEE #1's assets:

  E.g. TEE #2's LCD controller must not be able to access TEE #1's NFC peripheral device.

- And the REE must not be able to use ANY of its Bus Managers to access any asset of TEE #1 or TEE #2.

Isolator type #A

These logical components provide the means of stopping one application interfering with an OS, driver, or another application. They are typically implemented via a Memory Management Unit (MMU) or Memory Protection Unit (MPU). Each is separately associated with a Bus Manager. Where an Isolator type #A must associate with multiple Bus Managers, they form a cluster of Bus Managers, and all are typically of the same type. Each Isolator type #A stops only its associated Bus Manager, or cluster of Bus Managers, from unauthorized access.

If a single physical component can provide different rules to each Bus Manager, then it is providing multiple logical Isolator type #A components.

Isolator type #B

These logical components provide system wide isolation between all Execution Environments, by controlling access from all Bus Managers to all Bus Subordinates. While shown as a layer directly above the memory and IO assets of the system, they may have aspects throughout the system, providing isolation in the pipelines and caches of the system.

Hardware Isolation and Access Control Rules

Each hardware isolator has Access Control Rules (ACR) that it can use with or without software intervention. An example of such rules is MMU Page Tables. Each Execution Environment manages its own ACR which is configured via the appropriate Isolator Control Software (see section 2.2.4.1, Isolation of Trusted Resources). In a suitable system, Isolator type #A and Isolator type #B may be implemented in one physical component, if that one component enables the different management needs of the different Execution Environments.

The resulting implementation SHALL provide sufficient separation of the ACRs to allow the Execution Environments to maintain their required independence.

Debug Support

Debug systems shall not be allowed to bypass a TEE's asset access control. The GlobalPlatform TEE Protection Profile implies that debug in a TEE shall only be enabled by suitable mechanisms of that TEE (and not by enabling debug in other TEEs or the REE). This is reflected in the GlobalPlatform TEE TA Debug Specification [TEE TA Debug].

3.6.4.2 Isolated TEEs from the Software Point of View

From the software point of view, each TEE is composed of:

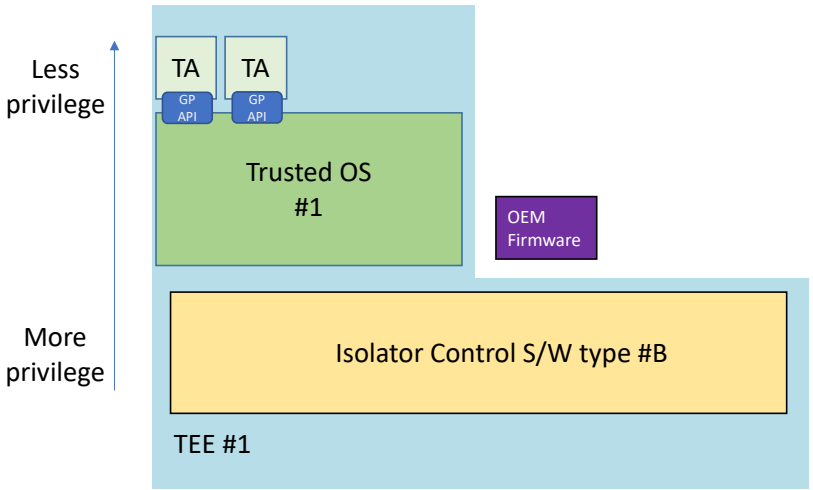
- The code, data, and keys that are used to boot the run-time TEE
- The code, data, and keys that control the part of the system on which the TEE depends to meet its Protection Profile. These are:
  - Software that may be common to multiple TEEs
  - The Trusted OS
  - Libraries that may be loaded into the TEE and provide functionality to other parts of the TEE or to TAs
  - Drivers that may be loaded into the TEE and provide functionality to other parts of the TEE or to TAs

In a complex system, with multiple TEEs, there may be a number of different software stacks (some with common components) involved in enabling a particular TEE. It is worth repeating that the ultimate binding factor in what is inside a TEE, versus what is outside a TEE, is what can interact with the assets of a TEE as defined in the GlobalPlatform TEE Protection Profile ([TEE PP]). If something can act directly on TEE assets then that something is part of the TEE.

3.6.4.2.1 A Single TEE System with a Hypervisor

Figure 3-15 shows a simple system with a GPD TEE alongside some additional software (in this case some OEM Firmware) outside of the GPD TEE's security boundary but sharing the same Isolator Control Software type #B.

Figure 3-15: Isolator Control Software Type #B



Isolator Control Software type #B

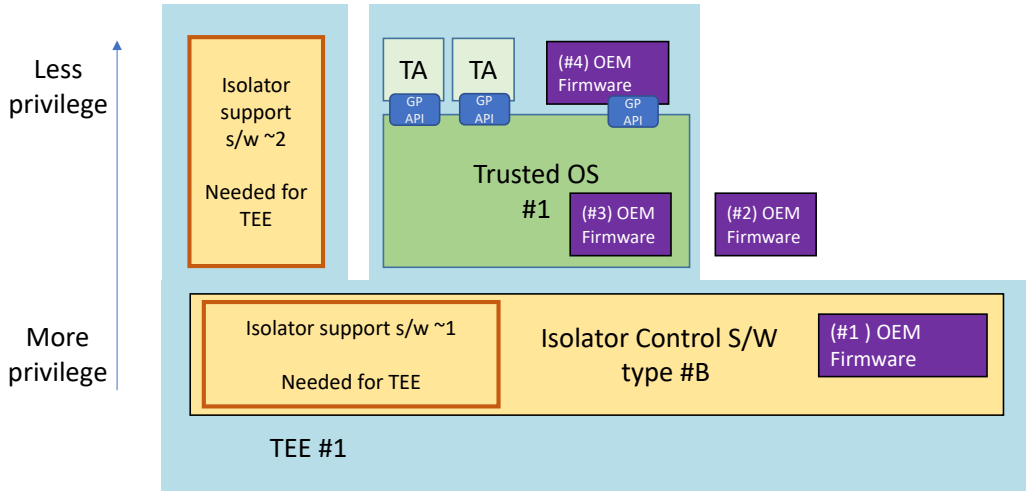
Isolator Control Software type #B must enable system level isolation between the software stacks that it hosts (at lower privilege levels) and provide communication between those stacks. In some systems the Isolator Control Software type #B may also perform additional tasks (such as allocating resources, managing resource sharing, providing para-virtualized services, etc.).

To not limit solutions by terminology, we have used “Isolator Control Software type #B” in the following example software architectures, rather than something more specific such as “Hypervisor” or “Monitor Mode”. “Isolator Control Software type #B” refers to whatever software controls Isolator type #B (e.g. a system level MMU or equivalent isolator) shown in Figure 3-14, along with any other required asset isolation mechanisms.

3.6.4.2.2 Minimizing Code with High Privilege

To reduce security risk it is desirable that any introduced code has the least possible impact on other code in the system if an issue occurs. The example system in Figure 3-16 illustrates various locations in which software may be executed when a hypervisor is added. The goal here is to discuss the locations that various typical support software packages may reside in an Isolator Control Software type #B enabled system.

Figure 3-16: Isolator Control Software Type #B and Minimal Privilege Code



Isolator support software

It is desirable for best security to minimize the amount of software running at higher privilege levels. As such, any software that necessarily is part of the Isolator Control Software should be moved from (~1) to (~2) where possible.

An example of such support software might be virtualized device services, cryptographic services, or storage service enablers.

OEM firmware

This firmware is a good example of software often associated with the TEE that must meet the balance of security risk vs. functional needs. Parts of this firmware may be found in many different locations depending upon the execution needs of that specific part:

- Integrated into Isolator Control Software type #B
  - This can be seen as an increased risk, as it increases the size of the most privileged code (Isolator Control Software type #B).
  - That risk then also impacts anything running on top of that Isolator Control Software type #B.
- Running in its own software stack alongside the Trusted OS software stack

        - Potentially complex because it has to provide all its own services which it might otherwise get from an OS.
    - Running as part of an OS's provided software (e.g. privileged driver)
        - A compromise where it impacts the privileged code size of the OS but has enabling OS services.
        - That risk then also impacts anything running on top of that OS.
    - Running as a user space application (or user space driver) on the Trusted OS
        - This provides the best protections for the rest of the system, and the potential of full portability.

In all cases, as we move further from the Isolator Control Software type #B there may be a disadvantage due to loss of flexibility (due to restrictions imposed by the underlying layers) and reaction speed (due to the additional layer transitions required to communicate to other parts of the system).

The following examples are provided to show a range of software configurations that may enable multiple TEEs separated by Isolator Control Software type #B and are not intended to be restrictive. Other variants may be applicable as long as they meet the GlobalPlatform TEE Protection Profile ([TEE PP]).

3.6.4.2.3 Various Examples of Multiple TEE Systems

Here we show a number of different TEE structures in a multi-TEE system. These are component diagrams, showing which parts of a system are considered part of the various EEs in the system. They are examples and a real system may have more TEEs and other software stacks alongside those shown. Figure 3-17 shows the software components enabling TEE #1 (inside the blue area). It is worth noting that the Isolator Control Software type #B may provide TEE-like protection to other vertical stacks that are not considered GPD TEEs due to their lack of compliant functionality support, but which may serve useful purposes in the system.

