GLOBALPLATFORM™

THE STANDARD FOR MANAGING APPLICATIONS ON SECURE CHIP TECHNOLOGY

GlobalPlatform Device Technology

TEE System Architecture

Version 1.0

Public Release

December 2011

Document Reference: GPD_SPE_009

This page intentionally left blank.

Contents

1 Introduction 5
1.1 Audience 5
1.2 IPR Disclaimer 5
1.3 Normative References 6
1.4 Terminology and Definitions 7
1.5 Abbreviations and Notations 10
1.6 Revision History 11
2 TEE Device Architecture Overview 12
2.1 Typical Chipset Architecture 12
2.2 Hardware Architecture 13
2.2.1 TEE Resources 14
2.2.2 REE and TEE Resources Sharing 15
3 TEE Software Interfaces 17
3.1 The TEE Software Architecture 17
3.1.1 REE Interfaces to the TEE 18
3.1.2 Trusted OS Components 18
3.1.3 Trusted Applications 18
3.1.4 Shared Memory 19
3.2 The TEE Client API Architecture 20
3.3 The TEE Internal API Architecture 21
4 TEE API Availability 22
4.1 Device States 22
4.2 Boot Time Environment 22
4.2.1 Typical Boot Sequence 23
4.3 Run-Time Environment 24
4.3.1 TEE Functionality Availability 24

Figures

Figure 2-1: Chipset Architecture 12
Figure 2-2: Hardware Architectural View of REE and TEE 15
Figure 2-3: Example Realizations of TEE 16
Figure 3-1: TEE System Architecture 17
Figure 4-1: Simplified Platform Boot Sequence 23

Tables

Table 1-1: Normative References 6
Table 1-2: Terminology and Definitions 7
Table 1-3: Abbreviations 10
Table 1-4: Revision History 11

1 Introduction

Modern devices, such as the Smartphone, offer a Rich Execution Environment (REE), providing a hugely extensive and versatile operating environment. This brings flexibility and capability, but at the same time leaves that 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 the security concepts involved and finally it explains some concepts relevant to the TEE functional availability in a device.

At the highest level, a Trusted Execution Environment (TEE) is an environment where the following are true:

- Any code executing inside the TEE is trusted in authenticity and integrity.
- The other assets are also protected in confidentiality.
  - The TEE shall resist to all known remote and software attacks, and a set of external hardware attacks.
- Both assets and code are protected from unauthorized tracing and control through debug and test features.

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 any possible implementation as long as the security principles are adhered to. Hence, any hardware or software architectural diagram in this document should be taken as an example and for reference only.

This TEE System Architecture is covering the first phase of a TEE standardization. Extension of the TEE System Architecture is expected in a second phase, as described in the TEE White Paper [2], e.g. the lifecycle of the Trusted Application.

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 the means of the TEE Client API

1.2 IPR Disclaimer

GlobalPlatform draws attention to the fact that claims that compliance with this specification may involve the use of a patent or other intellectual property right (collectively, “IPR”) concerning this specification may be published at https://www.globalplatform.org/specificationsipdisclaimers.asp. GlobalPlatform takes no position concerning the evidence, validity, and scope of these IPR claims.

1.3 Normative References

Table 1-1: Normative References

Standard / Specification	Description	Ref
OMTP ATE TR1	Open Mobile Terminal Platform (OMTP) Advanced Trusted Environment TR1 v1.1	[1]
TEE White Paper	The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market, GlobalPlatform White paper, Feb 2011	[2]
TEE Client API Spec	GlobalPlatform TEE Client API Specification v1.0, GlobalPlatform, July 2010	[3]
TEE Internal API Spec	GlobalPlatform TEE Internal API Specification v1.0, GlobalPlatform, December 2011 Note that this document is to be released at the same time as the TEE System Architecture	[4]
TEE Security Requirements	GlobalPlatform TEE Security Requirements v1.0, GlobalPlatform, [TBD] Note that this document is still under development and will be released in due course. Forward reference is made here, to indicate the future location of the detailed information relating to the TEE security requirements.	[5]

1.4 Terminology and Definitions

Table 1-2: Terminology and Definitions

Term	Definition
Application Programming Interface	A set of rules that software programs can follow to communicate with each other.
Boot time Non-trusted Execution Environment (BNEE)	A boot time execution environment that does not meet all the requirements or principles of a TEE, and that may be used to instigate an REE.
Boot time Trusted Execution Environment (BTEE)	A boot time execution environment that is protected at the same level as a TEE and may be used to instigate a TEE.
Boot time Execution Environment (BEE)	Any execution environment that exists prior to the loading and start of the REE. A BEE is terminated during the usual execution of a device and is solely present for providing support to enable the starting of another EE. May be either of the following: • Boot time Non-trusted Execution Environment (BNEE), which enables the start of an REE • Boot time Trusted Execution Environment (BTEE), which enables the start of a 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	Core migration is 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 [1], 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
In-package	While the default hardware security boundary is often described as on-SoC, in reality it is the SoC packaging material that forms the boundary. It is important to make this distinction because it enables the use of more than one die inside a package, and hence to place more facilities inside the hardware security boundary. These extra facilities would not be considered “on-SoC” but are considered “in-package”.

