GlobalPlatform Technology
TEE System Architecture
Version 1.1.0.10 (Target v1.2)

Public Review

September 2018

Document Reference: GPD_SPE_009

Contents

Figures

Tables

1 Introduction

Devices, from smartphones to servers, offer a Rich 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 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 Trusted Execution Environment (TEE) that meets the TEE Protection Profile ([TEE PP]) is an environment where the following are true:

- All code executing inside the TEE has been authenticated.
- Unless explicitly shared with entities outside the TEE:
  - The ongoing integrity of all TEE assets is assured through isolation, cryptography, or other mechanisms.
  - The ongoing confidentiality of the contents of all TEE data assets is assured through isolation or other mechanisms such as cryptography. Data assets include keys.
- TEE capabilities such as isolation or cryptography, can be used to provide confidentiality of the TA code asset.
- The TEE resists known remote and software attacks, and a set of external hardware attacks.
- Both code and other assets are protected from unauthorized tracing and control through 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.

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. Further extensions of the TEE System Architecture are expected in subsequent phases, as described in the TEE White Paper ([TEE White Paper]); e.g. a more flexible Trusted User Interface API, biometrics fingerprint API, and secure video content.

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.

1.1 Audience

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

- Trusted Execution Environments
- 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 ([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

Table 1-1: Normative References

Standard / Specification	Description	Ref
TEE White Paper	The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market, June 2015	[TEE White Paper]
GPD_SPE_010	GlobalPlatform Technology TEE Internal Core API Specification v1.1	[TEE Core API]
GPD_SPE_007	GlobalPlatform Technology TEE Client API Specification v1.0	[TEE Client API]
GPD_SPE_021	GlobalPlatform Technology TEE Protection Profile v1.2	[TEE PP]
GPD_SPE_025	GlobalPlatform Technology TEE TA Debug Specification v1.0	[TEE TA Debug]
GPD_SPE_024	GlobalPlatform Technology TEE Secure Element API Specification v1.1	[TEE SE API]
GPD_SPE_020	GlobalPlatform Technology TEE Trusted User Interface API Specification v1.0	[TEE TUI API]
GPD_SPE_055	GlobalPlatform Technology TEE Trusted User Interface Low-level API v1.0	[TEE TUI Low]
GPD_SPE_027	GlobalPlatform Technology TEE Management Framework Specification v1.0	[TEE Mgmt]
GPD_SPE_100	GlobalPlatform Technology TEE Sockets API Specification v1.0	[TEE Sockets]
GPD_GUI_069	GlobalPlatform Technology TEE Initial Configuration v1.1	[TEE Init Config]
GPD_SPE_075	GlobalPlatform Technology Open Mobile API Specification	[Open Mobile]
GP_REQ_025	GlobalPlatform Technology Root of Trust Definitions and Requirements v1.0.1	[RoT Req]

Standard / Specification	Description	Ref
OMTP ATE TR1	Open Mobile Terminal Platform (OMTP) Advanced Trusted Environment TR1 v1.1	[OMTP ATE TR1]
RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]
TCG	Trusted Computing Group Glossary, https://trustedcomputinggroup.org/wp-content/uploads/TCG-Glossary-V1.1-Rev-1.0.pdf, visited 2 August 2018.	[TCG]

Table 1-2: Informative References

Standard / Specification	Description	Ref
GPD_SPE_042	GlobalPlatform Technology TEE TUI Extension: Biometrics API v1.0	[TEE TUI Bio]
BSI-CC-PP-0084	Common Criteria Protection Profile Security IC Platform Protection Profile with Augmentation Packages	[PP-0084]

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]):

- **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.

Table 1-3: Terminology and Definitions

Term	Definition
Application Programming Interface (API)	A set of rules that software programs can follow to communicate with each other.
Biometrics API	This extension of the TEE Trusted User Interface Low-level API supports the discovery and identification of all biometric capabilities and the use of biometric functionality supported by hardware, entirely protected inside the TEE.
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*.
Core Migration	The transfer of the task of execution of code from one CPU core to another.
Enhanced Root of Trust (eRoT)	A Root of Trust whose integrity is verified by another Root of Trust at any point during the life cycle of the platform, with no inference of that verification measurement available from an ancestor entity.
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
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.

Term	Definition
Initial Root of Trust (iRoT)	A Root of Trust that a platform manufacturer provisions and initializes during the manufacturing process and that is the first 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 hence to place more facilities inside that hardware boundary. These extra facilities would not be considered "on-SoC" but are considered "in-package".
One Time Programmable (OTP)	A form of memory that can be read many times, but only written once. On a typical SoC implementing a TEE, this can be a very limited resource in the order of a few thousand bits at most. An example of this form of memory is e-fuse.
Platform	An execution environment inside a device. SE, TEE, and REE are examples of platforms.
REE Communication Agent	A Rich OS driver that enables communication between REE and TEE. Contrast *TEE Communication Agent*.
Rich Execution Environment (REE)	An execution environment comprising at least one Rich OS and all other components of the device (SoCs, other discrete components, firmware, and software) which execute, host, and support the Rich OS (excluding any TEEs and SEs included in the device). WARNING: In a previous version of this document the REE was considered to be everything outside of the TEE under consideration. In the new definition other entities are acknowledged. Contrast *Trusted Execution Environment*.
Rich OS	Typically an OS providing a much wider variety of features than that of the OS running inside the TEE. It is very open in its ability to accept applications. It will have been developed with functionality and performance as key goals, rather than security. Due to the size and needs of the Rich OS it will run in an execution environment outside of the TEE hardware (often called an REE – Rich Execution Environment) with much lower physical security boundaries. From the TEE viewpoint, everything in the REE has to be considered untrusted, though from the Rich OS point of view there can be internal trust structures. Contrast *Trusted OS*.
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.

Term	Definition
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	The owner or vendor of a combination of CA and/or TA software.
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 an REE with Trusted Applications running inside a TEE.
TEE Communication Agent	Trusted OS driver that enables communication between REE and TEE. Contrast *REE Communication Agent*.
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 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 Service Library	A software library that includes all security related drivers.
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 API to support TA development and/or compliance testing of the TEE Internal APIs.

Term	Definition
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.
Trusted Execution Environment (TEE)	An execution environment that runs alongside but isolated from an REE. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats. 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 *Rich Execution Environment*.
Trusted OS	The operating system running in the TEE. It 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. A TEE can host one and only one Trusted OS. Contrast *Rich 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 1-4: Abbreviations and Notations

Abbreviation / Notation	Meaning
API	Application Programming Interface
BIOS	Basic Input/Output System
CA	Client Application

Abbreviation / Notation	Meaning
DLM	Debug Log Message
DRAM	Dynamic Random Access Memory
DRM	Digital Rights Management
EE	Execution Environment
eRoT	Enhanced Root of Trust
GPD TEE	*See definition in Table 1-3.*
I/O	Input/Output
IC	Integrated Circuit
IoT	Internet of Things
IP	Internet Protocol
IPR	Intellectual Property Rights
iRoT	Initial Root of Trust
OEM	Original Equipment Manufacturer
OMTP	Open Mobile Terminal Platform
OS	Operating System
OTP	One Time Programmable
PCB	Printed Circuit Board
PMR	Post Mortem Reporting
RAM	Random Access Memory
REE	Rich Execution Environment
ROM	Read Only Memory
RoT	Root of Trust
rSD	root Security Domain
SD	Security Domain
SE	Secure Element
SoC	System-on-Chip
TA	Trusted Application
TCG	Trusted Computing Group
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TLS	Transport Security Layer
TPM	Trusted Platform Module
TUI	Trusted User Interface
UDP	User Datagram Protocol

Abbreviation / Notation	Meaning
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 1-5: Revision History

Date	Version	Description
December 2011	1.0	Initial Public Release
January 2017	1.1	Public Release
April 2018	1.1.0.4	Committee Review
June 2018	1.1.0.7	Member Review
September 2018	1.1.0.10	Public Review
TBD	1.2	Discuss 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 to be published in TEE Internal Core API) Discuss GlobalPlatform Root of Trust Definitions and Requirements in the context of TEE processing. Expand high-level security requirements discussion to include the required security assurance level and the activities of the GlobalPlatform TEE Security Evaluation Secretariat. Clarify minimum memory requirements of GlobalPlatform compliant TEEs.

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 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 separation from other environments in the device (including other TEEs). Anything that is not so separated SHALL be considered part of the TEE.

- 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.

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.

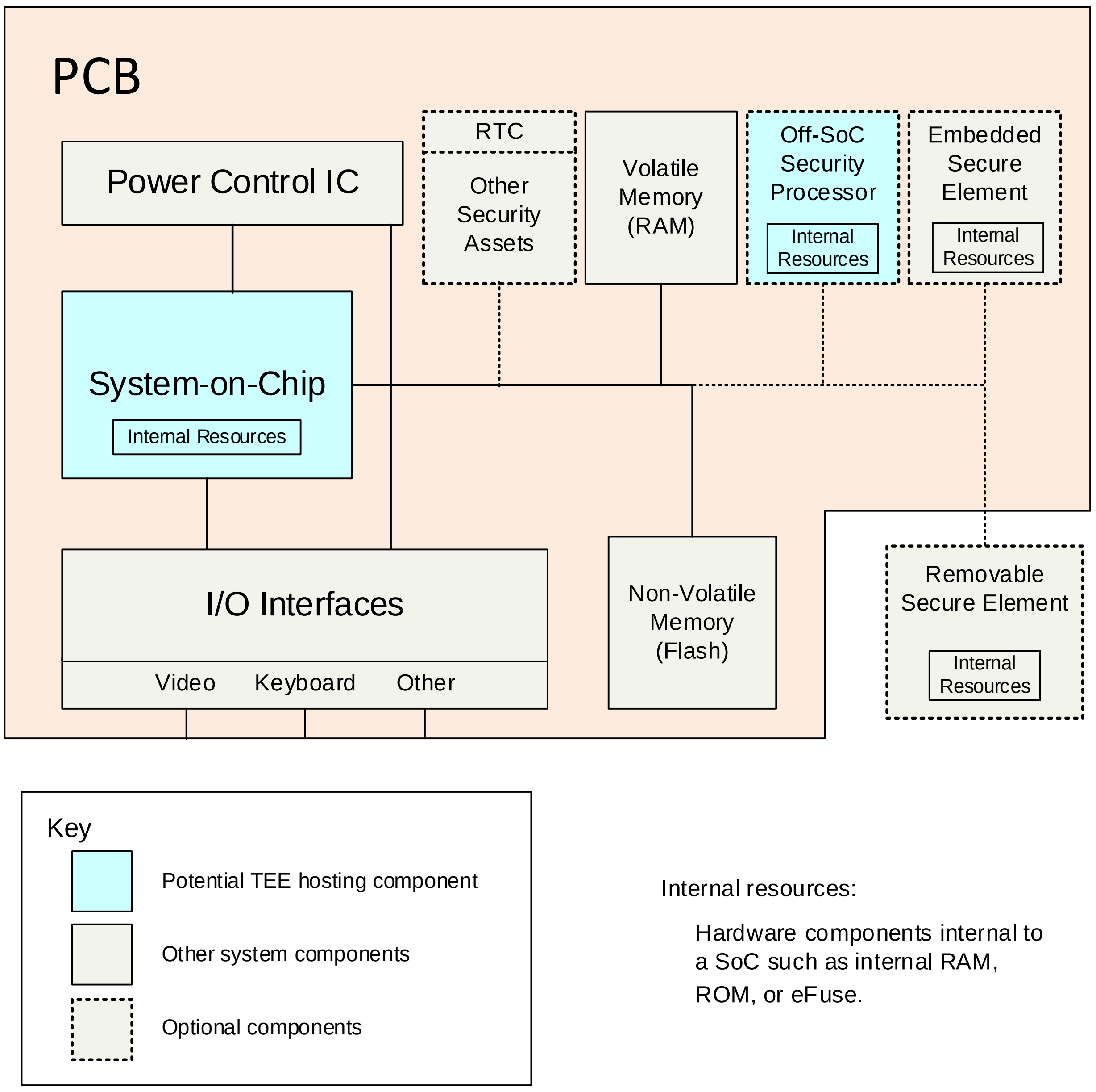
The remainder of this chapter describes the general device architecture associated with the TEE and provides a high level overview of the security requirements of a TEE.