Figure 3-17: TEE #1 Integral in the IC Package and Providing TEE Services to the Device

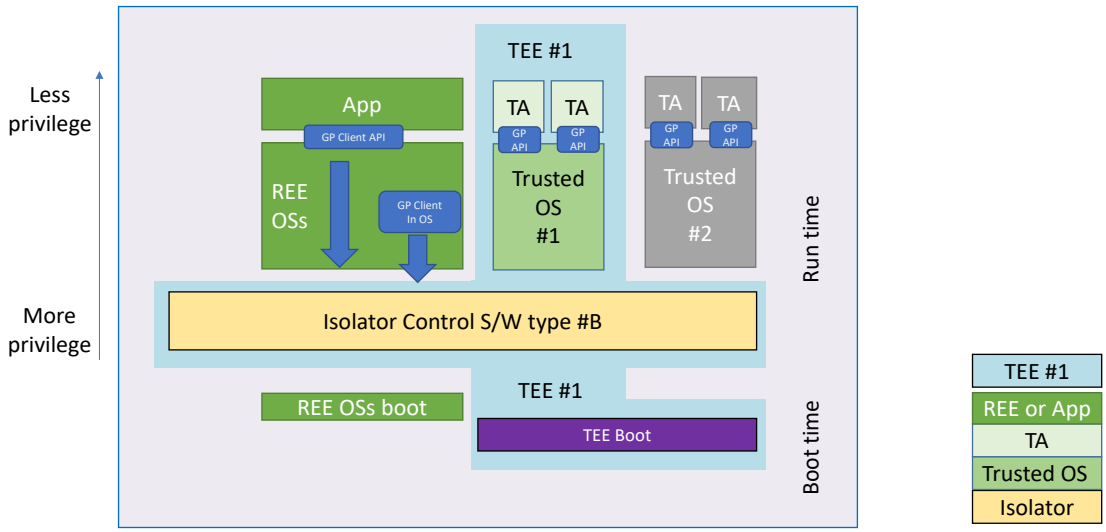
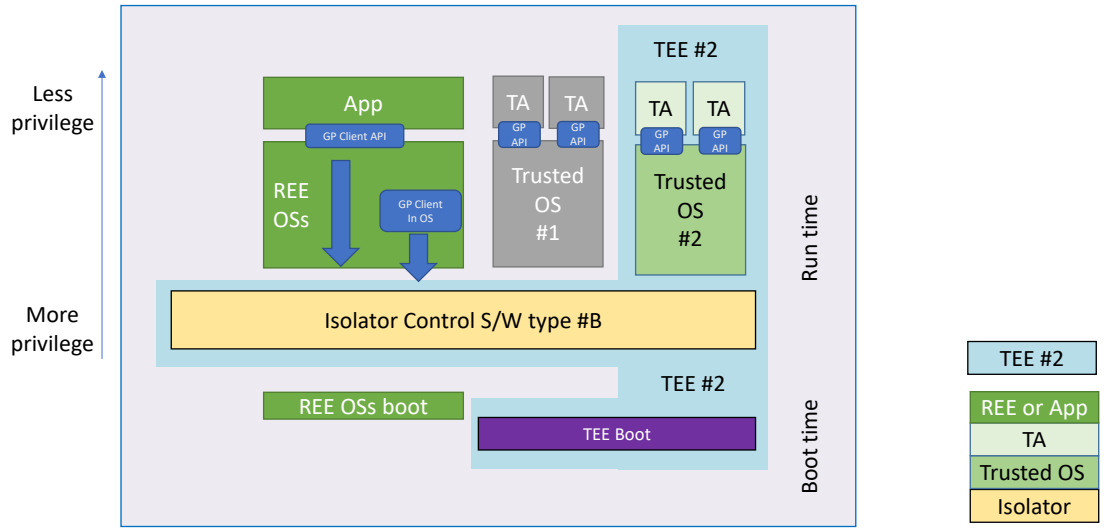


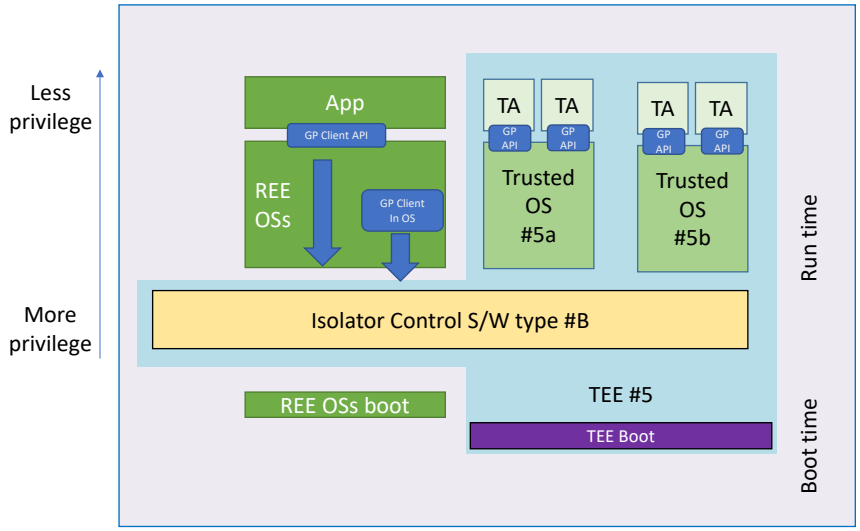
Figure 3-18 shows the software components enabling TEE #2. It can be seen that the boot and Isolator type #B components are shared with TEE #1. It should also be remembered that these are simplified diagrams and that other components such as the isolator support software may also be part of the TEE.

Figure 3-18: TEE #2 Integral in the IC Package and Providing TEE Services to the Device



Multiple Trusted OSes may gain security via a level of isolation between them, but due to interdependencies may form one TEE from the security analysis point of view. An example of this is where one Trusted OS is quite closed and providing hardware specific services, and the other Trusted OS is more generic but takes advantage of some of those services. This results in the following system:

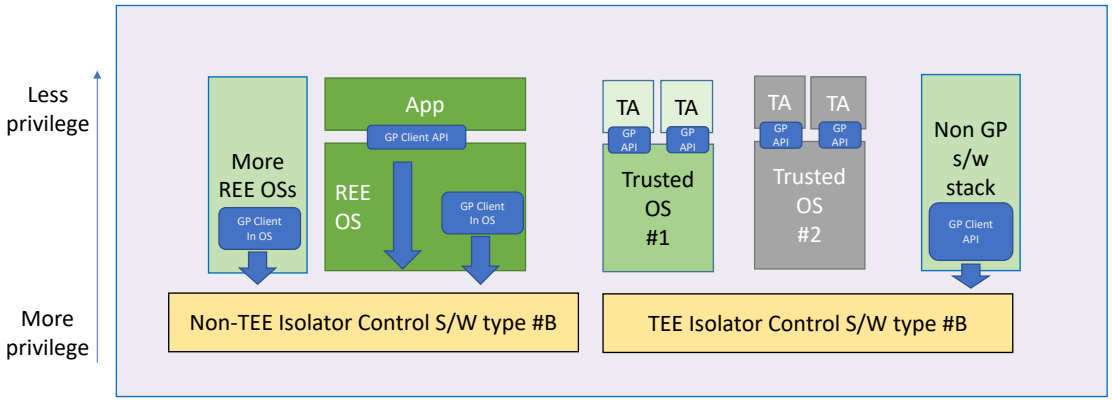
Figure 3-19: Trusted OS #5a and #5b Co-operating to Provide the Functionality of TEE #5



The TEE and OS numbering in the example does not indicate the number of TEEs in such a system but is just a reference to distinguish these from other TEE descriptions in this document. In addition, there is no actual requirement that both Trusted OSes be able to execute a GlobalPlatform compatible TA.

It is worth noting that to reduce the trusted computing base it may be worth splitting the Isolator Control Software type #B so that a more minimal one can serve the TEEs and other critical code stacks, while a richer one is available to the REEs. This is shown in Figure 3-20.

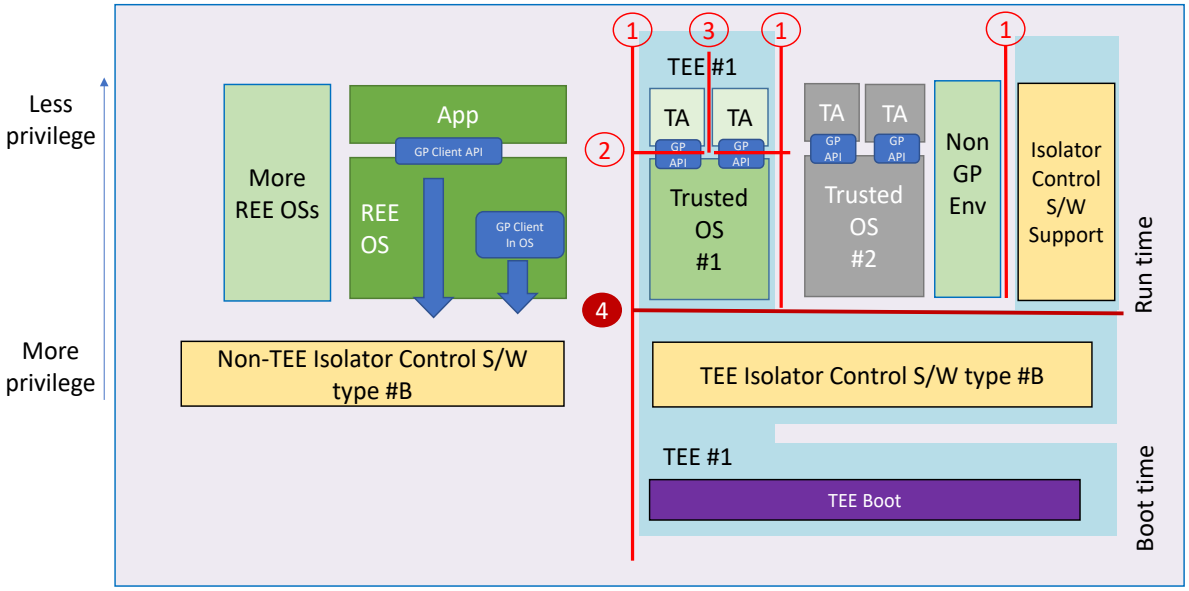
Figure 3-20: More than One Isolator Control Software Type #B in Device



3.6.4.2.4 Isolation Boundaries in a Hypervisor-based System

The use of hypervisor technologies requires an additional isolation boundary, providing access control, to be considered in the TEE design.