Term	Definition
One Time Programmable	A form of memory that may be read many times, but only written once. On a typical SoC implementing a TEE, this may 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.
Physical isolation	In this document, unless stated otherwise for particular assets, physical isolation of security related assets is considered to include isolation by electronic access control through the TEE system hardware, that may be configured by TEE resident boot or run-time software.
REE Communication Agent	An Rich OS driver that enables communication between REE and TEE. Contrast *TEE Communication Agent*.
Rich Execution Environment (REE)	An environment that is provided and governed by a Rich OS, potentially in conjunction with other supporting operating systems and hypervisors; it is outside of the TEE. This environment and applications running on it are considered un-trusted. 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 un-trusted, though from the Rich OS point of view there may be internal trust structures. Contrast *Trusted OS*.
Trusted storage	In GP TEE documents, *trusted storage* indicates storage that is protected to at least the robustness level defined for OMTP Secure storage (in section 5 of OMTP ATE TR1 [1]). 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 GP TEE Trusted storage is not considered hardware tamper resistant to the levels achieved by Secure Elements, but it is bound to the device
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*.
TEE Communication Agent	Trusted OS driver that enables communication between REE and TEE. Contrast *REE Communication Agent*.
TEE Service Library	A software library that includes all security related drivers.
Trusted Application (TA)	An application running inside the Trusted Execution Environment that provides security related functionality to Client Applications outside of the TEE. Contrast *Client Application*.

Term	Definition
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, see OMTP ATE TR1 [1].) Contrast *Rich Execution Environment*.
Trusted Functions	A Trusted OS components dedicated to exposing an inbuilt TEE capability (such as cryptography). This capability is in the long term roadmap for the TEE but will not be available in the initial releases
TEE Functional API	A REE software interface dedicated to exposing an inbuilt TEE capability (such as cryptography) for the CA developer. This capability is in the long term roadmap for the TEE but will not be available in the initial releases
Trusted OS	An operating system running in the TEE. It has been designed primarily to enable the TEE using security based design techniques. It provides the GP TEE Internal API to Trusted Applications and a proprietary method to enable the GP TEE Client API software interface from other EE. Contrast *Rich OS*.

1.5 Abbreviations and Notations

Table 1-3: Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
BEE	Boot time Execution Environment
BNEE	Boot time Non-trusted Execution Environment
BTEE	Boot time Trusted Execution Environment
CA	Client Application
DoS	Denial of Service
DRAM	Dynamic Random Access Memory
DRM	Digital Rights Management
EE	Execution Environment
GP	GlobalPlatform
I/O	Input/Output
IC	Integrated Circuit
IPR	Intellectual Property Rights
OEM	Original Equipment Manufacturer
OMTP	Open Mobile Terminal Platform
OS	Operating System
OTP	One Time Programmable
PCB	Printed Circuit Board
RAM	Random Access Memory
REE	Rich Execution Environment
ROM	Read Only Memory
RTC	Real Time Clock
SoC	System-on-Chip
SSL	Secure Socket Layer
TA	Trusted Application
TEE	Trusted Execution Environment
TPK	TEE Primary Key

1.6 Revision History

Table 1-4: Revision History

Date	Version	Description
December 2011	1.0	Initial publication.