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

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

Both the REE and the TEE utilize a number of resources such as processing core(s), RAM, ROM, cryptographic accelerators, etc. Figure 2-1 provides a simplified example of the resources that can exist at a device level. Figure 2-2 on page 19 provides an example of the resources that can be associated with a TEE hosting package such as the System-on-Chip (SoC) in Figure 2-1.

At any given time, resources are 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 the REE and other TEEs 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 trusted resources. These trusted resources are accessible only by other trusted resources and thereby make up a closed system that is protected from the REE and other TEEs.

Some resources accessible by the REE can be designed to also be accessible by the TEE without specific permission, whereas the opposite SHALL NOT hold. The REE SHALL only access TEE resources with specific permission.

In general terms, the TEE offers an execution space that provides a higher level of security than a Rich 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 the REE and other environments.
    - This is achieved through hardware mechanisms that those other environments cannot control.
- This protection always includes protection against other execution environments.
- 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 (such as those enabling debug of the REE) that cannot access unshared assets of the TEE.
- The Trusted OS run time environment is instantiated from a RoT inside the TEE through a secure boot process using assets bound to the TEE and isolated from the REE.
    - 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.
    - The Trusted Storage is bound to a 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 protection is at least equal to that of the TEE environment.
    - The Trusted Storage provides a minimum level of protection against rollback attacks.

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

      It is accepted that the actual physical storage can be in the REE and so is vulnerable to actions from outside of the TEE.
- 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 the REE software has requested.

The GlobalPlatform TEE Protection Profile ([TEE PP]) specifies the typical threats 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. 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. firmware TPM as defined in [TCG] 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 uses three classes of resources.

In-package resource

These resources are implemented in-package and so are protected 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 include trusted replay-protected external non-volatile memory areas, and trusted volatile memory areas. 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 SoC, 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 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 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.

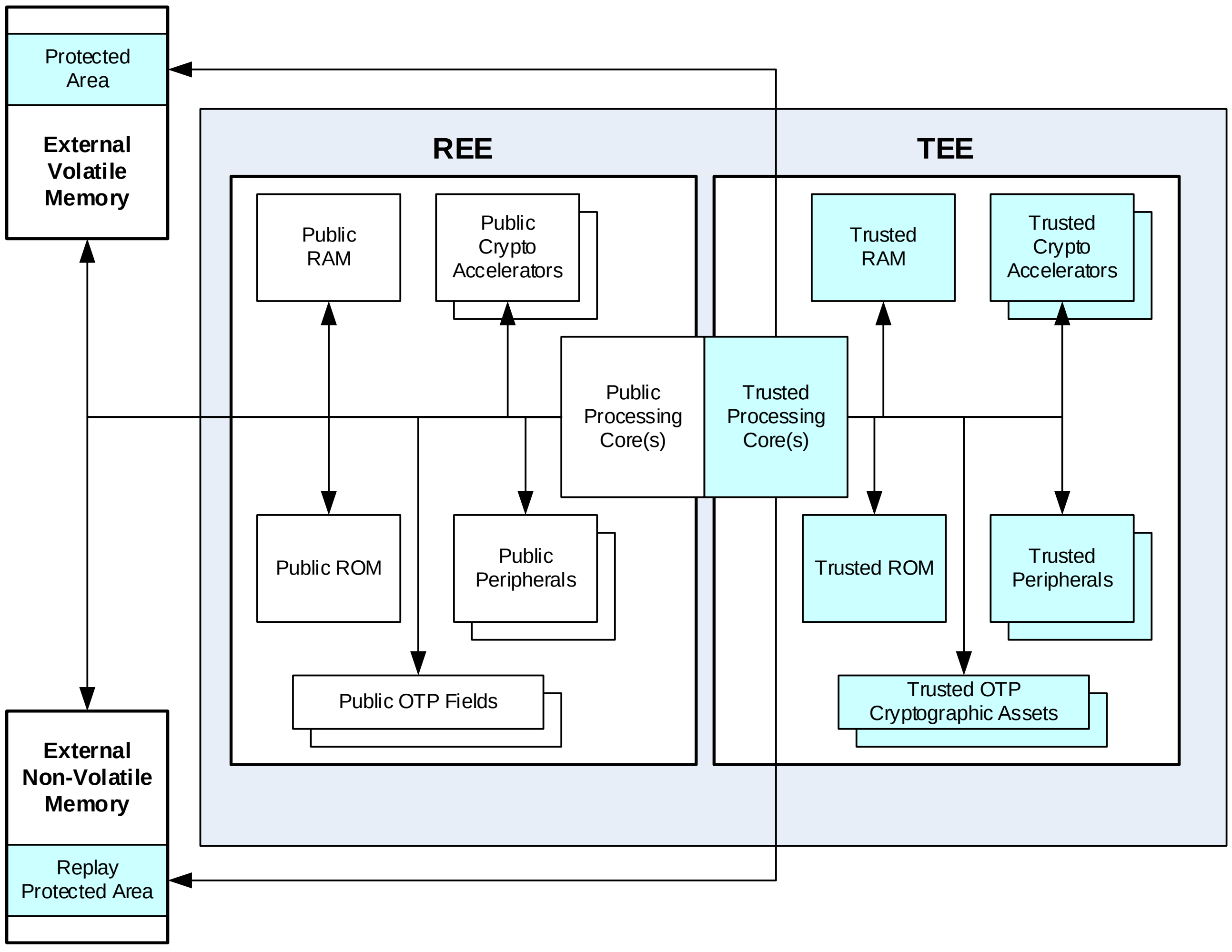
  Neither of the above use cases necessarily requires encrypted RAM storage, just isolation from the REE and other environments.
- Use keyboards and other I/O that are not accessible to the REE but are not guarded from physical attack.

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.

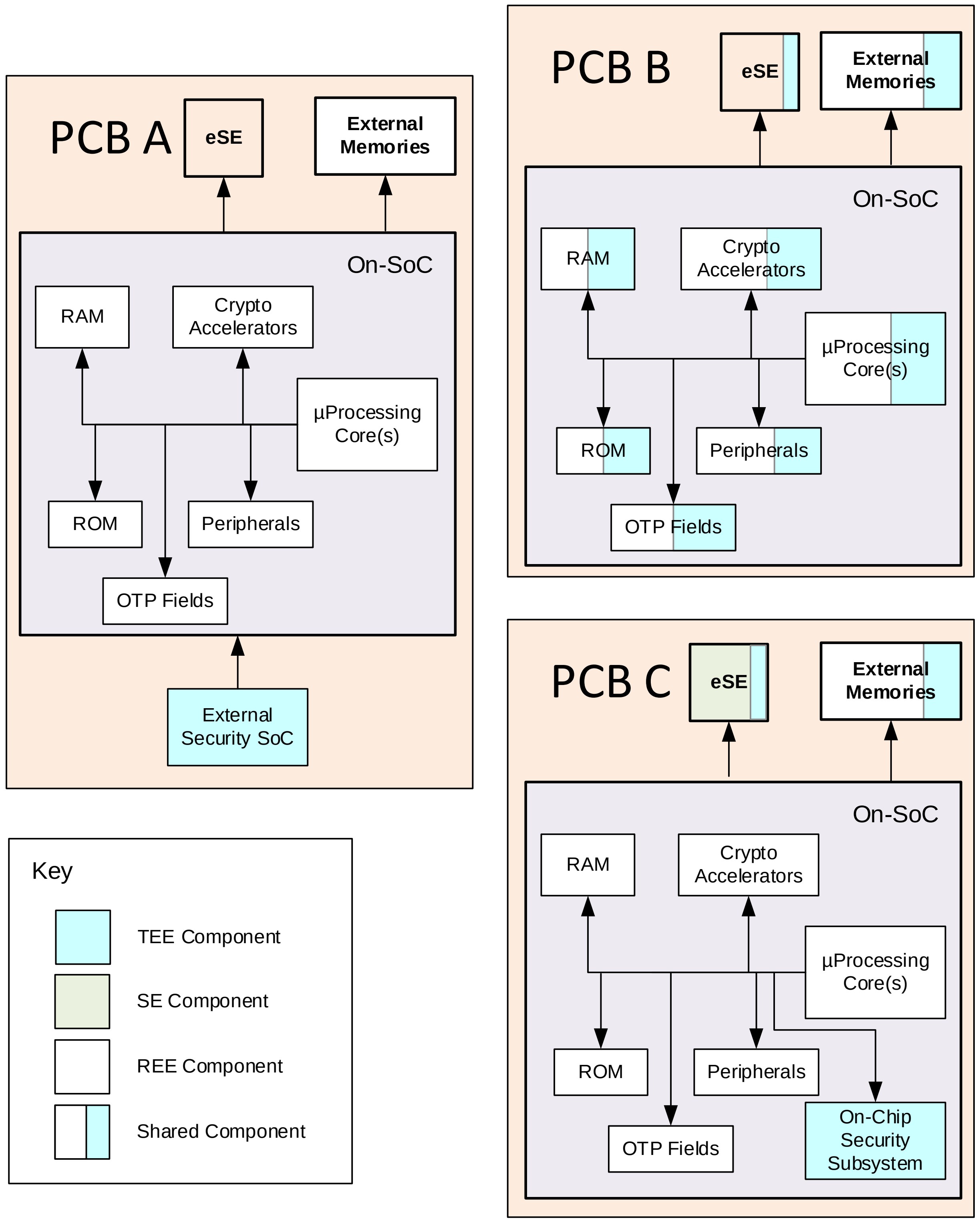
The REE has access to the untrusted resources, which can be implemented on-chip or off-chip in other components on the PCB. The REE cannot access the trusted resources. This access control is enforced through physical isolation, hardware logic based isolation, or cryptographic isolation methods. The only way for the REE to get access to trusted resources is via any 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 Architectural View of REE and TEE


Protected Area
External Volatile Memory
REE
TEE
Public RAM
Public Crypto Accelerators
Trusted RAM
Trusted Crypto Accelerators
Public Processing Core(s)
Trusted Processing Core(s)
Public ROM
Public Peripherals
Trusted ROM
Trusted Peripherals
Public OTP Fields
Trusted OTP Cryptographic Assets
External Non-Volatile Memory
Replay Protected Area

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



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.