Figure 3-21: Isolation Boundaries



In traditional [TEE PP]-compliant systems, we have the following isolation boundaries.

(1) The TEE must guarantee an access control boundary between the TEE and everything outside of the TEE.

This stops external actors from interfering with assets inside the TEE.

(2) The TEE must guarantee an access control boundary between the TA and the rest of the TEE.

This stops rogue TAs from interfering with the rest of the TEE.

(3) The TEE must guarantee an access control boundary between the TAs.

This stops rogue TAs from interfering with other TAs.

In a system where multiple TEEs rely on the trustworthiness of some common Isolator Control Software type #B, then the unshared parts of those also need to be isolated such that they cannot make unauthorized changes to the shared part. This means:

(4) The TEE must guarantee an access control boundary between the components unique to this TEE and components shared with other TEEs. This access control is typically implemented by the Isolator Control Software type #B.

This stops rogue TEEs from interfering with other TEEs.

Other isolation boundaries may also exist in a particular implementation.

3.6.5 Executing alongside Other Environments: Trust vs. Respect

It is a given that a Trusted OS can trust other components that make up the TEE it is running in – namely boot code and isolators.

The Trusted OS *respects* claimed security requirements made by entities outside of the TEE; however, this respect does not imply trust in those entities to respect the TEE's security requirements.

For example:

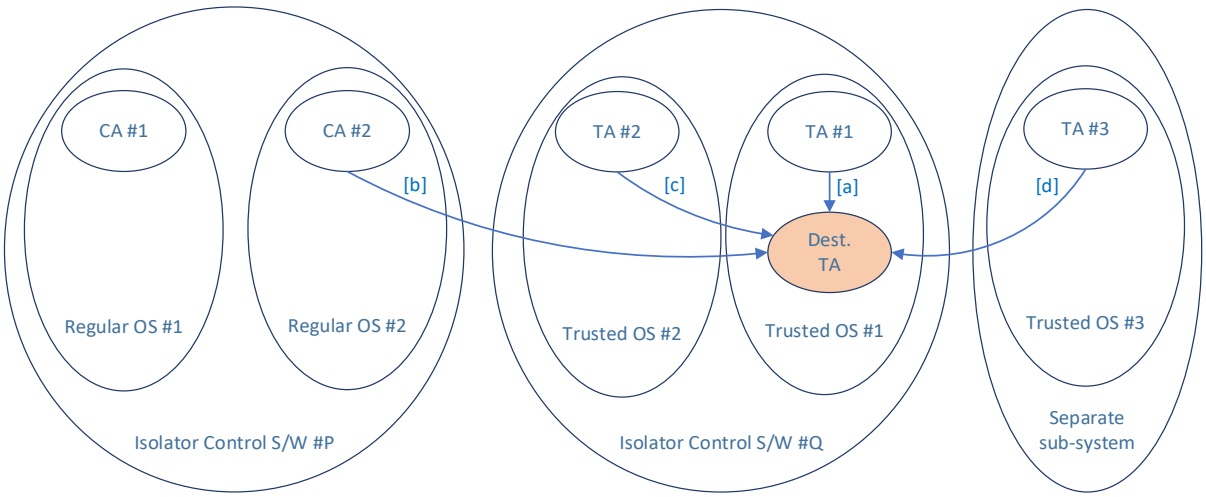
- Respect – If an area of address space is configured in the overall device to be private and not for the use of a particular TEE, then the TEE should respect this configuration even if no other system enforces this.
  - An exception may be made if the configuration would impact the secure functioning of that TEE.
- Trust – Respecting such security configurations should not be taken to imply that the TEE *trusts* either who made the configuration or what the configuration represents; however, it enables others to reason about the behavior of the TEE with respect to 'their' resources.

3.6.6 Communicating with Other Environments: Trust vs. Respect

Generally, a TA or Trusted OS acts in response to requests from other entities on the device, though sometimes those entities are just a channel for off-device communications.

The Trusted OS should generally *respect* claimed identities made by entities outside of the TEE but should not implicitly *trust* them.

Figure 3-22: Communication between Applications in Various Execution Environments



When considering communication, a session from a trusted/client process X (which in this example may be any of CA #1/2 or TA #1/2/3) to a Trusted Application (Dest. TA) may pass through several Execution Environments and isolators that are not implicitly trusted; however, each may provide contextual information related to the call. Whilst information from outside the target TEE cannot be trusted in an absolute sense, the recipient of the message may have external reasons to trust and/or respect this information within a given context.

For example:

- A Trusted Application having a session from another Trusted Application in the same Trusted OS can trust the client identity to be accurate, as they are both part of the same TEE (case [a] in Figure 3-22) but it has to make its own determination of the trustworthiness of the source TA functionality.
- A Trusted Application having a session from a REE identified as originating from CA #2 (case [b]) can respect a related security policy.
  - E.g. a policy may indicate that information contained in the message from CA #2 is not to be shared by the TA on any communication session with CA #1. It is accepted that the Trusted Application is constrained on its ability to apply this policy by outside factors; for example, a REE compromise could lead to the message content being rerouted before entering the TEE, or the CA identity being falsified, both of which are beyond the TA's control.
- A Trusted Application having a session from a TA in a different Trusted OS may be able to trust the origin Trusted OS identity if both share the same isolator (case [c]), but this does not imply that the origin TA identity can be trusted.
- A Trusted Application having a session from a TA in a different Trusted OS that is not using a shared isolator cannot trust the origin (case [d]).
- A session from a TA in another Trusted OS may, in future, identify the calling TA, Trusted OS, and/or Isolator(s) using an extension or update to the mechanism used to identify a client.
- External means, such as cryptographic signatures or secure channels, may be used to establish additional trust at either a TA-to-endpoint level or a Trusted-OS-to-other-environment level.

Contextual information, such as the routing of session messages, can be *trusted* when it is provided by a component within the TEE that hosts the session endpoint TA, but should only be *respected* if the provider of the information is outside of the TEE. Table 3-3 shows the trusted and untrusted parts of the routing and identity information for the scenarios from Figure 3-22.

Table 3-3: Trust of Communication

Case	From	To	Untrusted Path (outside destination TEE)	Trusted Path (within destination TEE)
[a]	TA #1	Destination TA		TA #1 → Trusted OS #1 → Dest. TA
[b]	CA #2		CA #2 → Regular OS #2 → Isolator #P → Bus	Bus → Isolator #Q → Trusted OS #1 → Dest. TA
[c]	TA #2		TA #2 → Trusted OS #2	Trusted OS #2 → Isolator #Q → Trusted OS #1 → Dest. TA
[d]	TA #3		TA #3 → Trusted OS #3 → Bus	Bus → Isolator #Q → Trusted OS #1→ Dest. TA

Notes

- The TEE Client API, in combination with the TEE Internal Core API identity methodology (ref [TEE Core API] section 4.1.1 and section 4.2.2), identifies properties of the source CA or TA but does not provide properties relating to the source OS/Isolator.

- The TEE Internal Core API provides TA-TA communication within a single TEE, and implementations may enable the TEE Client API to be used from within a TEE to enable communication with TAs in other TEEs. A future revision of [TEE Core API] may add integrated TA-TA communication across TEEs.

4 TEE Management

Management of the TEE and Trusted Applications running in the TEE is described in the GlobalPlatform TEE Management Framework specification ([TMF]). The remote management life cycles of Trusted Applications, GlobalPlatform style management Security Domains, and the TEE itself are also detailed in that specification.

An introduction to TEE Management is provided below. For clarity, the latest versions of each of the reference documents should be read in place of these introductions.

4.1 Overview of TEE and TA Management

Each GlobalPlatform style Security Domain (SD) has a nominal off-device “owner” with rights to control SDs and TAs directly and indirectly below the given SD. The exception to this is when the child SD is a root SD (rSD), because root SDs form a management isolation boundary which limits parental interference.

The TEE Management Framework provides means to securely manage Trusted Applications in a TEE. The following three layers are described.

Administration operations

- Defines the set of supported operations to manage Trusted Applications and Security Domains, the conditions of use and the detailed behavior of each operation.