2 TEE Device Architecture Overview

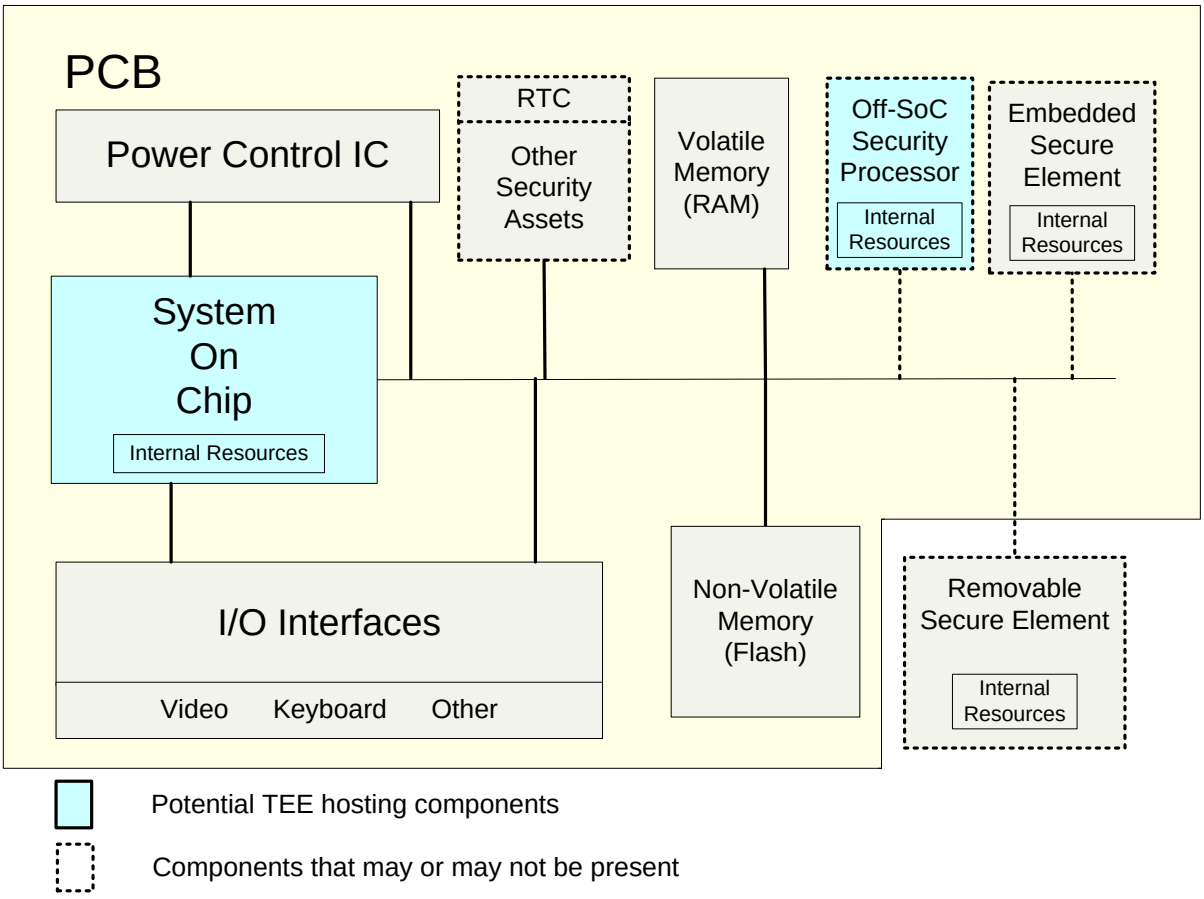
The TEE as an isolated execution environment is providing security features such as isolated execution, integrity of Trusted Applications along with confidentiality of their assets. The following chapter describes the general device architecture associated with the TEE along with a high level overview of the security requirements of a TEE.

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

2.1 Typical Chipset Architecture

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

Figure 2-1: Chipset Architecture



2.2 Hardware Architecture

Both the REE and the TEE utilize a number of dedicated resources such as processing core(s), RAM, ROM, cryptographic accelerators, etc. Figure 2-1 provides an example of the resources that may be found at a device level. Figure 2-2 provides an example of the resources that may be associated with a package such as the System-on-Chip in Figure 2-1. The entities other than the processing core(s) are referred to as the resources. Some resources accessible by the REE may also be accessible by the TEE whereas the opposite must not hold unless explicitly authorized by the TEE. The trusted resources are only accessible by other trusted resources and thereby make up a closed system that is protected from the REE. This implies, for example, that trusted RAM is only addressable from the trusted processing unit.

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

The high level security requirements of a TEE can be stated below:

- The primary purpose of the TEE is to protect its assets from the REE
  - This is achieved through hardware mechanisms that the REE cannot control.
- This protection must be extended to protection against other execution environments whose software, or location, is deemed untrustworthy.
- The TEE must be protected against a range of physical attacks.
  - Typically this protection will be a level less than that found to dedicated tamper resistant technology such as Secure Elements.
  - In-package intrusive attacks are outside of the scope of TEE protection.
- The debug facilities of a production TEE (or system components capable of accessing assets in the TEE) must either be disabled, or be controlled by an element that itself meets, or exceeds, the security requirements of the TEE.
  - This requirement places no restrictions on debug capabilities for system components (including the REE) that cannot access assets of the TEE
- The TEE must be instantiated through a secure boot process using assets bound to the SoC and isolated from the REE.
  - The integrity and authenticity, gained through secure boot, must extend throughout the lifetime of the TEE.
  - This integrity and authenticity must be retained through any state transitions in the system such as power transitions, or core migration.
- Trusted Storage of data and keys is provided by the TEE
  - The Trusted Storage must be bound to the device such that no unauthorized internal or external attacker may access, copy or modify the data contained.
    - The strength of this protection must be at least equal to that of the TEE environment.
  - The Trusted Storage must provide a minimum level of protection against rollback attacks
    - It is accepted that the actually physical storage may be in an unsecure areas and so is vulnerable to actions from outside of the TEE.