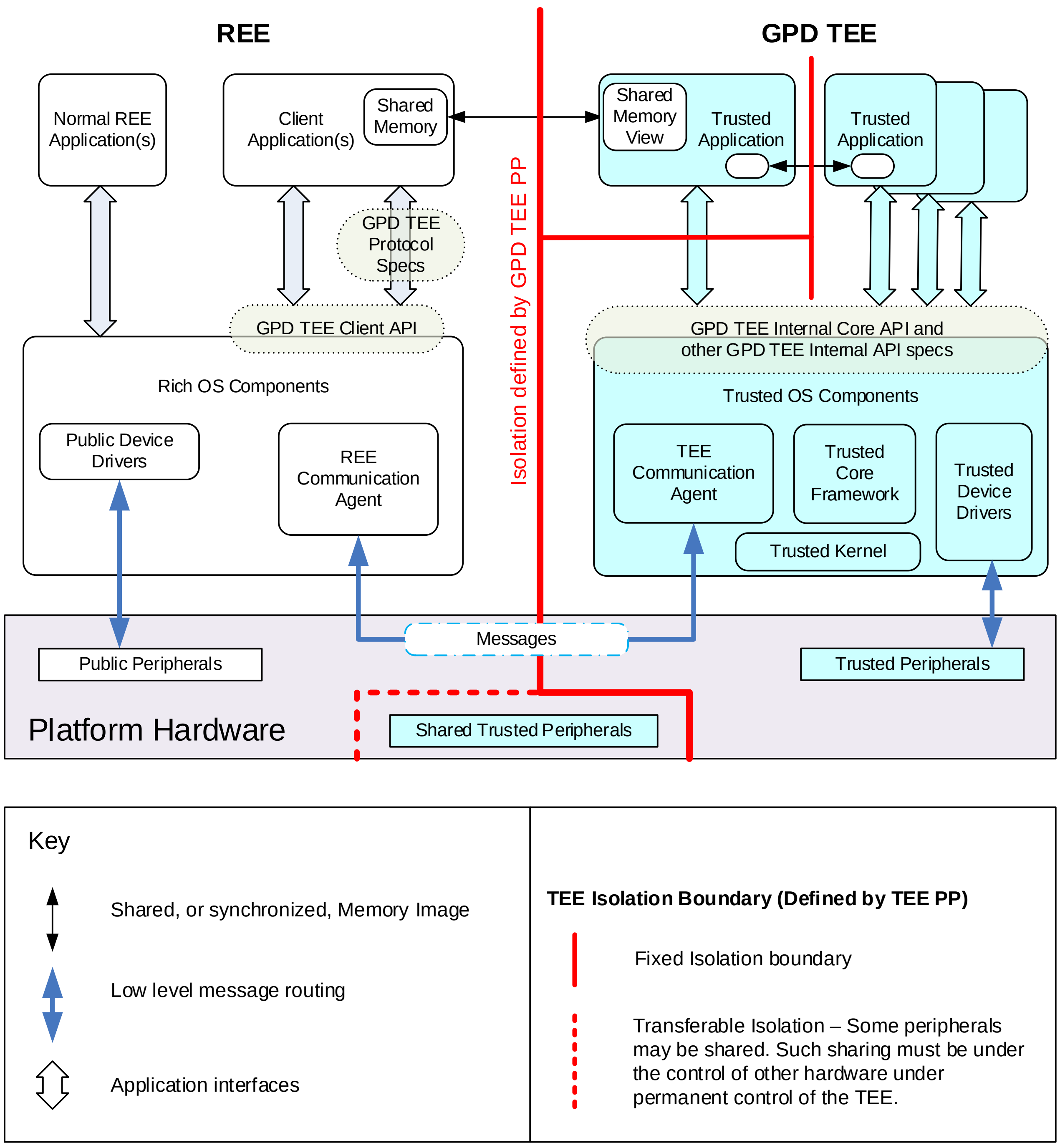
This chapter describes the general software architecture associated with the TEE, the interfaces defined by GlobalPlatform, and the relationship between the critical components found in the software system.

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

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 (TA) to provide isolated and trustworthy capabilities, which can then be used through intermediary Client Applications (CA).

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 Rich 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]) and TEE Management Framework Specification ([TEE Mgmt]) 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 the Trusted Applications, which run on top of that code.

Trusted OS Components consist of:

- The Trusted Core Framework which provides OS functionality to Trusted Applications.
  - The Trusted Core Framework is part of the TEE Internal Core API, discussed in section 3.5.1.
- Trusted Device Drivers which provide a communications interface to trusted peripherals that are dedicated to the TEE.

Both the Trusted Applications and Trusted Core Framework make use of scheduling and other OS management functions provided by the Trusted Kernel. The 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.

3.2.3 Trusted Applications (TAs)

The 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. 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 access to a memory area accessible to both the TEE and the REE. 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 a potential for a Client Application or Trusted Application to modify the memory contents asynchronously with the other parties acting on that memory.

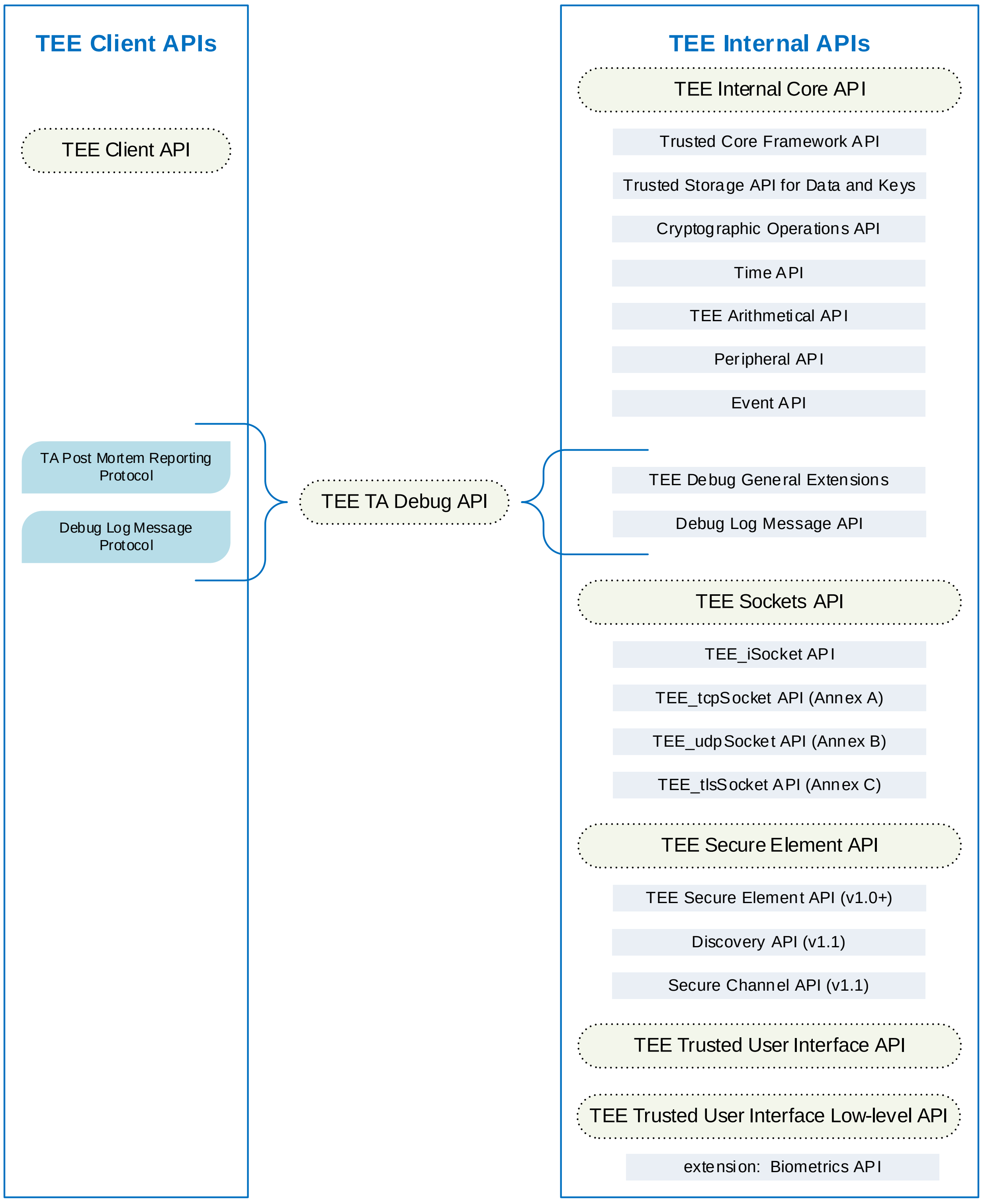
3.2.5 TA to TA Communication

A TA can communicate to another TA. This uses the same process used by the CA to communicate to the TA, but a trustworthy indicator allows the receiving TA to be assured that communication has not been exposed outside the TEE. This simplifies the matter of determining whether to trust the communication content and also the metadata associated with the content, such as the identity of the calling TA. Because of this trust relationship, a TA in another TEE in the same device is treated as though it is an REE-based CA, because the receiving TA's TEE has no reason to trust the calling TA's TEE.

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


TEE Client APIs
TEE Client API
TA Post Mortem Reporting Protocol
Debug Log Message Protocol
TEE TA Debug API
TEE Internal APIs
TEE Internal Core API
Trusted Core Framework API
Trusted Storage API for Data and Keys
Cryptographic Operations API
Time API
TEE Arithmetical API
Peripheral API
Event API
TEE Debug General Extensions
Debug Log Message API
TEE Sockets API
TEE_iSocket API
TEE_tcpSocket API (Annex A)
TEE_udpSocket API (Annex B)
TEE_tlsSocket API (Annex C)
TEE Secure Element API
TEE Secure Element API (v1.0+)
Discovery API (v1.1)
Secure Channel API (v1.1)
TEE Trusted User Interface API
TEE Trusted User Interface Low-level API
extension: Biometrics API

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 the TEE, establish a session with a trusted application, set up shared memory, send trusted application specific commands to invoke a trusted service, and 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 TEE Internal APIs such as the TEE Secure Element API, TEE Sockets API, and TEE TA Debug API.

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:

- The Trusted Applications, which make use of TEE Internal APIs
- The TEE Internal API library implementations
- The 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 a number of different 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].

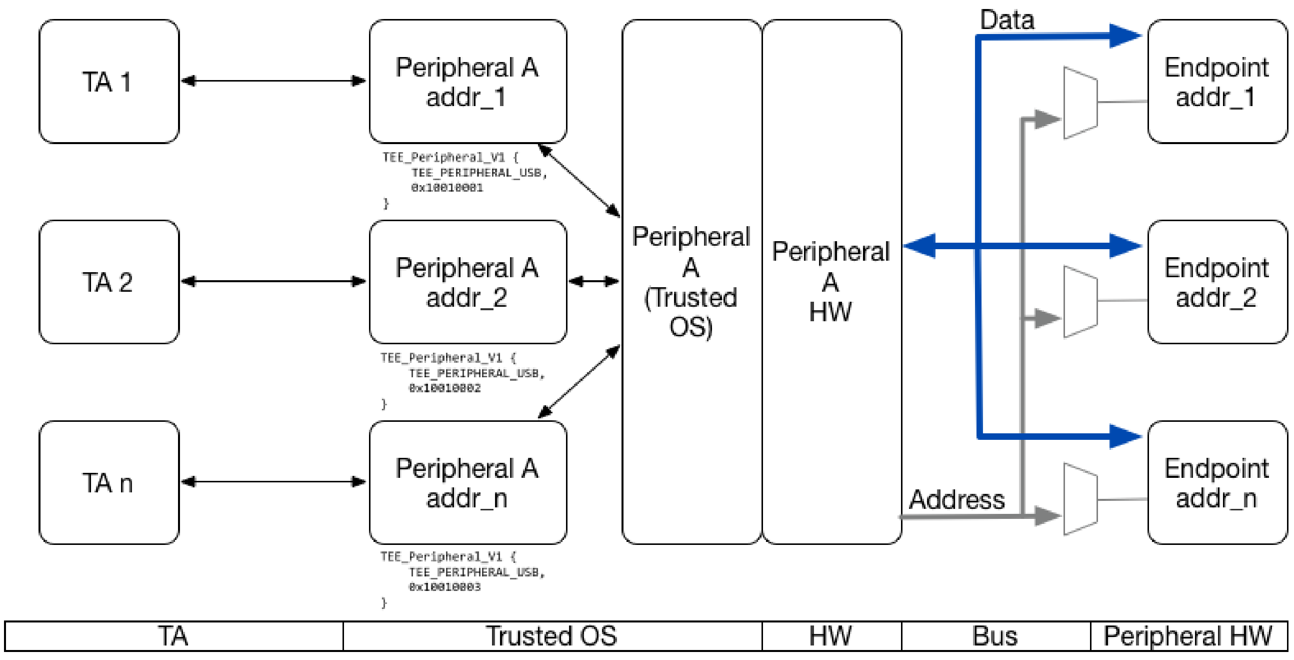
3.5.1.1 Peripheral and Event Access

With the addition of the Peripheral and Event APIs in [TEE Core API] v1.2, the TEE Internal Core API supports 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^{2}C interface can support separate endpoints for each I^{2}C address, while itself being the exclusive owner of the I^{2}C peripheral. Such drivers SHOULD ensure that information leakage between clients of the different endpoints is prevented.