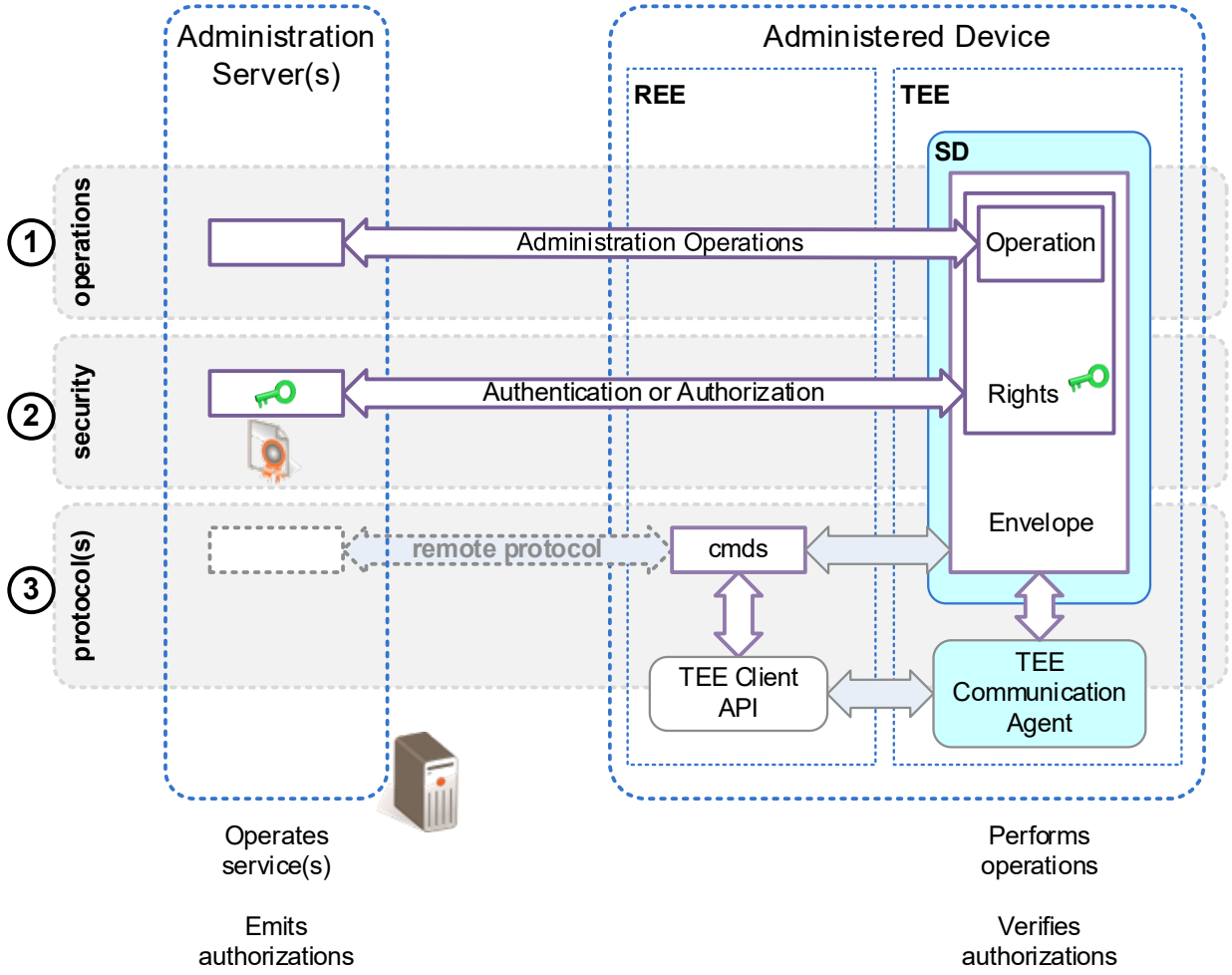
Security model

- Defines who the actors are and how the different business relationships and responsibilities can be mapped on the concept of Security Domains with privileges and associations.
- Defines the security mechanisms used to authenticate the entities establishing a communication channel, to secure the communication, and to authorize the administration operations to be performed by Security Domains.
- Defines schemes for key and data provisioning and describes the associated key management.

Protocols

- Defines the command set (over the TEE Client API) to be used to perform administration operations.
- Defines the command set to be used to establish a secure session with a Security Domain.

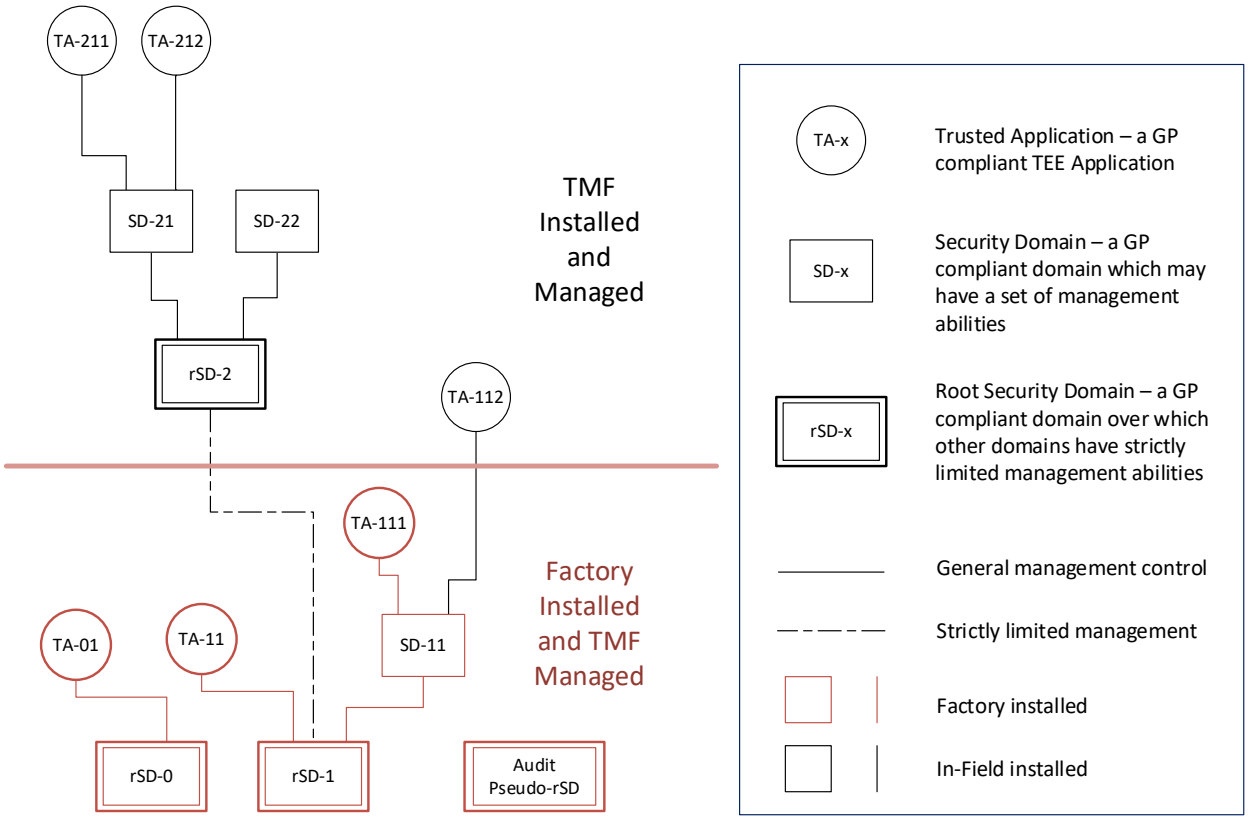
Figure 4-1: TEE Management Framework Structure



4.2 Overview of TA Management Hierarchies

The following diagram shows an example of possible management relationships between Security Domains and between Security Domains and Trusted Applications enabled by the GlobalPlatform TEE Management Framework specification ([TMF]).

Figure 4-2: Security Domain Management Relationships



The above diagram is just an example of how a management structure can be developed on a platform.

In Figure 4-2:

- rSD-1 is the direct parent of TA-11.
- rSD-1 is the indirect parent of TA-111.
- The owner of rSD-1 can potentially control any SD-1* or TA-1* but cannot control any of the other current SDs on the example platform, due to rSD-2.
- The owner of rSD-1 can install rSD-2 but cannot interact with any of rSD-2's direct or indirect children and is strictly limited in the operations it can perform on rSD-2.
- From a remote entity point of view, all the rSDs can be considered Roots of Trust because there is no entity which can vouch for their "boot" state but they can vouch for the existence of all their child SDs and TAs.

There are some exceptions to the above rules, such as with regard to factory reset. For more detail, see [TMF].

[TMF] places no restriction on the number of SDs (including rSDs) or TAs that can be installed in the factory or in the field. Particular platform implementations and GlobalPlatform TMF configurations have limits on available storage resources and these limits affect the numbers of TAs and SDs that might be deployed on that platform.

[TMF] defines various SD and TA management operations such as installation, removal, updating, blocking, and personalization. Particular platforms and GlobalPlatform TMF configurations can choose to restrict the availability of certain TEE Management Framework management operations on that platform and similarly particular Security Domains can choose to limit the operations available to their child Security Domains.

4.3 Overview of ASN.1 Profile, X.509 Profile, and Cross-profile Mapping

The TEE Management Framework (TMF) defines a security model for the administration of GlobalPlatform compliant Trusted Execution Environments (TEE), and the administration and life cycle management of Trusted Applications (TA) and their corresponding Security Domains (SD).

- *TEE Management Framework (TMF) including ASN.1 Profile* ([TMF]) defines extensive commands for administration and life cycle management based on ASN.1 message format.
- *TMF: Open Trust Protocol (OTrP) Profile* ([TMF OTrP]) defines essential TEE management messages and essential TA and SD life cycle management messages based on JSON message format.