- Software in the REE must not be able to call directly to TEE Functions or Trusted Core Framework.
  - The REE software must go through protocols such that the Trusted OS or Trusted Application performs the verification of the acceptability of the operation that the REE software has requested.
- A GlobalPlatform TEE will be based on the security requirements described in OMTP ATE TR1 [1] Profile 2.

Detailed definition of these requirements will be found in TEE Security Requirements [5]

2.2.1 TEE Resources

There are three classes of resources in use by a TEE:

Primary TEE resources – In-package resource

All TEE primary trusted resources must be implemented in-package, which means that they will be protected from physical attacks by the processor packaging.

Secondary TEE resources – Off-package resource

An exception to the in-package requirement can be trusted replay-protected external non-volatile and trusted volatile memory areas. For these memory areas the security must be fulfilled by using proven cryptographic methods. Only TEE will have capability to decrypt and access the plaintext content stored in these otherwise un-trusted addressable memory locations.

When off-package resources are used to achieve security properties (e.g., trusted monotonic counters, real time clock), the TEE must implement protections against tampering with communication to such resources, and the package level protections of these resources must be as good as those of the TEE primary trusted resources.

Tertiary TEE resources – exposed or partially exposed resources

A further exception is that TEE controlled trusted areas of external components may contain data not guarded by a proven cryptographic method, but only at the discretion of the relevant Trusted Application. This is needed to:

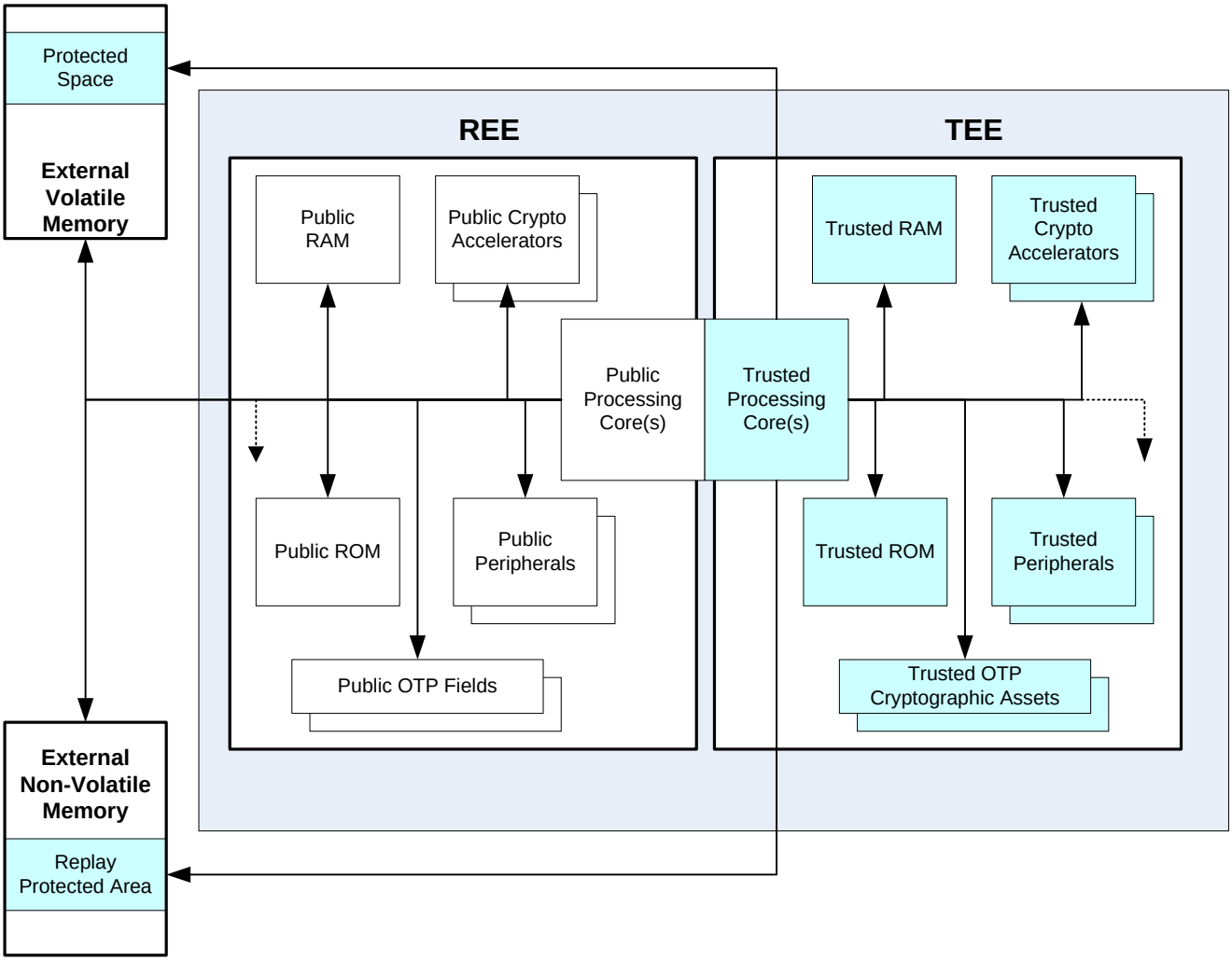
- Enable trusted DRAM-based buffers where the data is in the clear but must not be attacked by REE software while being manipulated (e.g., SSL or DRM stream buffers)
- Provide space for a trusted screen frame store