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



```
TEE_Peripheral_V1 {
    TEE_PERIPHERAL_USB,
    0x10010001
}
```

```
TEE_Peripheral_V1 {
    TEE_PERIPHERAL_USB,
    0x10010002
}
```

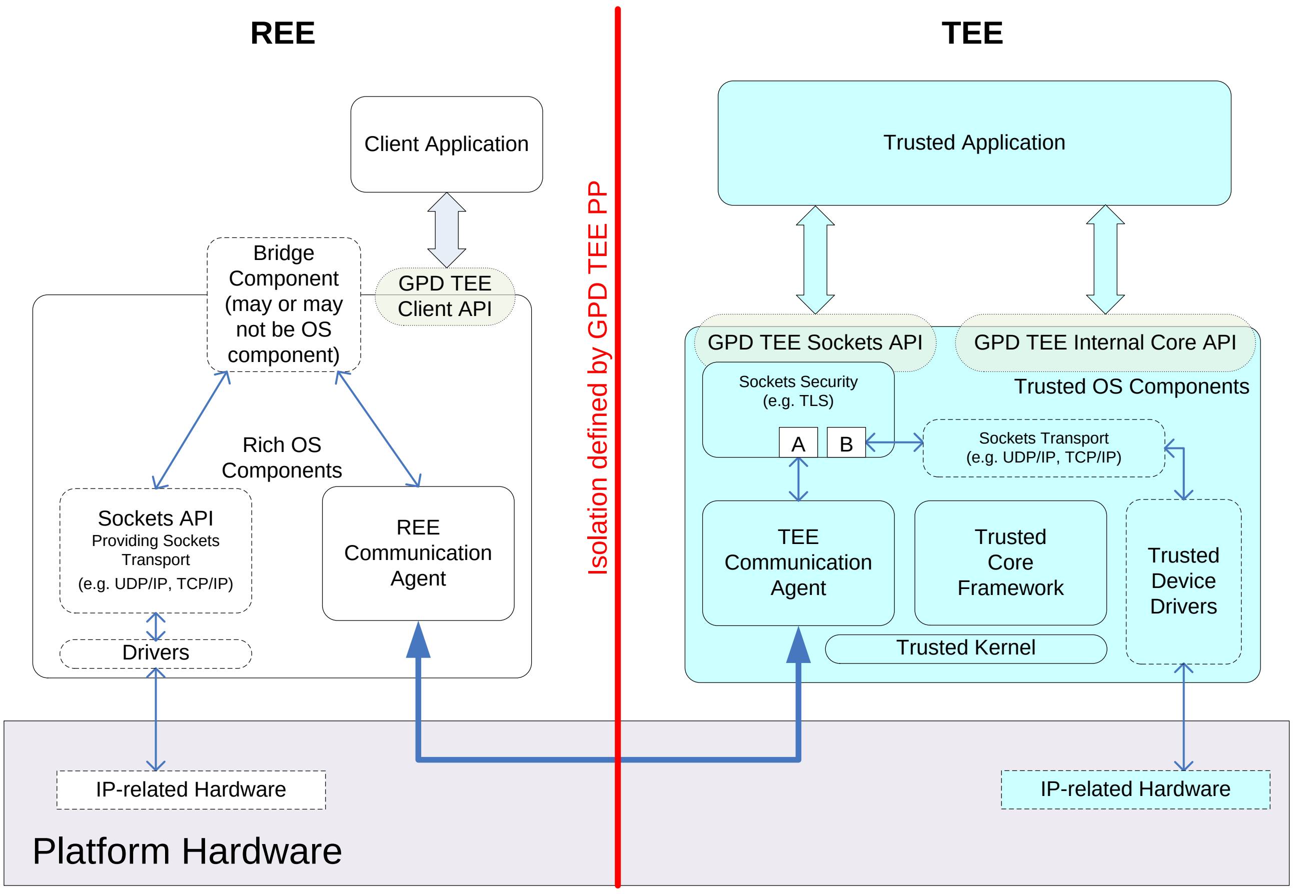
```
TEE_Peripheral_V1 {
    TEE_PERIPHERAL_USB,
    0x10010003
}
```

3.5.2 The TEE Sockets API

The GlobalPlatform TEE Sockets API ([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]).

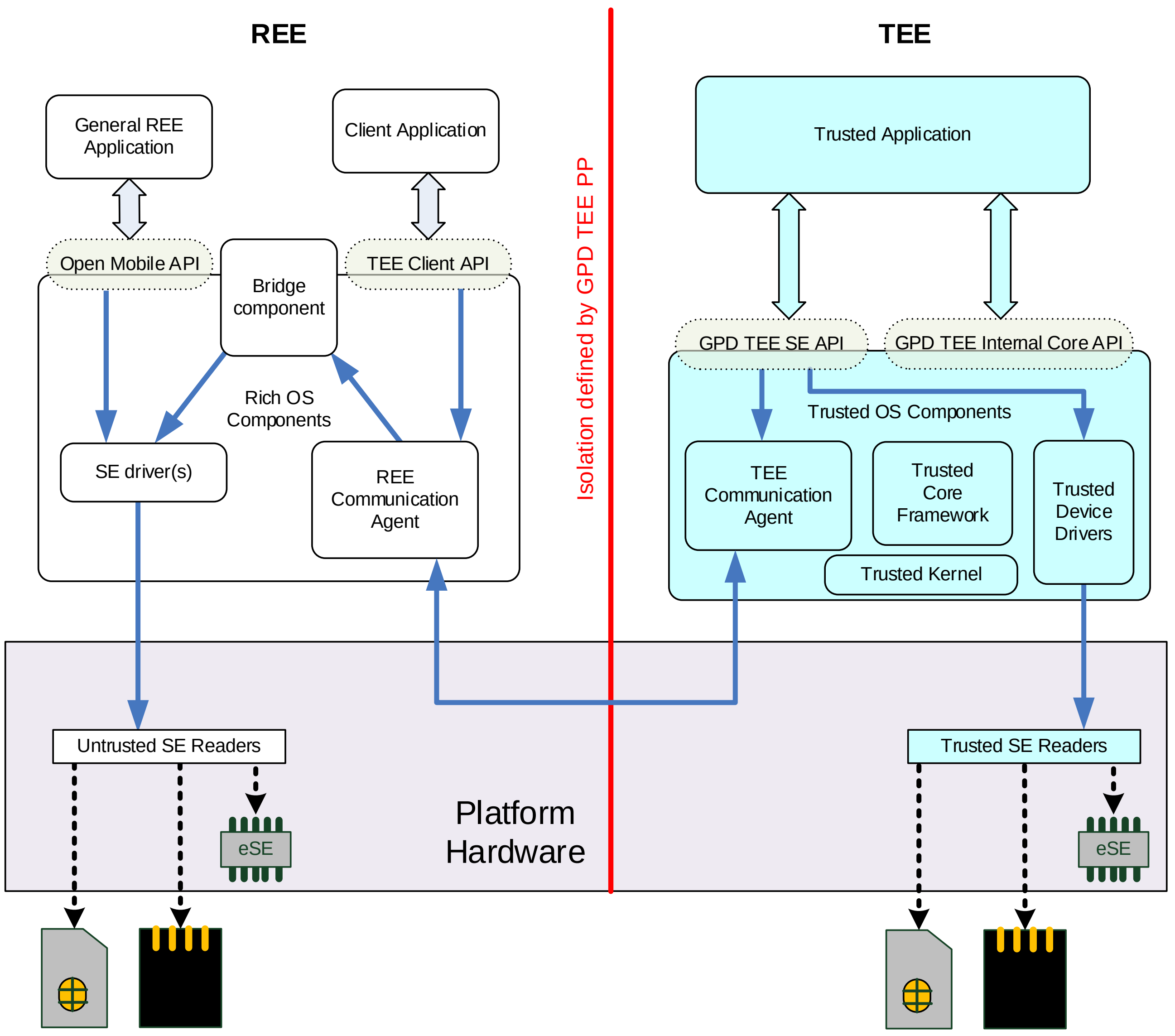
3.5.4 The TEE Secure Element API

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 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 lying in 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



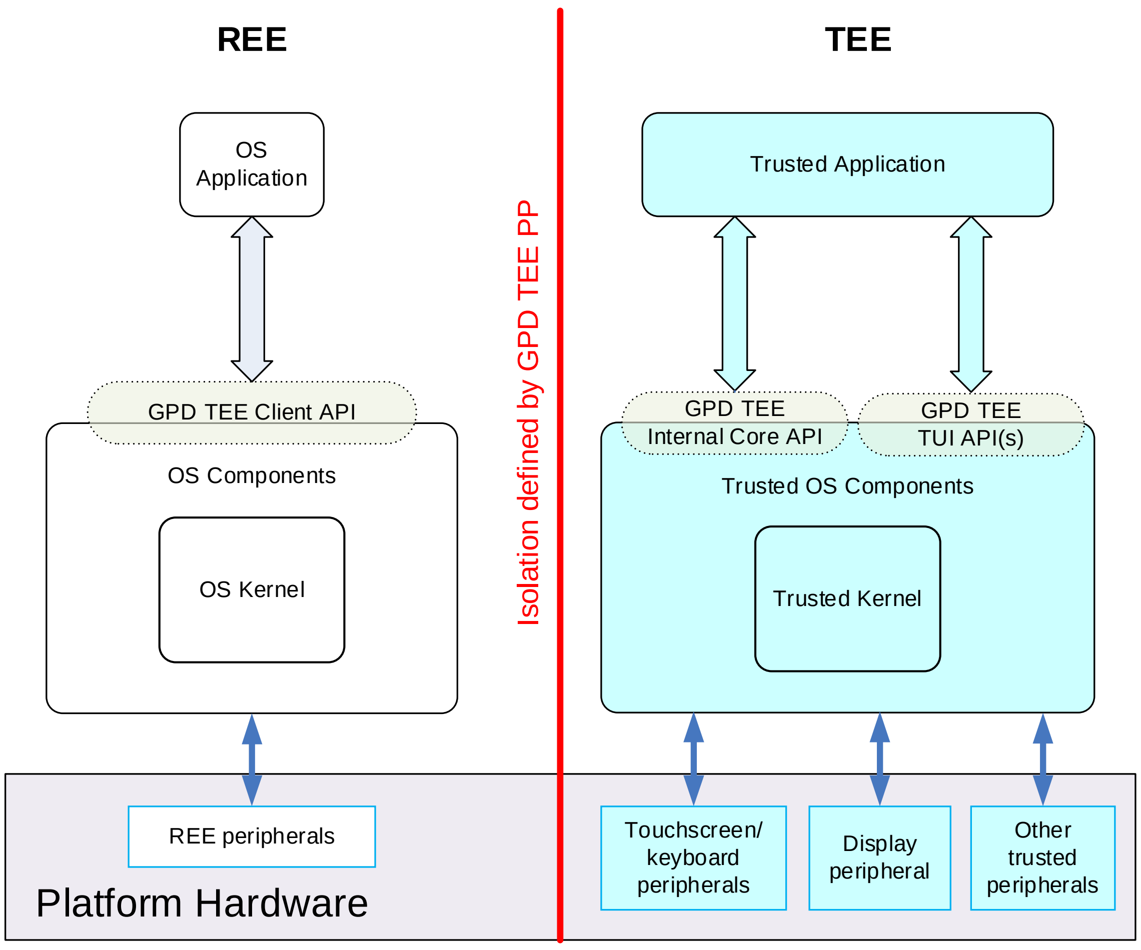
More information about the TEE Secure Element API can be found in the GlobalPlatform TEE Secure Element API Specification ([TEE SE API]).