A TEE MAY support the ASN.1 Profile (as described in [TMF]), the OTrP Profile (as described in [TMF OTrP]), or both.

- Trusted Service Managers (TSMs) or Outside World Entities (OWEs) that support the ASN.1 Profile use ASN.1 Profile commands to administer the TEEs on authorized devices.
- OWEs that support the OTrP Profile use OTrP messages to administer TAs and SDs on authorized devices.
- A Service Provider (SP) may choose between a TSM or an OWE for the life cycle management of its TAs in TEEs.

The execution of a TMF OTrP request message SHALL provide the same result that the equivalent TMF ASN.1 Profile command would have achieved.

A TEE that already supports the ASN.1 Profile may integrate OTrP Profile support using one of the following methods:

- Implementing OTrP Profile functionality directly into the TEE Trusted OS.
- Implementing an OTrP Mapping Implementation Layer (OMIL) that reuses the existing ASN.1 Profile support. OMIL needs to store secrets and state. It must therefore be implemented within the same Security Domain as the TEE. It may be implemented as a combination of Client Application and Trusted Application, but the Client Application must not have access to any data that may be used to compromise the system.

The *TMF: Open Trust Protocol (OTrP) Mapping* specification ([TMF OTrP Mapping]) focuses on the latter method and recommends the implementation details for OMIL, its responsibilities, as well as details on how to map OTrP request messages to ASN.1 Profile commands and ASN.1 Profile response output back to OTrP response messages. It is assumed that OMIL has no special access to the TEE – that is, it can only issue TEE Core API and TMF commands using the TEE Client API.

4.4 Roots of Trust and the Trusted Management Framework

The GlobalPlatform definition of whether a module is a Root of Trust depends upon whether other entities can report a measurement of the module under consideration. Since every SD is capable of acting as an Audit SD, they are all optionally capable of reporting the TEE properties *gpd.tee.trustedos.implementation.binaryversion* and *gpd.tee.firmware.implementation.binaryversion*. These values MAY contain measurements of the Trusted OS and other underlying firmware in the TEE.

- If the TEE does not report a measurement of its boot state through this interface, then the rSDs are Roots of Trust for the [TMF] services because they impose integrity requirements on other security domains and the contents of those domains.
- If the TEE does report this measurement, then the measuring boot code is the Root of Trust and the [TMF] services are considered provided by security modules vouched for by the RoT boot stage that made that measurement.
- More information about TEE management can be found in [TMF].

4.5 Configurations of the TMF ASN.1 Profile

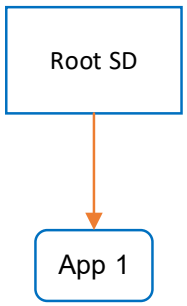
TEE Management Framework (TMF) Initial Configuration ([TMF ASN.1 Config]) defines two TMF ASN.1 profile configurations for particular sorts of devices. One configuration is focused on enabling the most constrained IoT devices, and the other specifies a minimum set of abilities for everything else. This enables those interested in developing and managing TAs to understand the minimum expectations on the sort of TA and SD management structures that might be created on those devices.

4.5.1 Model A Is the Constrained IoT Device Design

Model A provides the minimum functionality to remotely manage a device that has one set of TAs (App1 below) in the TEE.

It has a very simple relationship model that needs to be tested for it to pass compliance.

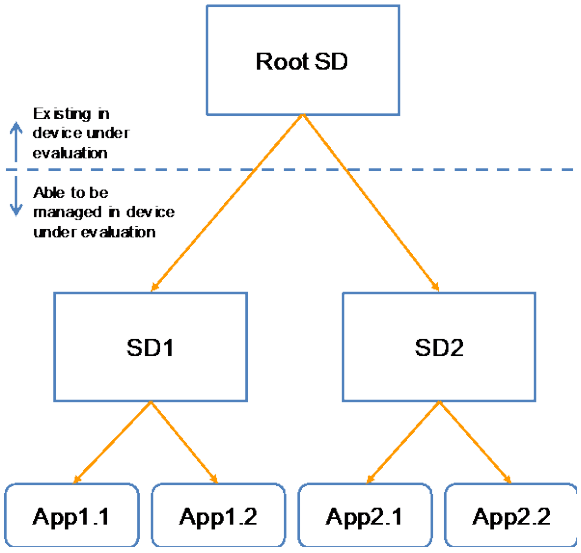
Figure 4-3: TMF ASN.1 Configuration Model A – Only One Root SD



4.5.2 Model B Is for More Complex Devices

The following shows the minimum functionality of a Model B configuration and defines the testing limits that need to be validated for such a configuration to be validated.

Figure 4-4: TMF ASN.1 Configuration Model B – One Root SD and One Level of SD



Model B configurations SHALL find the following pre-existing in the device presented for evaluation:

- A root SD (rSD, as described in [TMF] section 4.1.3.3)

Model B configurations SHALL, subsequently:

- Permit the installation of two child SDs directly associated with the root SD.
- Permit that the child SDs are installed and managed with SD Management and TA Management privilege.
- Permit the installation and management of two TAs (App in Figure 4-4) in each of the two child SDs.

Vendors SHALL provide credentials or other materials to enable the creation of TMF commands required to support this model.

4.6 TMF OTrP Profile

The GlobalPlatform TEE Management Framework (TMF), as described in *TEE Management Framework including ASN.1 Profile* ([TMF]), defines standardized methods to administer a Trusted Execution Environment (TEE) from outside of the TEE. These methods use ASN.1 message formatting to describe the message properties and associated authorizations.

The TMF Open Trust Protocol Profile (OTrP Profile, as described in [TMF OTrP]) provides an alternative framework for the administration of selected TEE management operations and of Trusted Applications (TAs) and their corresponding Security Domains (SDs). These methods are based on JSON message formatting to describe the message properties and X.509 certificates describing the authorizations.

Figure 4-5: TMF OTrP Architecture

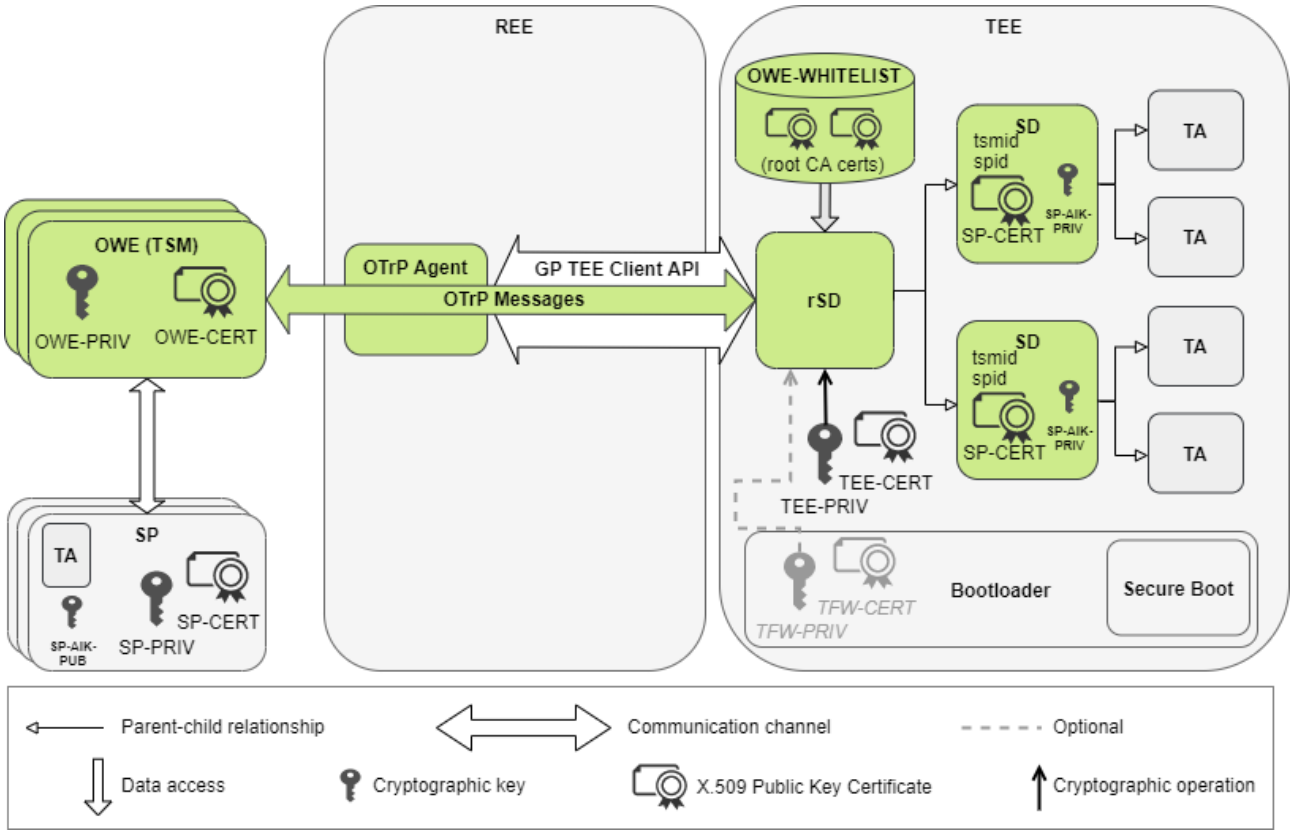


Figure 4-5 shows an architectural overview of OTrP Profile where one or more Outside World Entities (OWEs) interact with an end user's device using OTrP messages. An OWE is similar to a Trusted Service Manager (TSM), which is responsible for the life cycle management of Trusted Applications (TAs) running on TEEs of devices. Service Providers (SPs) rely on OWEs for distribution and life cycle management of their TAs in their users' devices.