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

2.2.2 REE and TEE Resources Sharing

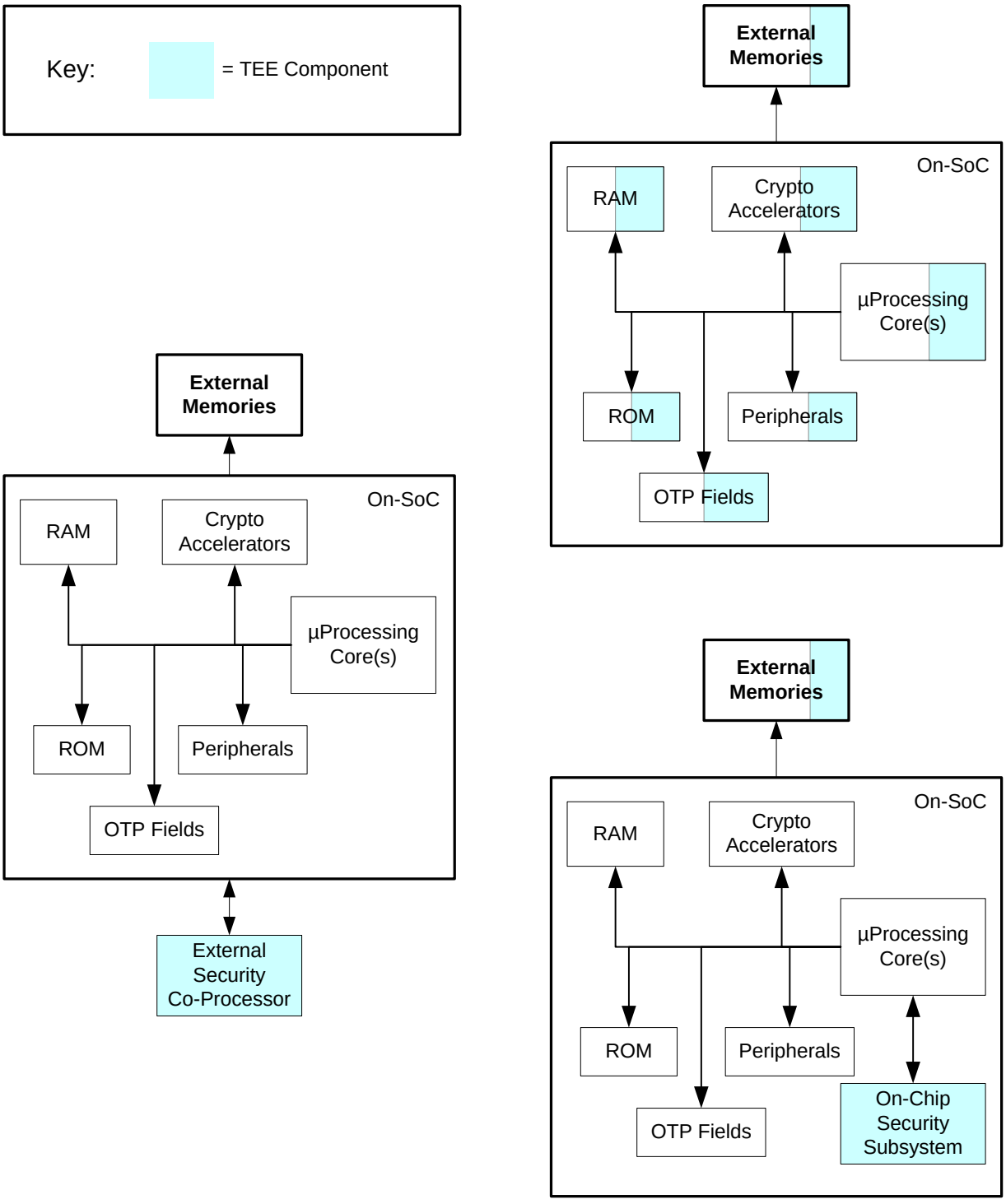
The REE has access to the un-trusted resources, which may be implemented on-chip or off-chip in other components on the PCB. The REE cannot access the trusted resources. This access control must be enforced and can potentially be implemented by physical isolation of trusted resources from un-trusted resources. 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 in a controlled and protected manner and vice versa.