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 only known 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. 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.

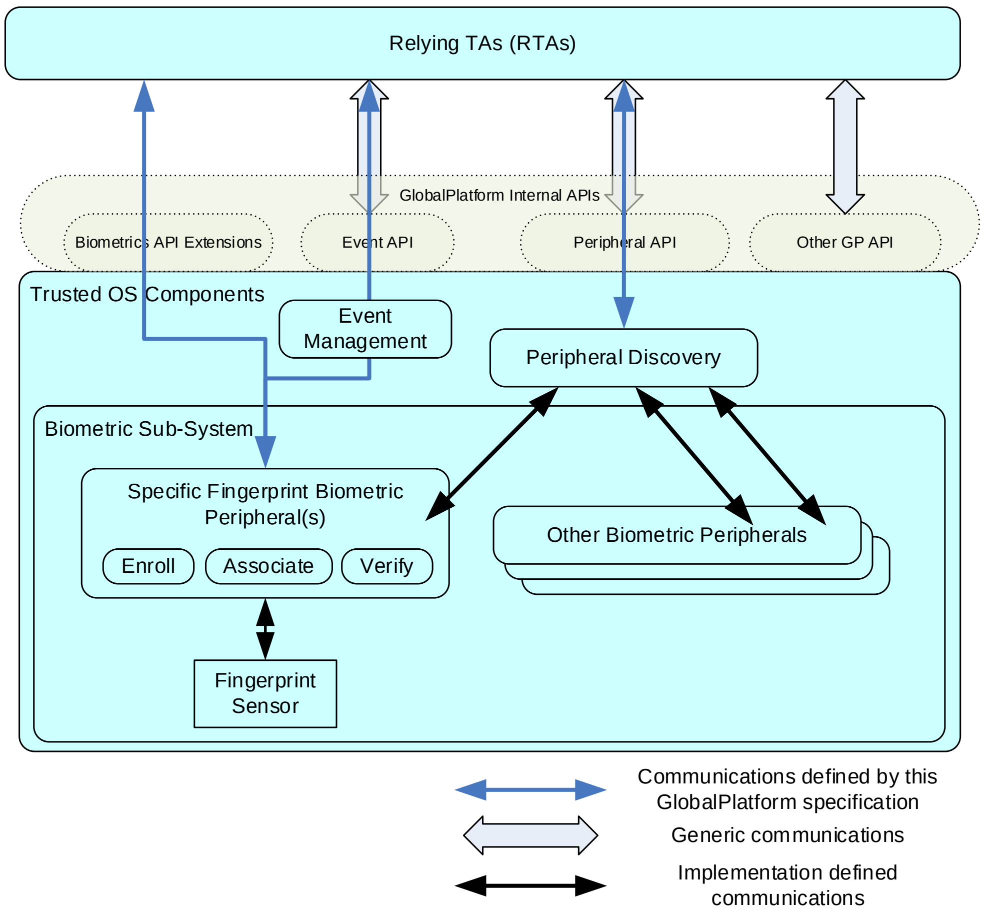
In general, it is an implementation choice as to whether particular functionality is implemented in the generic Biometric Sub-system, the Biometric Peripheral, or the Biometric Sensor. Such decisions SHALL be transparent to the calling TA.

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.

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

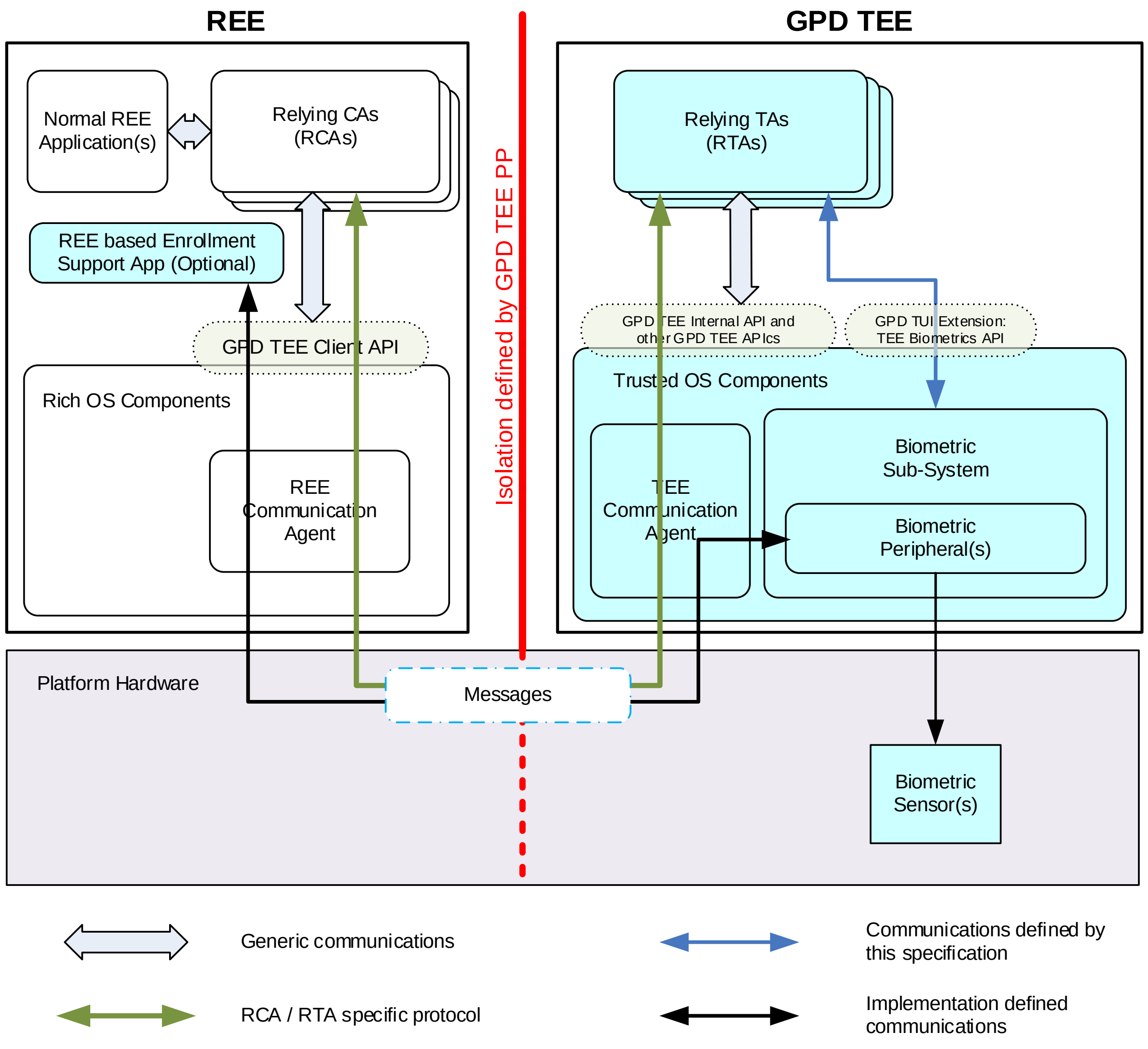

Relying TAs (RTAs)
GlobalPlatform Internal APIs
Biometrics API Extensions
Event API
Peripheral API
Other GP API
Trusted OS Components
Event Management
Peripheral Discovery
Biometric Sub-System
Specific Fingerprint Biometric Peripheral(s)
Enroll
Associate
Verify
Other Biometric Peripherals
Fingerprint Sensor
Communications defined by this GlobalPlatform specification
Generic communications
Implementation defined communications

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


REE
GPD TEE
Normal REE Application(s)
Relying CAs (RCAs)
REE based Enrollment Support App (Optional)
GPD TEE Client API
Rich OS Components
REE Communication Agent
Isolation defined by GPD TEE PP
Relying TAs (RTAs)
GPD TEE Internal API and other GPD TEE APIcs
GPD TUI Extension: TEE Biometrics API
Trusted OS Components
TEE Communication Agent
Biometric Sub-System
Biometric Peripheral(s)
Platform Hardware
Messages
Biometric Sensor(s)
Generic communications
Communications defined by this specification
RCA / RTA specific protocol
Implementation defined communications

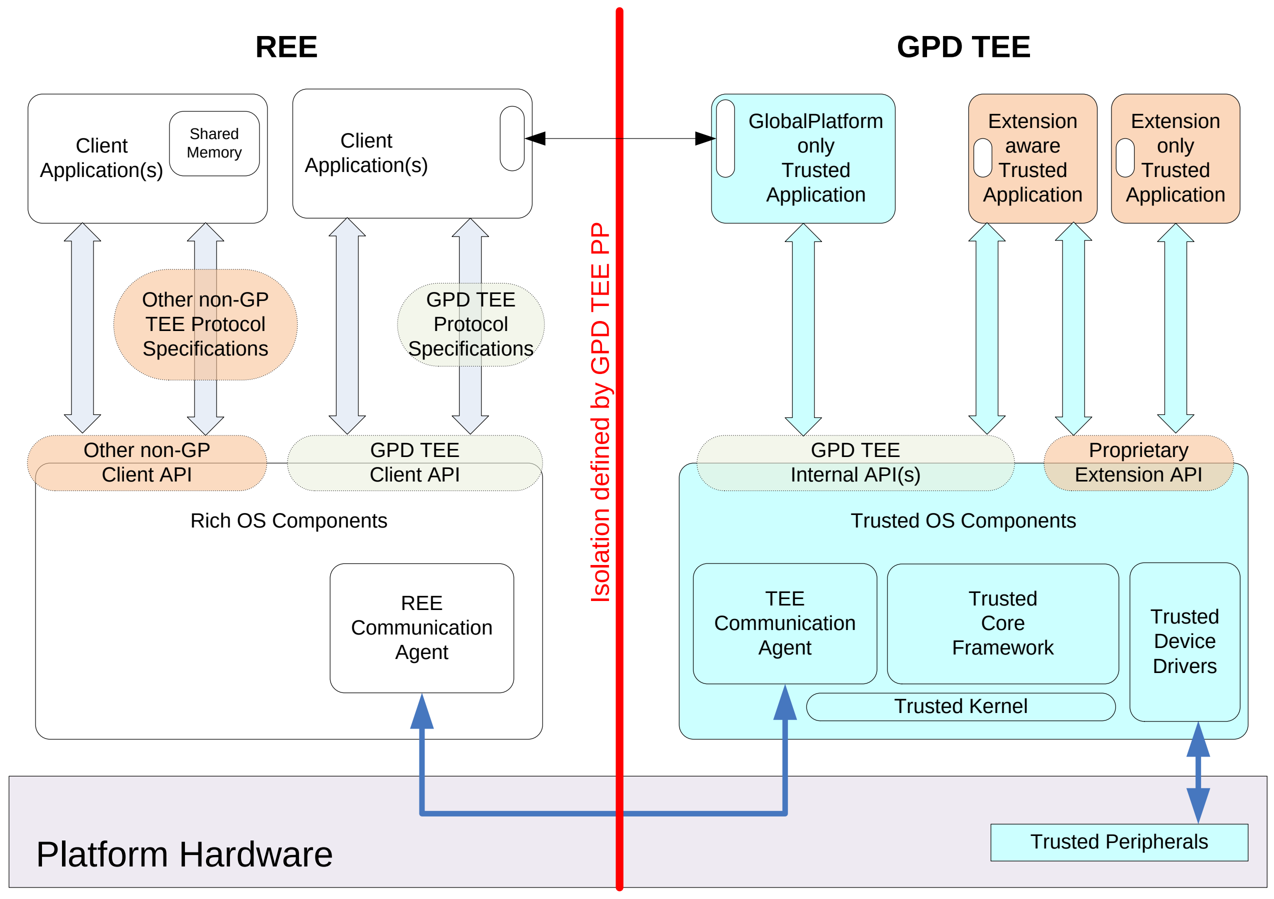
3.6 Variations of TEE Architecture Found on Real Devices