An OTrP Profile compliant TEE SHALL have at least one root Security Domain (rSD), to which OTrP messages are sent through the device REE. If more than one rSD exists, the device SHALL have one rSD as the default rSD to which OTrP messages SHALL be sent unless a target rSD is indicated in the messages. OTrP messages follow a request-response pattern, where an OWE requests an operation and the TEE SHALL respond to the request. An OWE SHALL always initialize an OTrP session with a TEE by requesting the Device State Information (DSI) of the TEE. A TA is always installed in the context of a Security Domain (SD). In essence, an SD and a TA have a parent-child relation; i.e. the TA is a child node of the SD. Furthermore, in OTrP Profile, SDs SHALL be directly associated with the rSD; i.e. the rSD is the immediate parent of its child SDs. An OTrP Agent running on the REE SHALL be responsible for channeling to the OTrP messages between OWEs and relevant rSDs using GlobalPlatform TEE Client API interfaces.

4.6.1 TMF OTrP Configurations

GlobalPlatform TMF: OTrP Profile Initial Configuration ([TMF OTrP Config]) describes the following set of configurations:

Config1: Essential – minimum implementation requirements for TMF OTrP functionality

- This requires the device to provide functionality equivalent to the TMF ASN.1 Model B (see section 4.5.2) excluding those parts enabled by Config2 (below).

Config2: Extension – implementation requirements for additional, selected TEE management functionality

- This adds the OTrP interface to provide a discovery mechanism like the Audit mode in TMF ASN.1 profile functionality.
- This enables a TEE to perform a factory reset function, restoring TEE TAs to their factory state or removing them if they were not part of the current factory image.

Vendors SHALL implement Config1 alone to claim TMF OTrP Profile compliance.

Vendors MAY implement Config2 in addition to Config1 but SHALL NOT implement Config2 alone.

4.7 OMIL and OTrP to ASN.1 Profile Mapping

The *TMF: Open Trust Protocol (OTrP) Mapping* document ([TMF OTrP Mapping]) specifies a protocol mapping between TMF OTrP Profile messages and TMF ASN.1 Profile commands, which enables a device already supporting the ASN.1 Profile to potentially install a TA or additional Trusted OS capability that would enable it to use OTrP profile messages.

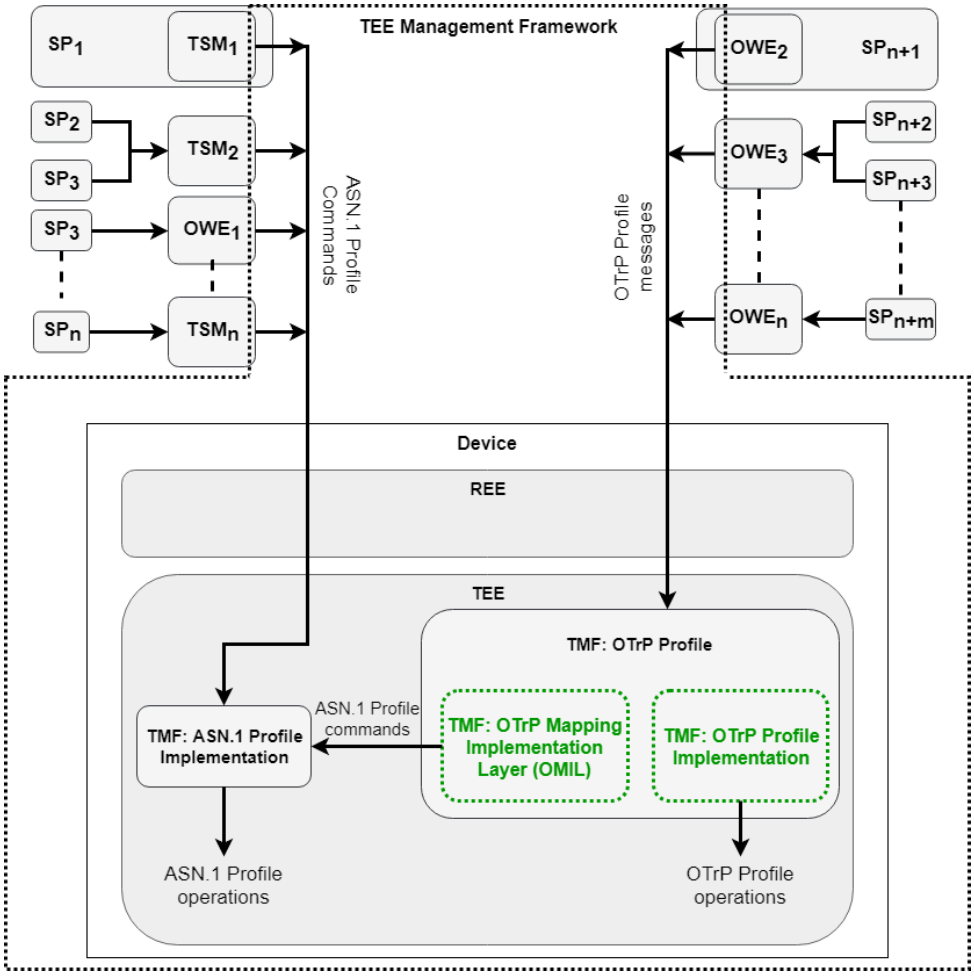
The document specifies how TMF OTrP request messages received by a TEE are mapped to TMF ASN.1 Profile commands and how TMF ASN.1 response output is mapped to TMF OTrP response messages. Direct mapping is not mandatory – that is, incoming request messages from the OTrP Profile are not required to be converted to the ASN.1 Profile – but this is a possible realization where appropriate.

To enable description of this interworking we have introduced a component called an OMIL (OTrP Mapping Interface Layer). This is presented purely to enhance description of the interfaces; implementations may use a separate component, a component integrated with a current implementation such as a TMF ASN.1 Profile handler, or any other method that meets the security and functional requirements of the overall system.

While [TMF OTrP Mapping] is written from the point of view of adding the functionality of the newer TMF OTrP specification to a system using the older TMF ASN.1 Profile specification, it can equally be used as a guide to the reverse mapping, though some limitations in the TMF OTrP specification may not enable 100% coverage.

The effect relationship created by this mapping is shown in Figure 4-6, which depicts an overview of a TEE that supports both the ASN.1 Profile and the OTrP Profile.

Figure 4-6: TEE with Joint ASN.1 and OTrP Managements Enabled via OMIL



5 TEE Implementation Considerations

The TEE and its capabilities may be closely coupled to the capabilities of the REE and the state of the device it resides in. It is therefore important for the developer of REE Client Applications, and even the Regular OS itself, to understand the availability of the TEE capabilities, along with the general security states (and hence vulnerabilities) that can be found in typical devices. Toward that end, this chapter lists some of the possible device states and discusses the notions of *boot time environment* and *run-time environment*. Some clarifications are given regarding dependencies and the availability of TEE functionalities with respect to the Regular OS.

5.1 Device States

Devices implementing a TEE can be found in a number of states that are not defined in GlobalPlatform specifications, but that are still useful for the developer to understand.

Devices implementing the TEE provide trusted mechanisms to control the corresponding security environments and transitions.

The specific implementations and characteristics of these and other similar states are up to the device manufacturers and the OEMs.

Examples of some such states:

- Devices in manufacturing, which can offer neither security nor functional compliance at various stages of their creation
- Development devices, which might have reduced security but provide TEE compliant functionality
- Production devices, which provide TEE compliant functionality and security
- And finally, devices that have somehow failed, and which will still block access to TEE held user data, while enabling various levels of debug access through secure mechanisms

5.2 Boot Time Environment

The term “boot time” refers to the time frame from the reset/power-up of the underlying hardware to the time an operating system has completed its initialization and loading. Based on this definition, boot time software also includes any firmware/ROM code that takes over the control of execution after the device is reset.

The integrity of the initial trusted boot code is intrinsically guaranteed. Furthermore, flexible trusted boot requirements and OEM-dependent boot operations require that, during boot time, some services or operations need to be performed in a Trusted Execution Environment. Therefore, a minimal set of the TEE capabilities exists during the device boot time. To enable some of these services, a Trusted OS (or some simplified version thereof) can also exist.

A typical TEE secure boot is based on three key components:

- A fixed set of innately trusted components; typically the smallest distinguishable set of hardware and/or software that is inherently trusted and tied to the logic/environment where trusted actions are performed
- Immutable boot software stored, for example, in in-chip TEE ROM
- The isolated TEE where this security critical boot software is executed

It is not the current intention of GlobalPlatform to define the boot time capabilities of a TEE; however, if a TEE Trusted OS is required to function during boot, then it is recommended for compatibility and ease of development that it implements as much of a subset of the TEE Internal APIs as it is capable of providing.