Figure 2-2: Hardware Architectural View of REE and TEE



Note that the architectural view of TEE and REE as 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.

Figure 2-3: Example Realizations of TEE



3 TEE Software Interfaces

The TEE is a separate execution environment that runs alongside the Rich OS and provides security services to that rich environment and applications running inside the environment. The TEE exposes sets of APIs to enable communication from the REE and others to enable Trusted Application software functionality within the TEE.

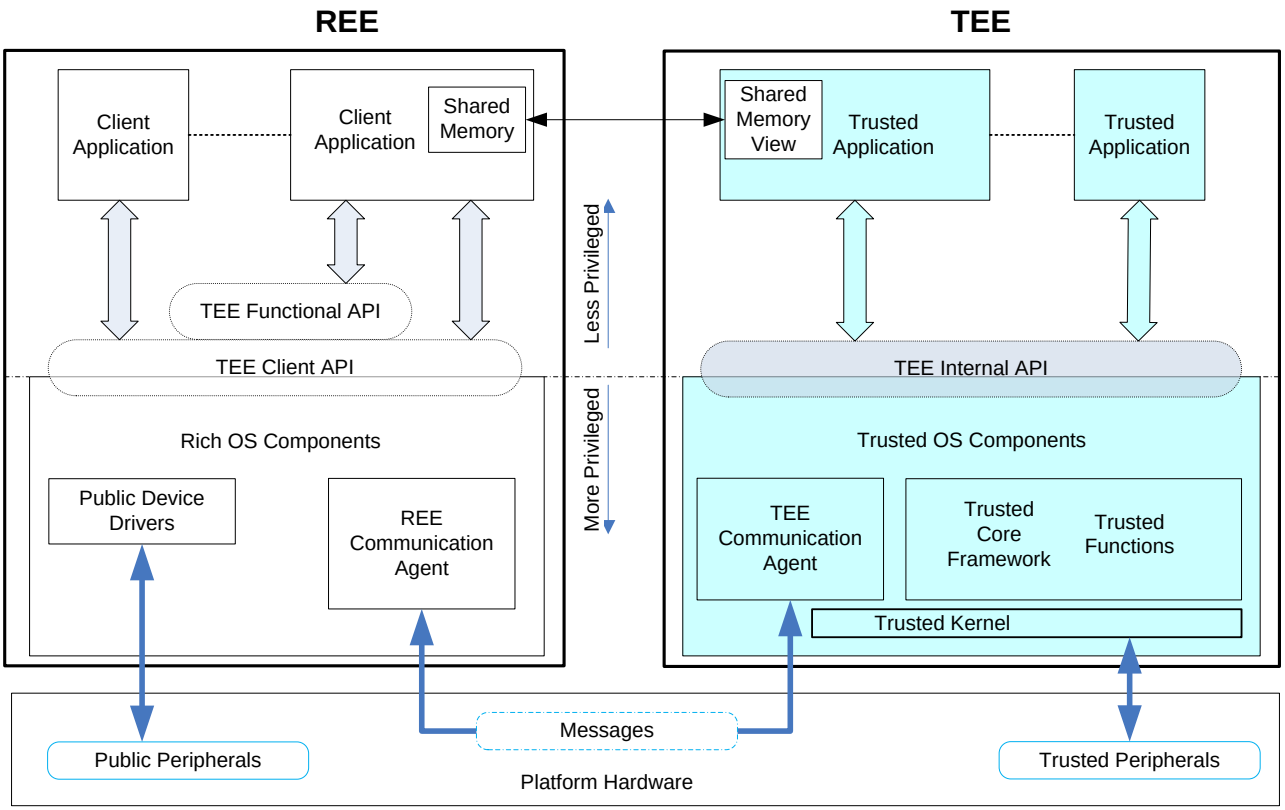
The following chapter describes the general software architecture associated with the TEE, the interfaces defined by GlobalPlatform and the relationship between the critical components found in the software system.

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

3.1 The TEE Software Architecture

The relationship between the major software systems components is outlined in the block architecture below (Figure 3-1).

Figure 3-1: TEE System Architecture



The goal of the TEE Software Architecture is to enable Trusted Applications (TA) to provide isolated and trustworthy capabilities for service providers, which can then be used through intermediary Client Applications (CA).

3.1.1 REE Interfaces to the TEE

Within the REE, the architecture identifies two API interfaces and a supporting communication agent.

- The TEE Functional API will offer Client Applications a set of Rich OS-friendly APIs. These will allow access to some TEE services, such as cryptography or trusted storage, with a programming model familiar to Rich OS application developers. The definition of the TEE Functional API is part of the GlobalPlatform TEE deliverables roadmap.
- 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 REE Communication Agent provides REE support for messaging between the Client Application and the Trusted Application.