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:

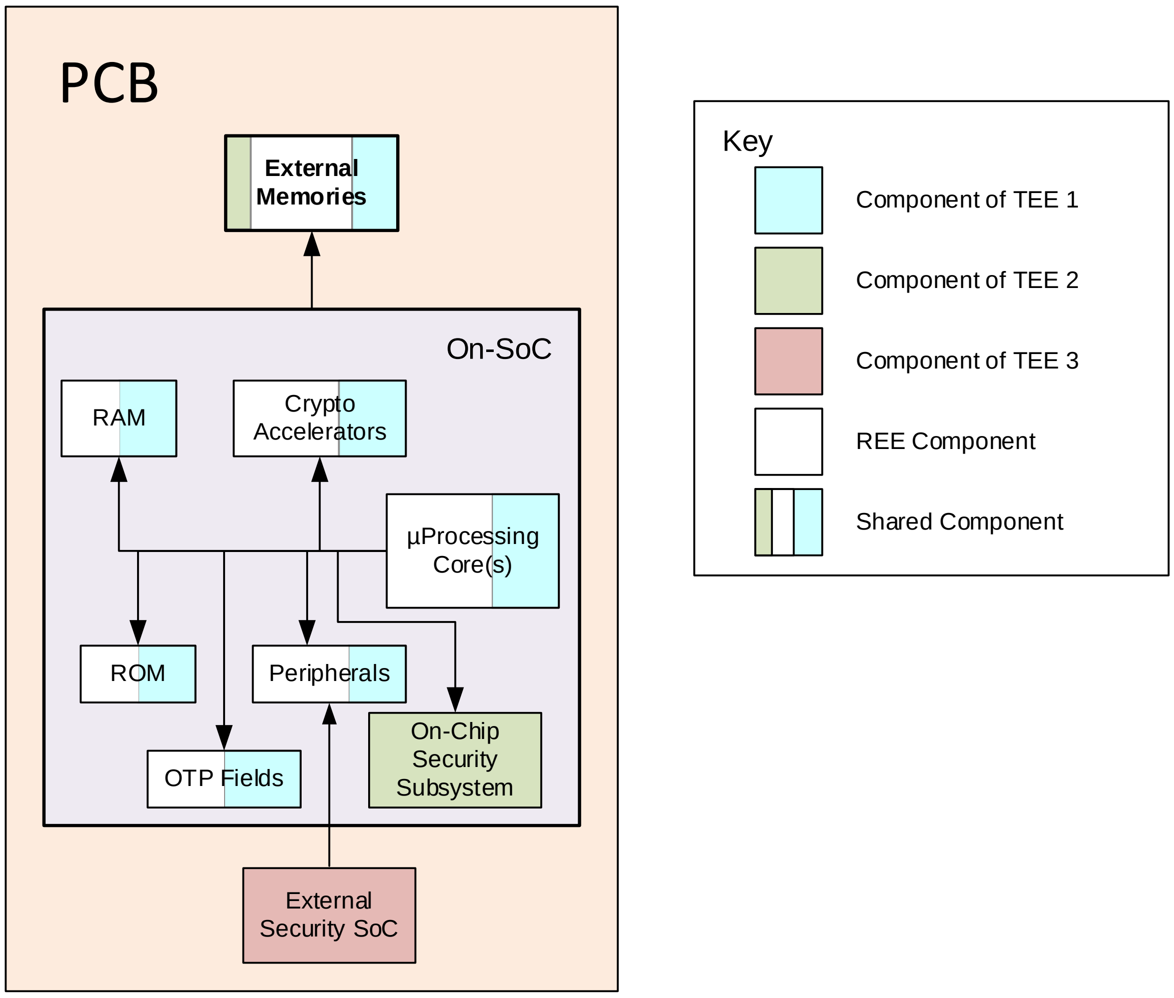
- This is one example configuration of a proprietary extension of a compliant GPD TEE, and other configurations can exist.
- There is no specified limitation on the number of OSes in an REE or the number of TEEs in one device.
- Shared trusted peripherals (as illustrated and described in Figure 3-1)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

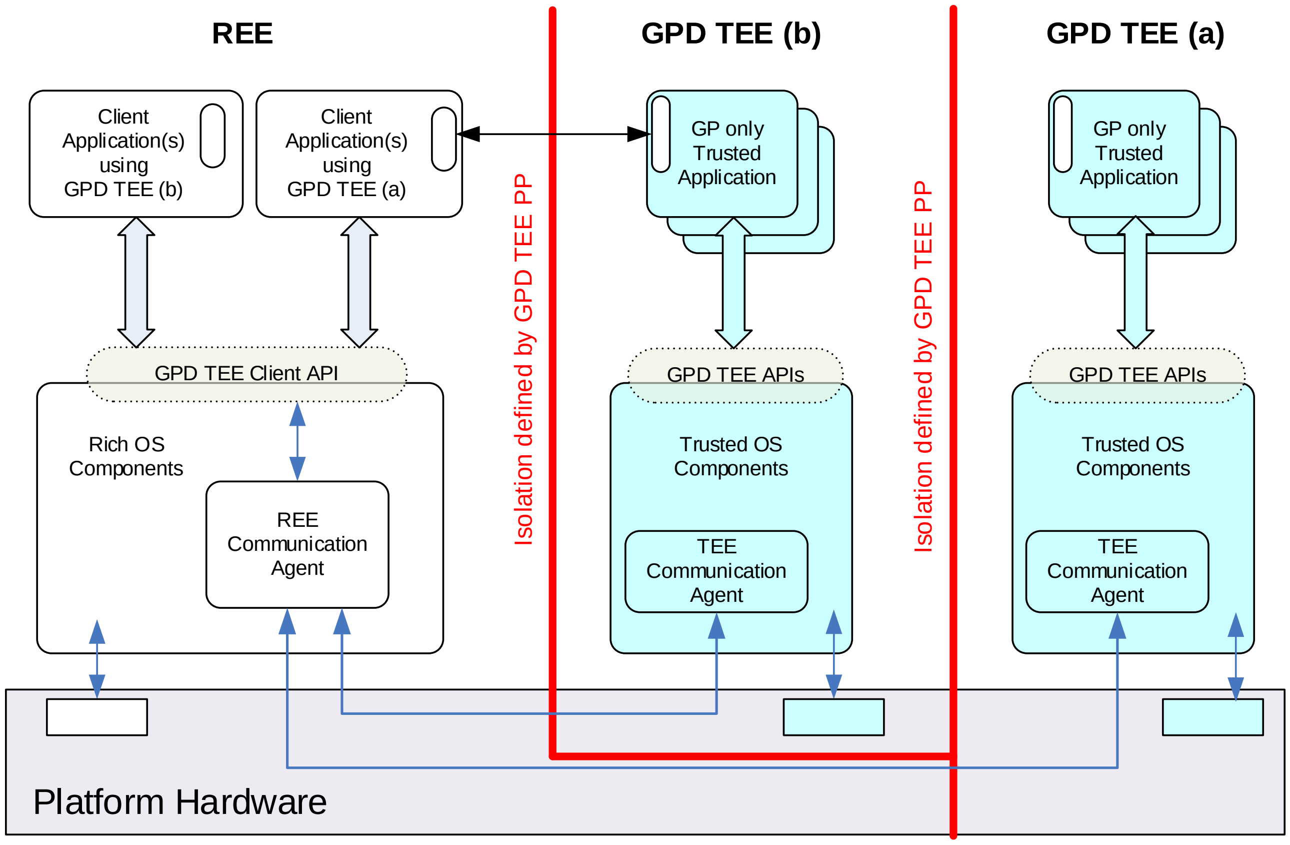
This does not prevent the GPD TEE from sharing resources with other TEEs in much the same way that it can share with the REE. For example, a Trusted User Interface is typically owned in an untrusted mode by the REE and only taken over by the TEE and put in a trusted state when needed. With multiple TEEs, 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 is treated by a Trusted Application on the initial supposition that the endpoint is untrusted (in the same way that a TA is designed to treat anything outside its local 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 Rich OS (i.e. its BIOS, UEFI, etc.) to choose which GPD TEE it uses. Potentially in a system with more than one Rich OS, each can choose to use a different TEE.

Figure 3-11: Multiple GPD TEEs in One Device



Please note:

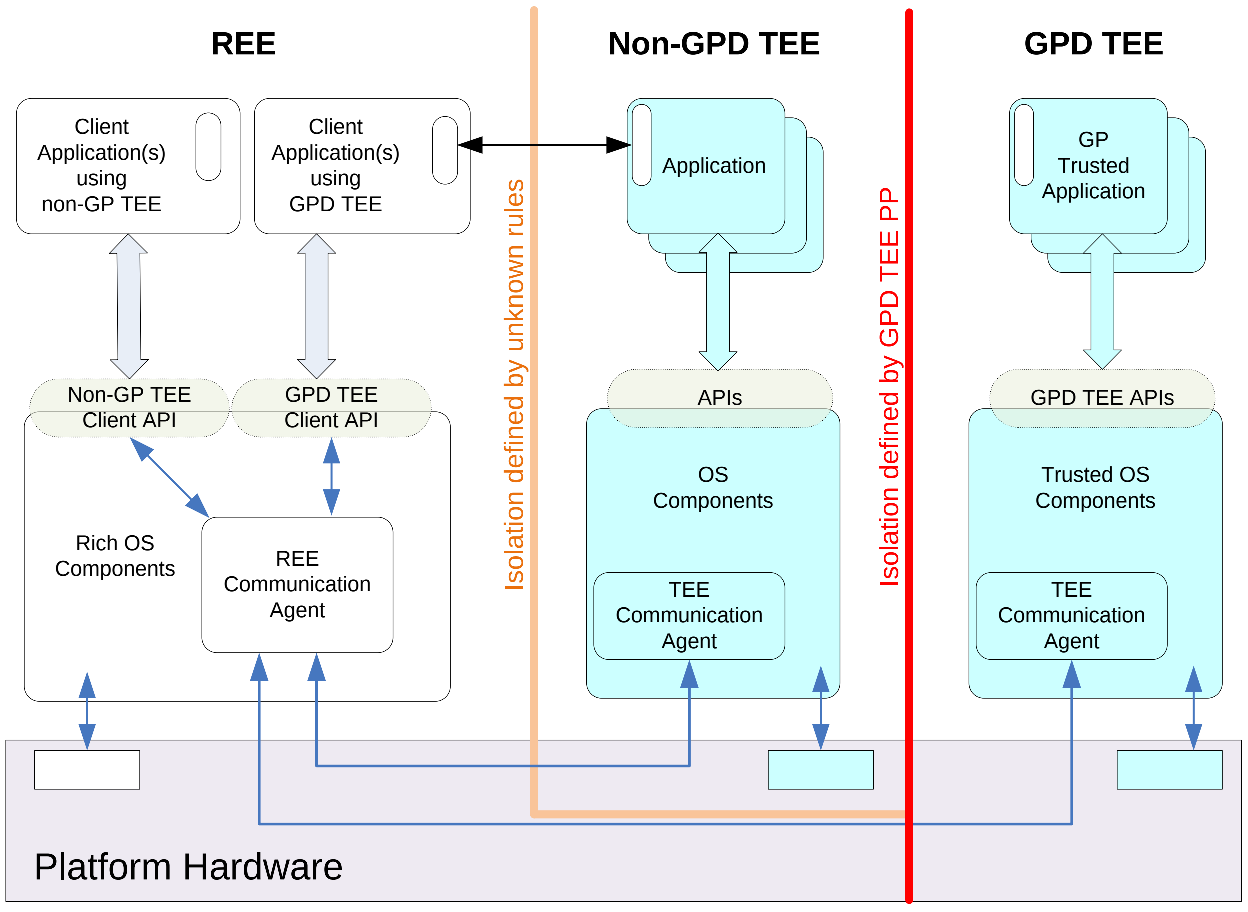
- This is one example configuration of a system with two GPD TEEs, and other configurations can exist.
- There is no specified limitation on the number of TEEs and OSes in the REE in one device.
- Shared trusted peripherals (as illustrated and described in Figure 3-1) are possible in the configuration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

4 TEE Management

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

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.