5.2.1 Typical Boot Sequence

The figures that follow depict three simplified examples of secure boot of a TEE. Common to all solution examples, the device boots from the TEE boot ROM code inside the IC package containing the TEE (which might not be the IC package containing the REE). The TEE boot ROM can then load further firmware components and verify them before execution. To verify them, code in the boot ROM uses the information found in the fixed set of innately trusted hardware components (for example, information stored in the TEE boot ROM or one time programmable (OTP) fuses). The firmware components are typically stored in rewriteable non-volatile memories such as flash storage but can also be part of the TEE ROM code.

Before exiting the secure boot process, the firmware or the TEE platform code loads and can verify REE boot loader(s) before their execution. Typically, if any loaded software component verification fails up to this point, the boot process halts and the device reboots with a possible error report/indication. In a successful case, the REE boot loader starts the process of loading the Regular OS or further boot loader components.

OEMs can differentiate by implementing trusted firmware to be run early in the boot sequence. This gives the OEM the flexibility to bring in its own keys, certificate format, signature schemes, etc. Figure 5-1 through Figure 5-3 illustrate example boot sequences, and others can exist.

Figure 5-1: Example Boot Sequence: Trusted OS Early Boot

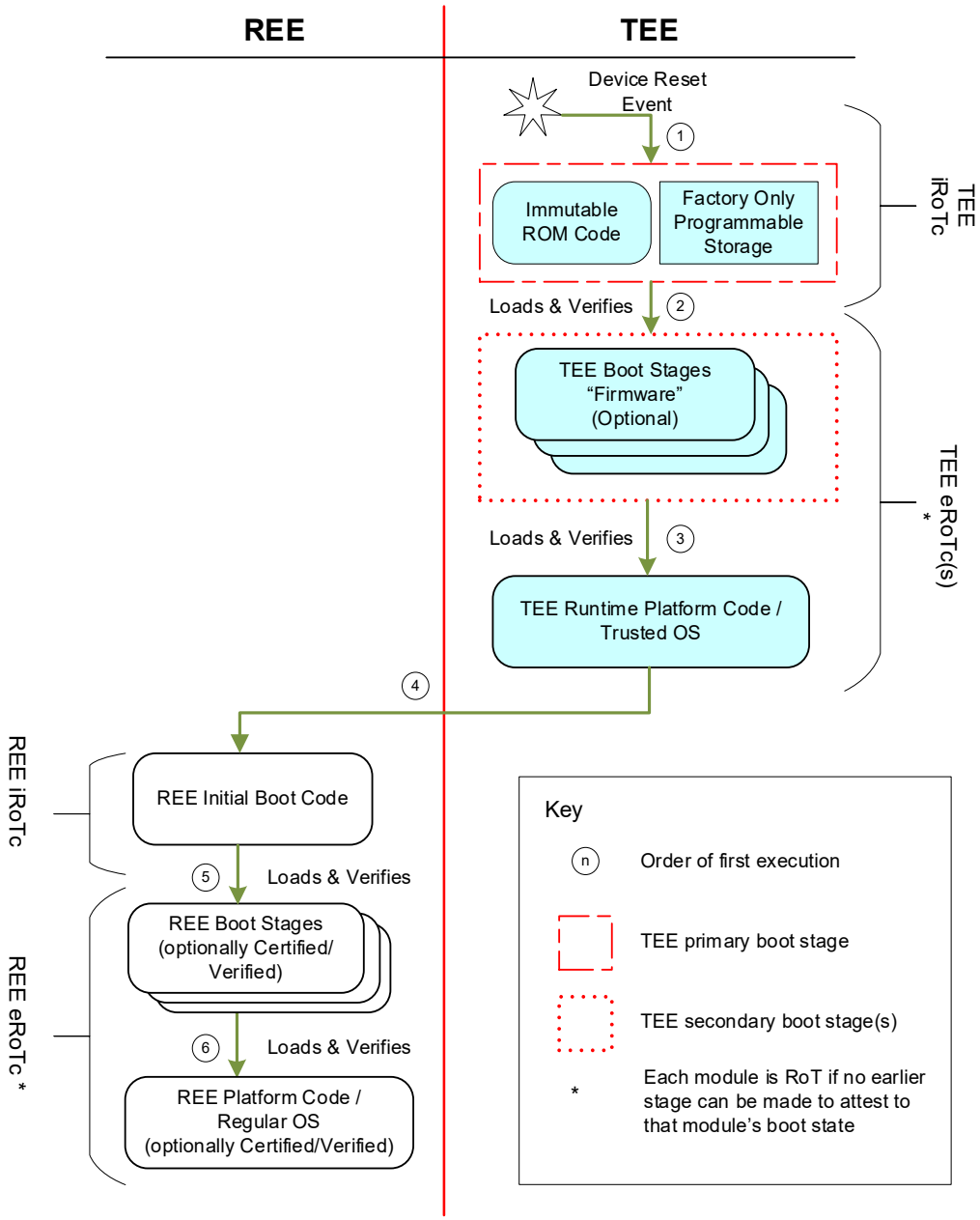


Figure 5-2: Example Boot Sequence: ROM-based Trusted OS

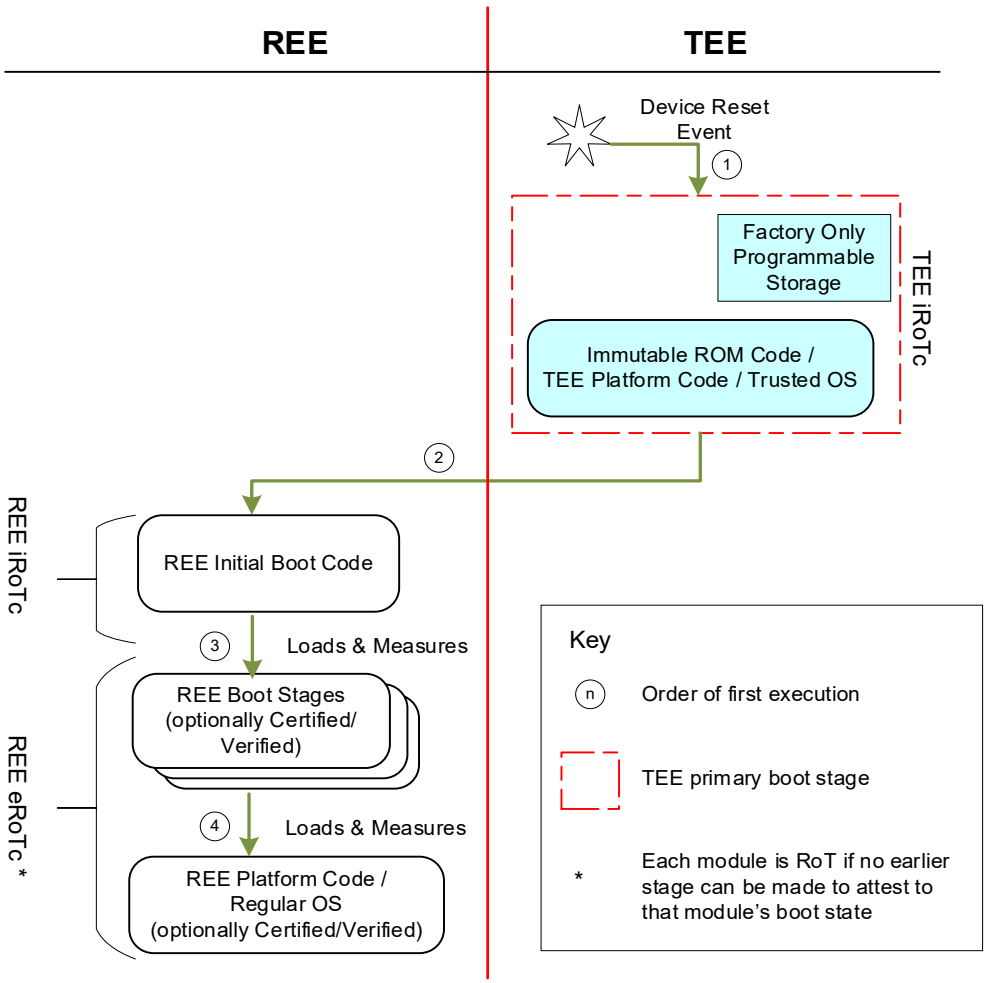
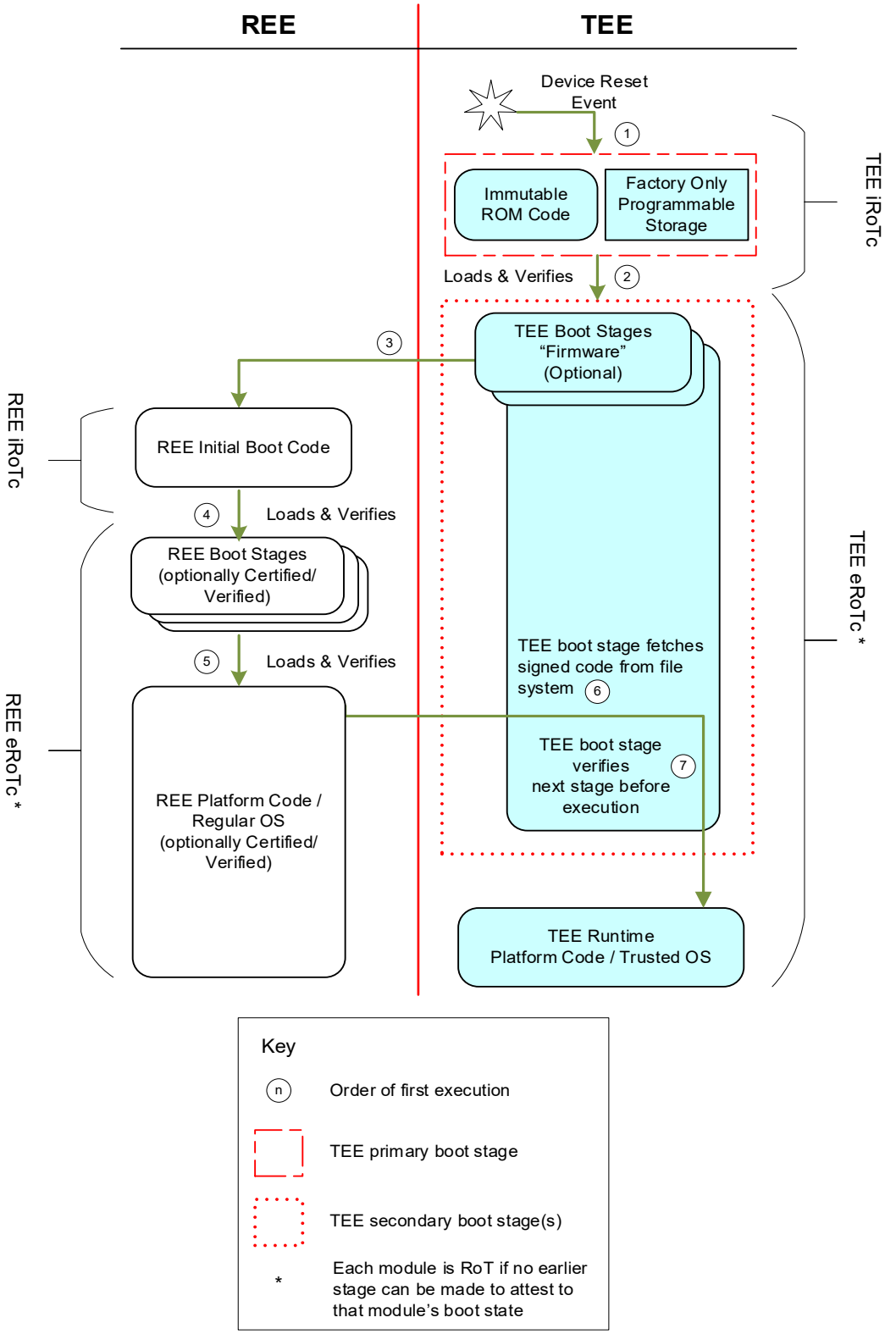