3.1.2 Trusted OS Components

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

Trusted OS Components consist of

- The Trusted Core Framework provides the OS like functionality to Trusted Applications
  - See Trusted Core API in Section 3.3.
- The Trusted Functions provides the supporting facilities for application developers.

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

- The TEE communication agent is a special case of a Trusted Core Framework function API. It works with its peer, the REE Communication Agent, to safely transfer messages between CA and TA.

3.1.3 Trusted Applications

The Trusted Applications interface to the rest of the system via APIs exposed by Trusted OS components.

- The TEE Internal API defines the fundamental software capabilities of a TEE
- Other APIs may be defined to support interface to further proprietary Trusted Functions

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 space which is separated from the physical memory 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.

3.1.3.1 Trusted Application Lifecycle

From the point of view of this document, the ownership of the TEE is typically defined by the ability to directly or indirectly manage Trusted Applications in the TEE.

This is currently proprietary, however in later phases GlobalPlatform standards will be defined to

- Enable service providers to discover the owner or owners of the TEE
- Discover what in-built services a particular TEE may offer
- Request download of a TA to a specific device
- Manage the lifecycle of TAs installed on a specific device

Please see the GlobalPlatform TEE roadmap (Section 6 of TEE White Paper [2]) for later the timelines for initial standardization work to be applied in this area.

3.1.4 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 REE. The API design allows this feature to be implemented by the Trusted OS as either memory copies, or by 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 and Internal APIs.

Care should 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 The TEE Client API Architecture

GlobalPlatform specifies the TEE Client API in the GlobalPlatform TEE Client API Specification [3]. 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 may be built on top of the foundation provided by the TEE Client API – for example, to cover 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 whose role is to handle the communications between the REE and the TEE.

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

More information on the TEE Internal API can be found in the specification [3]

3.3 The TEE Internal API Architecture

GlobalPlatform specifies the TEE Internal API in the GlobalPlatform TEE Internal API Specification [4]. The TEE Internal API concentrates on the various interfaces to enable a Trusted Application to make best use of the standard TEE capabilities.

Higher level standards and protocol layers may be built on top of the foundation provided by the TEE Internal API – for example, to cover common tasks such as 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 the TEE Internal API.
- The TEE Internal API library implementation.
- The Trusted OS Components, which are shared amongst all Trusted Applications, and whose role is to provide handle the system level functionality required by the Trusted Applications.

The TEE Internal API provides a number of different sub-sets of functionality to the Trusted Application.

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 DRM.
TEE Arithmetical API	This API provides arithmetical primitives to create cryptographic functions not found in the Cryptographic API.

More information on the TEE Internal API can be found in the specification [4].

4 TEE API Availability

The TEE and its capabilities will be closely coupled to those of the REE and the state of the device it resides in. It is therefore important for the developer of those 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 may be found in typical devices. As such this chapter describes the notions of Run Time Environment, clarifies the Boot Time Environment and lists some of the possible device states. Some clarifications are given regarding the dependencies and the availability of the TEE functionalities with respect to the Rich OS.

4.1 Device States

Devices implementing a TEE can be found in a number of states that are outside of the standard, but are still useful for the developer be aware of.

Devices implementing the TEE must provide trusted mechanisms to indicate the state of the device and control the corresponding security environments and transitions.

Examples of the some such states:

- Devices in manufacturing, which may offer neither security nor functional compliant at various stages of their creation.
- Development devices, which may, or may not, have reduced security but should provide TEE compliant functionality.
- Production devices, which must provide TEE compliant functionality and security.
- And finally, devices that have somehow failed, and which must block access to TEE held user data, while enabling various levels of debug access through secure mechanisms.

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

Life cycle state changes must not lower the security of the TEE.

4.2 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 EE 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.

As indicated in OMTP ATE TR1 [1], it is assumed that 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. Hence a minimal set of the TEE capabilities must exist during the boot time and to enable some of these services a Trusted OS (or some simplified version thereof) may also exist. Such a minimal set of TEE capabilities is referred to here as a Boot Time TEE (BTEE).

It is not the current intention of GlobalPlatform to define the capabilities of a BTEE, however if a 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 API as it is capable of providing.