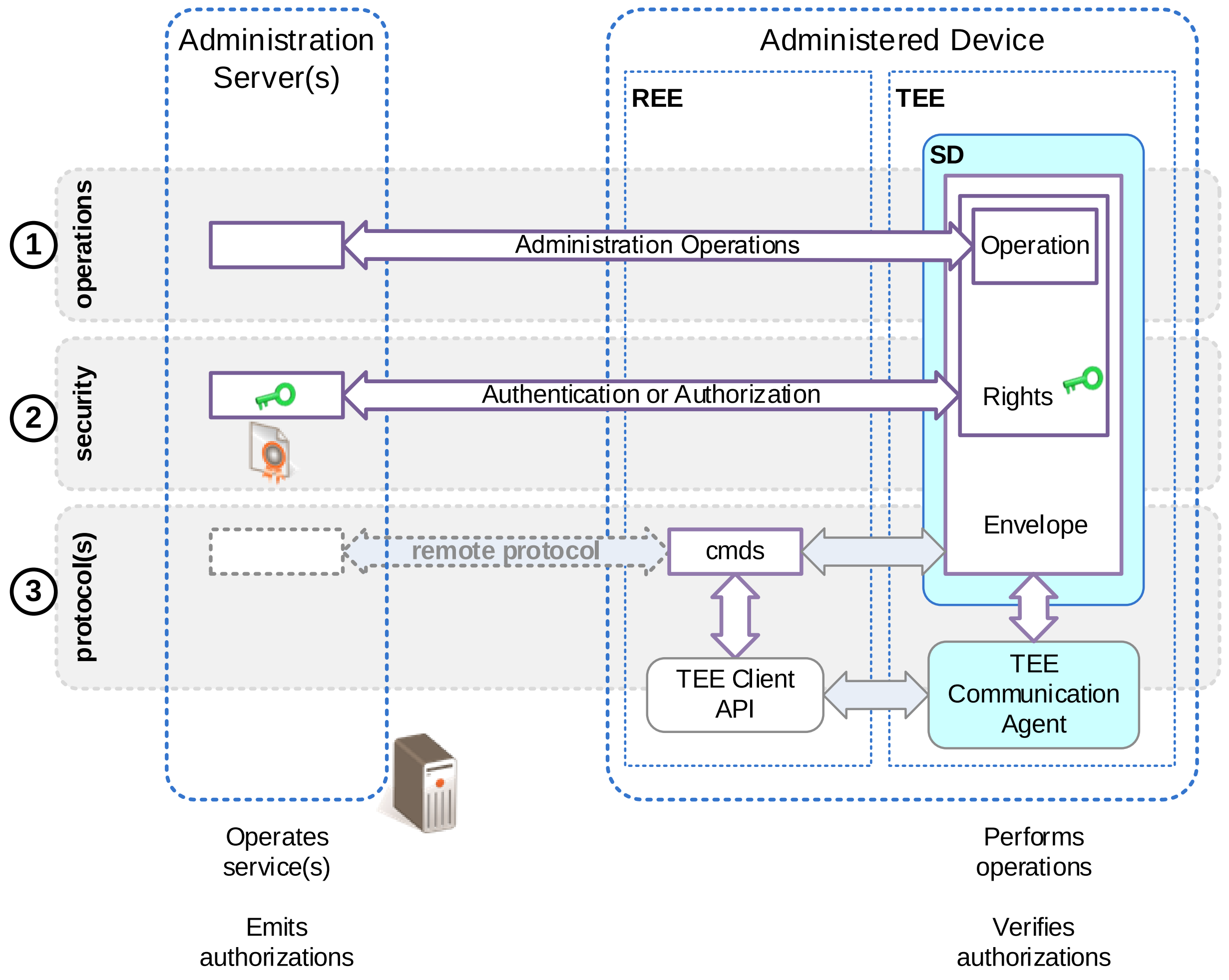
Security model

- 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

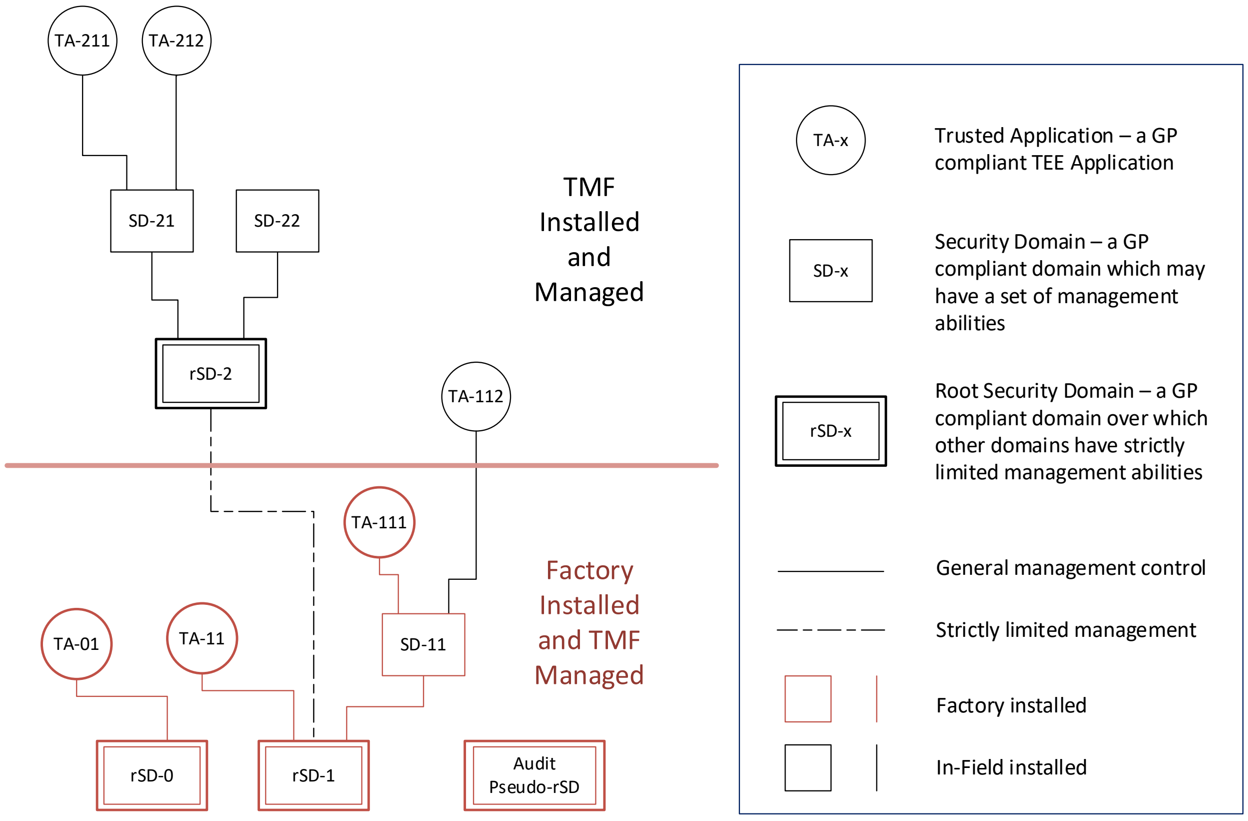
- Defines the command set (over the TEE Client API) to be used to perform administration operations.
- Defines the command set to be used to establish a secure session with a Security Domain.

Figure 4-1: TEE Management Framework Structure



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 ([TEE Mgmt]).

Figure 4-2: Security Domain Management Relationships


TA-211
TA-212
SD-21
SD-22
rSD-2
TMF Installed and Managed
TA-112
TA-111
Factory Installed and TMF Managed
TA-01
TA-11
SD-11
rSD-0
rSD-1
Audit Pseudo-rSD
TA-x
Trusted Application – a GP compliant TEE Application
SD-x
Security Domain – a GP compliant domain which may have a set of management abilities
rSD-x
Root Security Domain – a GP compliant domain over which other domains have strictly limited management abilities
General management control
Strictly limited management
Factory installed
In-Field installed

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 [TEE Mgmt].

[TEE Mgmt] 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 have limits on available storage resources and these limits affect the numbers of TAs and SDs that might be deployed on that platform.

[TEE Mgmt] defines various SD and TA management operations such as installation, removal, updating, blocking, and personalization. Particular platforms 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.

Future specifications from GlobalPlatform are expected to provide defined configurations for particular sorts of devices (e.g. IoT and smartphone), enabling 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.

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 [TEE Mgmt] 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 [TEE Mgmt] 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 [TEE Mgmt].

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 Rich 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 Rich 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, which typically is 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 that is 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.