Figure 5-3: Example Boot Sequence: Trusted OS On-demand Boot



5.3 Run-time Environment

The term “run-time” refers to a property of the overall Execution Environment where an operating system has fully completed its initialization/boot operations and is fully operational, as opposed to the interval before the operating system is fully operational, as discussed in section 5.2.

The dependencies between the Trusted OS and the Regular OS are implementation dependent. Current GlobalPlatform specifications standardize the behavior of the system once the Regular OS is operational. This does not mean that there are no capabilities when the Regular OS is not operational (see section 5.2).

5.3.1 TEE Functionality Availability

The claimed TEE functionality and availability can have dependencies on the REE, which may limit the TEE’s abilities.

However, in a device that claims to have a TEE available to CAs, a CA in a fully booted Regular OS SHALL be able to access a TEE’s full claimed functionality via an appropriate TA. This includes the scenario where the full functionality of the TEE is only initialized in reaction to that CA’s request to access the TA.

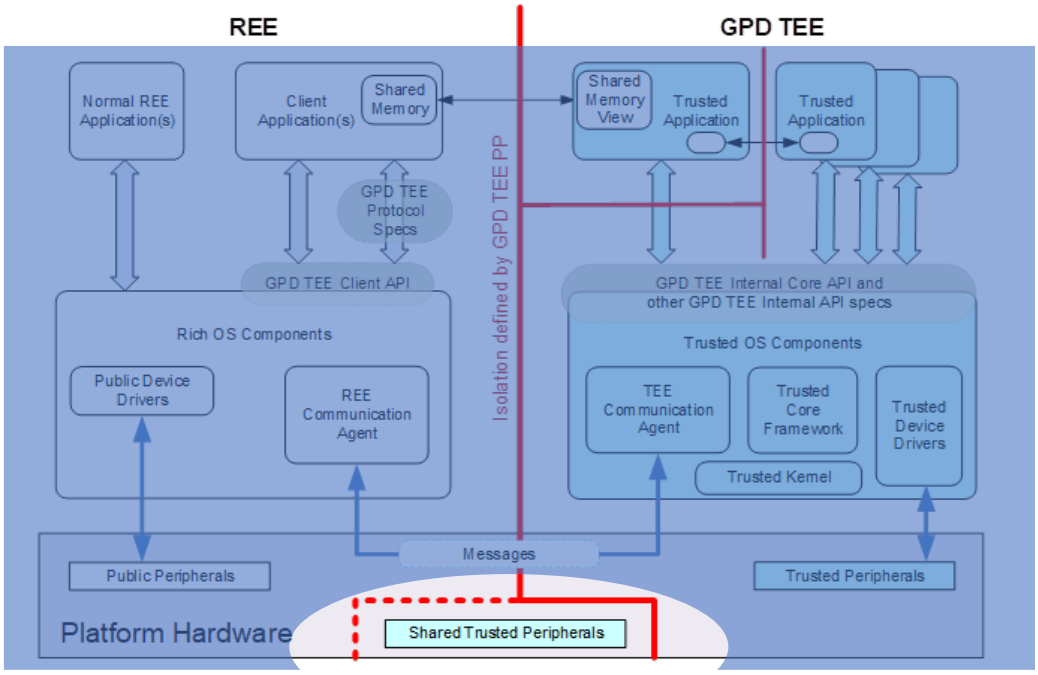
A TEE is permitted to provide a subset (or all) of its service during earlier stages in the boot processes.

The above guarantee of availability to Client Applications means that effects such as power state changes, where the Client Applications are not aware of such a change, are not noticeable via their connection to Trusted Applications unless a Trusted Application chooses to expose such information.

5.4 Transfer of Hardware Components to and from the TEE

One advantage of the TEE over other HSM-like security solutions is its ability to share trusted peripherals (such as displays and touch sensors) with other Execution Environments in a device.

Figure 5-4: Shared Trusted Peripherals



While enabling a range of unique and valuable use cases, this does bring risks. When a Shared Trusted Peripheral is transferred to a TEE's control from elsewhere, the transferred component must not have the potential to violate the Protection Profile of the TEE or the other environments in the device.

5.4.1 Accidental Exposure of TEE Assets by Transfer

In general, when a Shared Trusted Peripheral is transferred out of the TEE, the TEE must not accidentally transfer a controlled data asset out at the same time.

> For example, consider a touch panel controller (TPC). When transferred out of the TEE, the TPC registers should not contain any information on recent touch locations that might be used to infer a password entry or other similar action.

When a TEE is evaluated for compliance to the GlobalPlatform TEE Protection Profile, testing will have shown that such transfers do not enable such violation.

5.4.2 Moving a Shared Trusted Peripheral into TEE Use

In general, when a Shared Trusted Peripheral is transferred into the TEE, the TEE SHALL NOT accidentally create a security hole at the same time.

> For example, consider a simple Direct Memory Access (DMA) controller. When transferred into the TEE, the operating state of a DMA should not enable the DMA to be set up by something in the REE to instigate unauthorized data transfer from inside the TEE to outside of the TEE.

For Shared Trusted Peripherals that have some level of programmability:

- When a transferred Shared Trusted Peripheral with internal firmware is transferred to a TEE, one of the following SHALL be true:
  - That firmware SHALL NOT have been modified by systems outside of the TEE.
  - Or any modifications made outside of the TEE SHALL be evaluated by the TEE before the system is used.

    > This might be done by restarting the Shared Trusted Peripheral, so that its firmware goes through a trusted boot sequence, or by having a part of the TEE check some form of signature of the current firmware.
  - Or the TEE SHALL reload the image from a known good image.
- When a transferred Shared Trusted Peripheral with modifiable functionality (e.g. an FPGA or PLA) is transferred to a TEE, one of the following SHALL be true:
  - That functionality SHALL NOT have been modified by systems outside of the TEE.
  - Or any modifications made outside of the TEE SHALL be evaluated by the TEE for acceptability, before the system is used.

    > This might be done by having a part of the TEE check some form of signature of the transferred Shared Trusted Peripheral's image followed by a forced reload of that validated image into the transferred Shared Trusted Peripheral.
  - Or the TEE SHALL reload the image from a known good image followed by forcing the reload of that image into the Shared Trusted Peripheral.

When a TEE is evaluated for compliance to the GlobalPlatform TEE Protection Profile, testing will have shown that such transfers do not enable such violation.

5.4.3 Respect of REE Security and Transfer of Assets to the REE

While it is up to a REE to define its protection profile (or equivalent security statement), and it is up to a device implementer to bring together the various security needs of the systems in a device, the TEE implementor should enable the system to meet those restrictions where the TEE may impact them.

As such a TEE shall not act on (read/write/delete/block) an external EE (e.g. through shared memory) unless this is in line with the Access Control Rules and settings of that external EE.

It follows that any EE in a device might also wish to take TEE-like precautions (as mentioned in section 5.4.2) when receiving a Shared Trusted Peripheral from a TEE or elsewhere. Whether it needs to do so depends on its trust relationship with the sourcing EE.