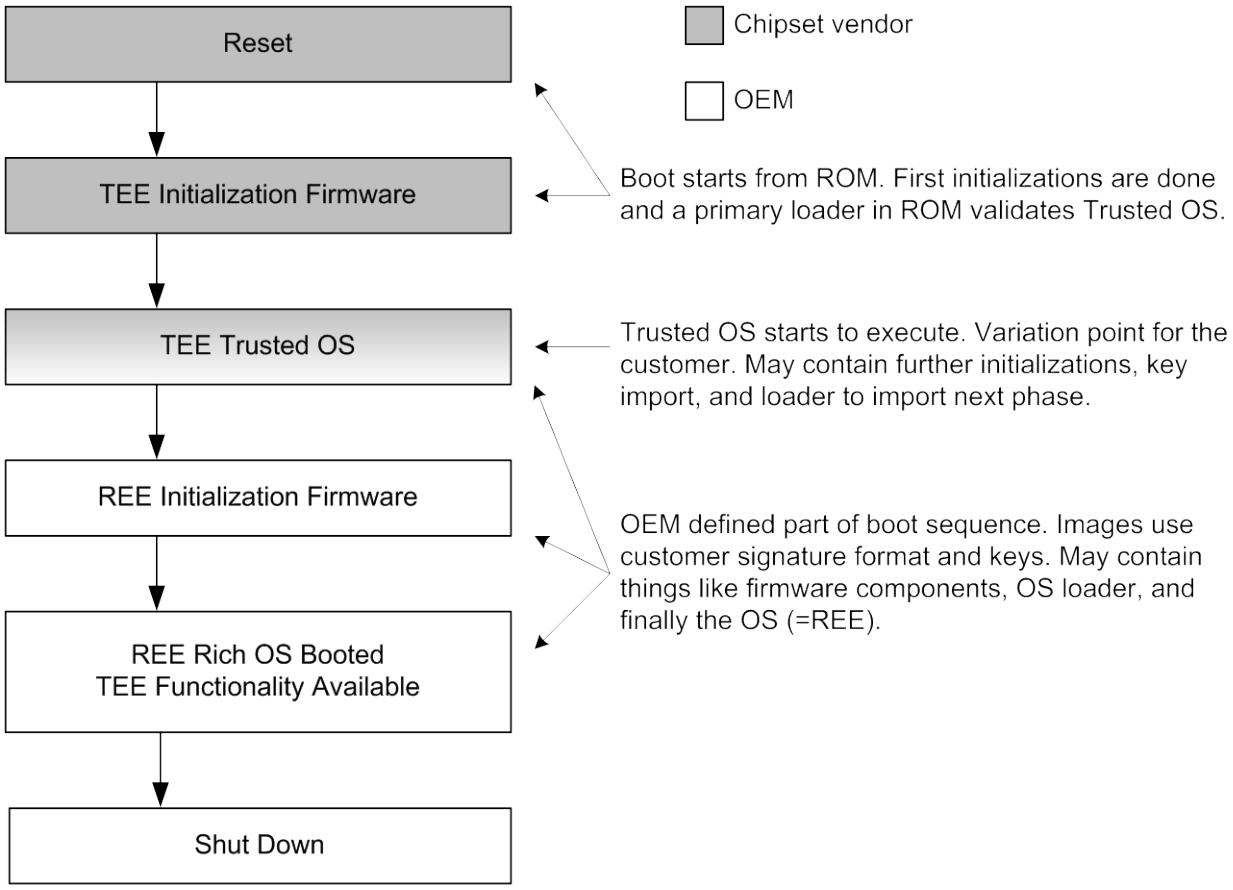
4.2.1 Typical Boot Sequence

A simplified example boot sequence flow diagram is illustrated in Figure 4-1. This flow diagram illustrates how OEMs may differentiate by implementing trusted firmware to be run early in the boot sequence. This gives the OEM the flexibility to bring in their own keys, certificate format, signature schemes, etc. Alternatively, an OEM may implement a longer boot sequence that is tied to chipset supplier implemented code. While this gives a certain level of OEM flexibility, the minimum TEE functional availability is always as described in section 4.3.1

In order to maintain the chain of trust, it is critical that the TEE is instantiated either:

- during the secure boot sequence, or
- at a later stage by a software component which is itself instantiated, and trusted to the same level as the TEE, by the boot sequence.

Figure 4-1: Simplified Platform Boot Sequence



4.3 Run-Time Environment

The term “run-time” refers to a property of the overall execution environment where an EE operating system has fully completed its initialization/boot operations and is fully operational, as opposed to the duration where the EE operating system is not fully operational as explained in section 4.2.

The dependencies between the Trusted OS and the Rich OS are implementation dependent. This current GlobalPlatform specification targets to standardize the behavior of the system, once the Rich OS is operational. This does not mean that there may not be capabilities when the Rich OS is not operational, see section 4.2

4.3.1 TEE Functionality Availability

While the TEE as a protected environment will always meet its protection requirements, its functionality and availability may have dependencies on the REE.

The TEE functionality is available when the REE is available for REE Client Applications.

It is guaranteed to work for Client Applications.

This means that effects such as power state changes, where the Client Applications are not aware of such a change, must not be noticeable via their connection to Trusted Applications.