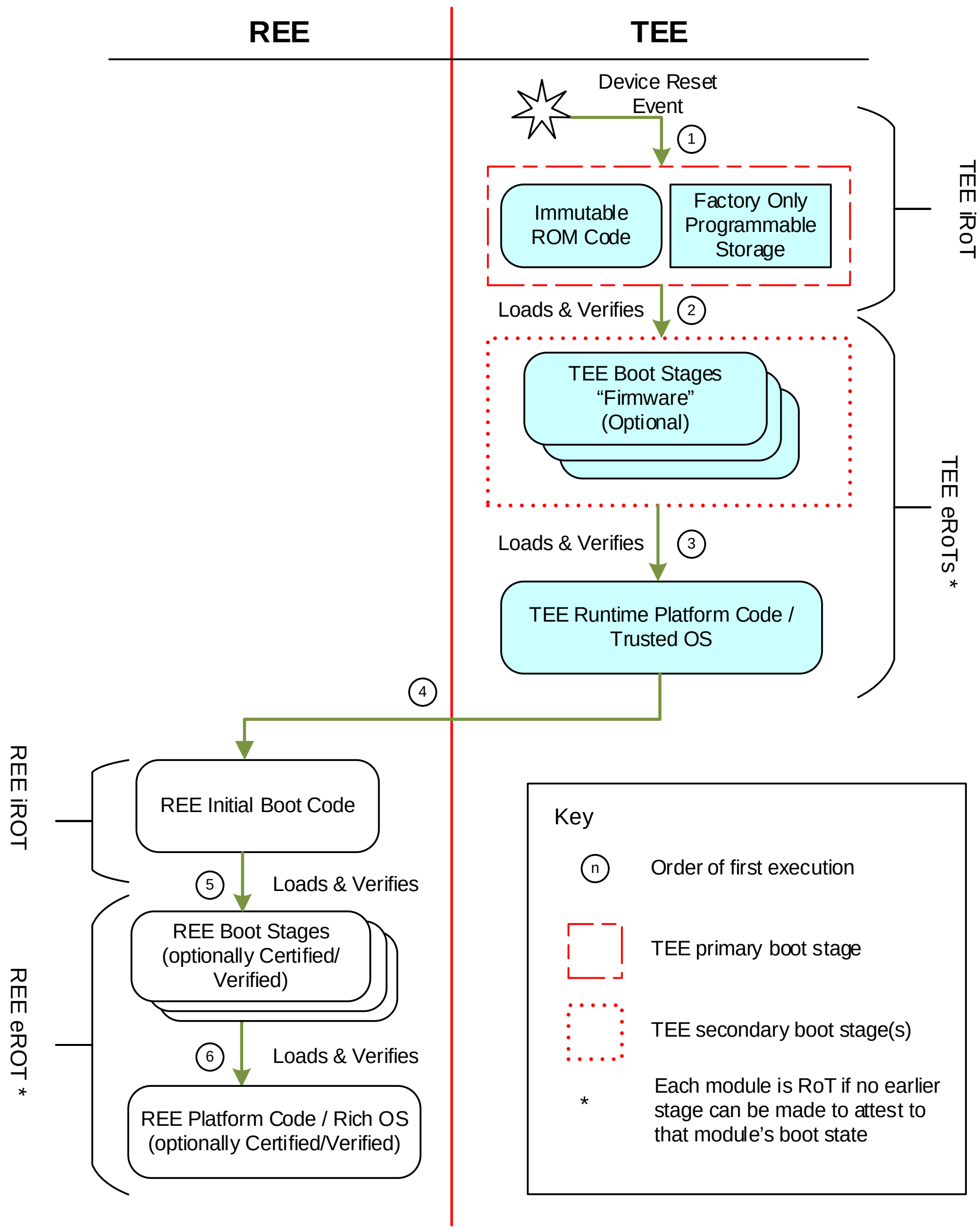
5.2.1 Typical Boot Sequence

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 SoC containing the TEE (which might not be the SoC 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.

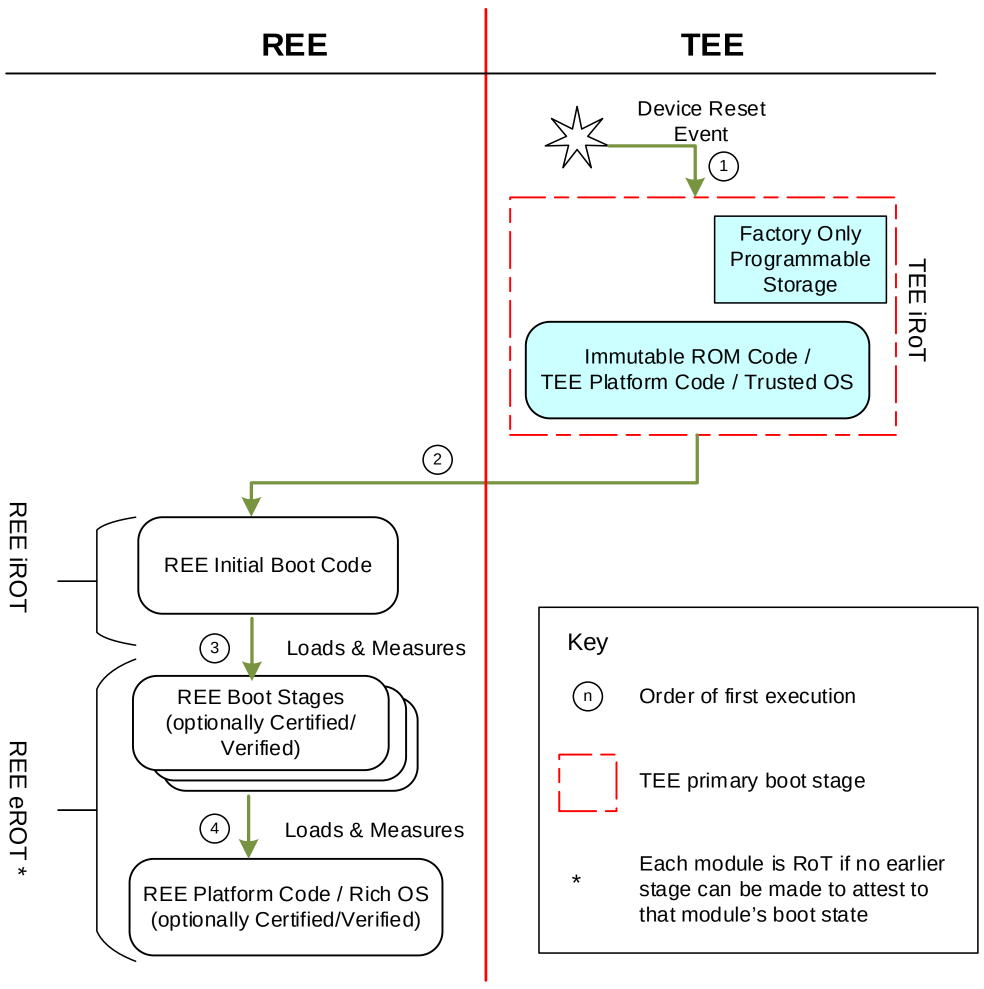
Before exiting the secure boot process, the firmware or the TEE platform code loads and can verify REE boot loader(s) before their execution. Typically, if any loaded software component verification fails up to this point, the boot process halts and the device reboots with a possible error report/indication.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

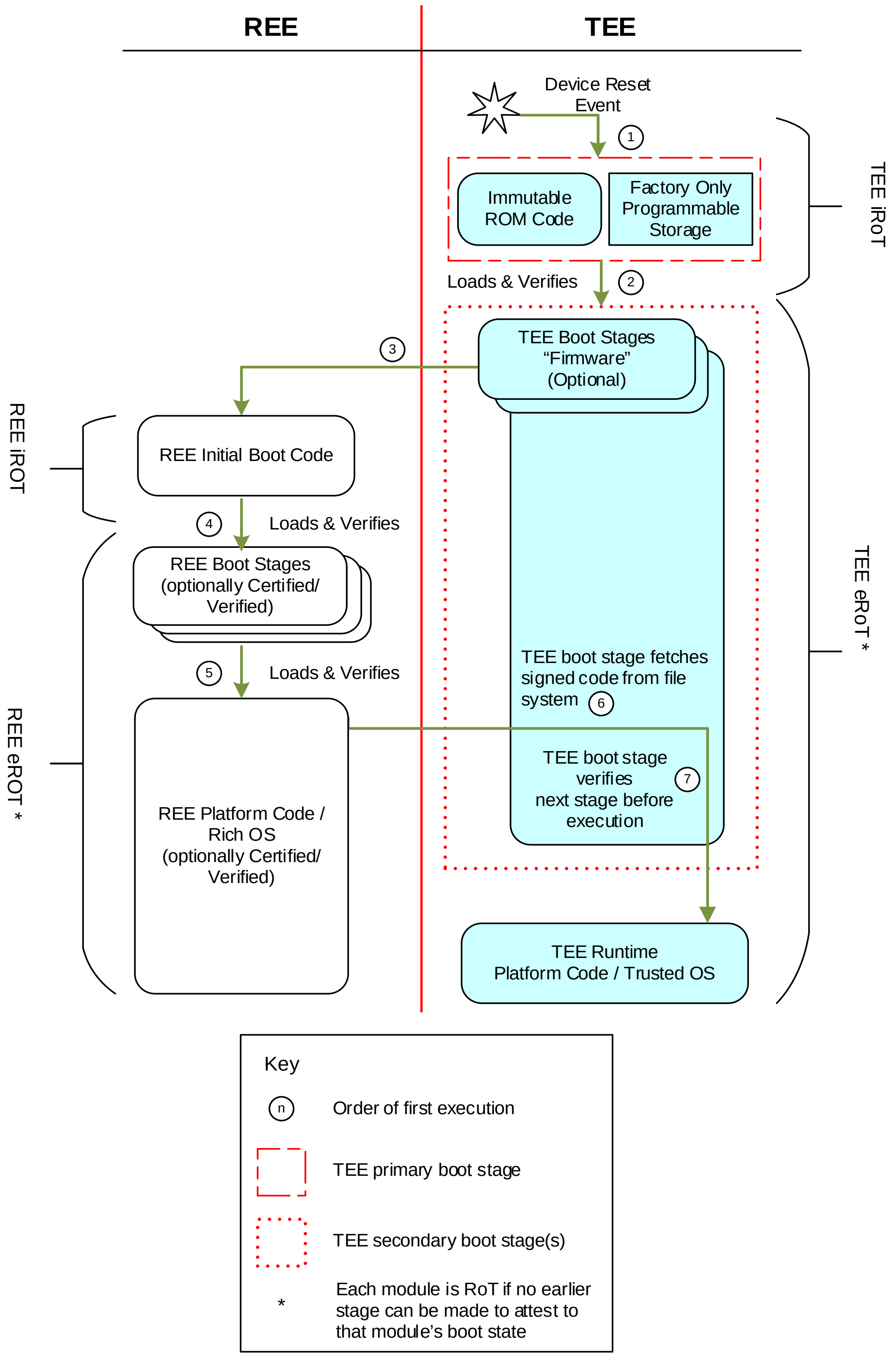
673 **Figure 5-1: Boot Sequence: Trusted OS Early Boot**


REE
TEE
Device Reset Event
1
Immutable ROM Code
Factory Only Programmable Storage
TEE iRoT
Loads & Verifies
2
TEE Boot Stages "Firmware" (Optional)
Loads & Verifies
3
TEE Runtime Platform Code / Trusted OS
TEE eRoTs *
4
REE iROT
REE Initial Boot Code
5
Loads & Verifies
REE Boot Stages (optionally Certified/ Verified)
REE eROT *
6
Loads & Verifies
REE Platform Code / Rich OS (optionally Certified/Verified)
Key
n
Order of first execution
TEE primary boot stage
TEE secondary boot stage(s)
*
Each module is RoT if no earlier stage can be made to attest to that module's boot state

674 **Figure 5-2: Boot Sequence: ROM-based Trusted OS**


REE
TEE
Device Reset Event
1
Factory Only Programmable Storage
Immutable ROM Code / TEE Platform Code / Trusted OS
TEE iRoT
2
REE iROT
REE Initial Boot Code
3
Loads & Measures
REE Boot Stages (optionally Certified/ Verified)
REE eROT *
4
Loads & Measures
REE Platform Code / Rich OS (optionally Certified/Verified)
Key
n
Order of first execution
TEE primary boot stage
*
Each module is RoT if no earlier stage can be made to attest to that module's boot state

675 **Figure 5-3: Boot Sequence: Trusted OS On-demand Boot**


REE
TEE
Device Reset Event
1
Immutable ROM Code
Factory Only Programmable Storage
TEE iRoT
Loads & Verifies
2
3
TEE Boot Stages "Firmware" (Optional)
REE iROT
REE Initial Boot Code
4
Loads & Verifies
REE Boot Stages (optionally Certified/ Verified)
5
Loads & Verifies
TEE eRoT *
TEE boot stage fetches signed code from file system
6
TEE boot stage verifies next stage before execution
7
REE eROT *
REE Platform Code / Rich OS (optionally Certified/ Verified)
TEE Runtime Platform Code / Trusted OS
Key
n
Order of first execution
TEE primary boot stage
TEE secondary boot stage(s)
*
Each module is RoT if no earlier stage can be made to attest to that module's boot state

676

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

5.3.1 TEE Functionality Availability

TEE functionality and availability can have dependencies on the REE.

The TEE functionality (i.e. providing GlobalPlatform compliant response to Client API or Internal API commands) is guaranteed to be available whenever the REE is available for REE Client Applications.

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.