ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. «Τεχνοοικονομική Διοίκηση και Ασφάλεια Ψηφιακών Συστημάτων»

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ

«DEVELOPMENT OF CRYPTOGRAPHIC ALGORITHMS IN THE TRUSTED EXECUTION ENVIROMENT »

Σπουδαστής:

ΠΕΡΡΩΤΗΣ ΛΕΩΝΙΔΑΣ

Επιβλέπων Καθηγητής:

Δρ. ΞΕΝΑΚΗΣ ΧΡΗΣΤΟΣ

ΙΑΝΟΥΑΡΙΟΣ 2018

ABSTRACT

Almost a decade ago, the first phones appeared with hardware based TEEs, reaching today, all modern mobile devices contain a TEE.

Despite the large-scale deployment, the use of TEEs functionalities has been limited to mobile device constructors and a closed community to develop applications for them. Moreover, the use of hardware-based TEEs in application development and research are expensive and often proprietary.

Nowadays, many industry associations like GlobalPlatform are working to standardize the specifications of the TEEs, so it is possible for any developer to create an application for TEEs, with a predefined standardization in a virtual trusted environment. This will help developers and researchers, to enhance the protection and functionality to new applications and services. This thesis deals with the development of a Trusted Application in a virtual Trusted Execution Environment, that makes use of all the security features this architecture provides. All of the tools and the development environment that are used, are common and well-known to the developer community. Finally, we would like to underline that open virtual TEE provides the ability to developers and researchers, of applications and services that have been developed, to be tested, refined and the continuation of development.

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα Μεταπτυχιακή Διπλωματική Εργασία (ΜΔΕ), εκπονήθηκε στο πλαίσιο των σπουδών μου, στο Μεταπτυχιακό Πρόγραμμα Σπουδών «Τεχνοοικονομική Διοίκηση και Ασφάλεια Ψηφιακών Συστημάτων» του τμήματος Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς.

Με τη περάτωση της παρούσας διπλωματικής εργασίας, θα ήθελα να ευχαριστήσω θερμά όλους τους καθηγητές του τμήματος της Ασφάλειας Ψηφιακών συστημάτων του Πανεπιστημίου Πειραιά, για τις πολύτιμες γνώσεις που μου μετέδωσαν, αλλά και για τις συμβουλές τους. Να ευχαριστήσω ειδικότερα τον επιβλέποντα καθηγητή μας Δρ. Χρίστο Ξενάκη, για τη βοήθεια του, τις συμβουλές του με τη καθοδήγηση του οποίου, αποφάσισα να ασχοληθώ με ένα θέμα τόσο ενδιαφέρον και ταυτόχρονα μη διαδεδομένο, καθώς και να ολοκληρώσω με επιτυχία όλους τους στόχους που είχα θέσει εξαρχής, για την εργασία αυτή.

Ιδιαίτερα όμως αισθάνομαι την ανάγκη και την υποχρέωση να ευχαριστήσω για ακόμα μια φόρα τους γονείς μου και τους ανθρώπους που με στήριξαν σε αυτό το ταξίδι της γνώσης.

Τέλος θα ήθελα να ευχαριστήσω θερμά τον συνάδελφο Εμμανουήλ Αγγελάκη, με τον οποίο συνεργαστήκαμε από τις πρώτες ήδη μέρες του μεταπτυχιακού προγράμματος μέχρι και το τέλος του. Μέσα από αυτή τη συνεργασία, αναπτύχθηκε μια πραγματική φιλία και αλληλοεκτίμηση και αισθάνομαι πολύ τυχερός με το γεγονός αυτό.

Contents

LIST OF ABBREVIATIONS

API - APPLICATION PROGRAMMING INTERFACE
BYOD - BRING YOUR OWN DEVICE
CA - CLIENT APPLICATION
DRM - DIGITAL RIGHTS MANAGEMENT
DRM - DIGITAL RIGHTS MANAGEMENTS
EAL2+ - EVALUATION ASSURANCE LEVEL
FIDO - FAST IDENTITY ONLINE
GP- GLOBAL PLATFORM
HD - HIGH DEFINITION
HMAC - KEYED-HASH MESSAGE AUTHENTICATION CODE
HSM - HARDWARE SECURITY MODULES
IPC - INTERPROCESS COMMUNICATION
MNO - MOBILE NETWORK OPERATOR
NFC - NEAR FIELD COMMUNICATION
NVM - NON-VOLATILE MEMORY
OEM - ORIGINAL EQUIPMENT MANUFACTURERS
OMTP - OPEN MOBILE TERMINAL PLATFORM
OS - OPERATING SYSTEM
PIN - PERSONAL IDENTIFICATION NUMBER
POS - POINT OF SALE
RAM - RANDOM ACCESS MEMORY
REE - RICH EXECUTION ENVIRONMENT
ROM - READ ONLY MEMORY
RPMB - REPLAY PROTECTED MEDIA BLOCK
SE - SECURE ELEMENT
TA - TRUSTED APPLICATION
TEE - TRUSTED EXECUTION ENVIRONMENT
TPM - TRUSTED PLATFORM MODULE
UI - USER INTERFACE

UUID - UNIVERSALLY UNIQUE IDENTIFIER

WAC - WHOLESALE APPLICATIONS COMMUNITY

eMMC - EMBEDDED MULTIMEDIA CARD

1. INTRODUCTION

A Trusted Execution Environment (TEE) is a secure area of the main processor. It is isolated from the normal processing environment where the device operating system and applications run. The TEE guarantees code and data loaded inside to be protected, because the architecture provides the ability of tampering, resulting in data confidentiality and integrity. The TEE provides security features such as isolated execution, integrity of Trusted Applications along with confidentiality of their assets. In general, the TEE offers an execution space where the security is higher and efficient than the rich mobile operating system (RICH-OS) and provides more functionality than the secure element (SE).

Open Mobile Terminal Platform (OMTP) first defined the TEE in their 'Advanced Trusted Environment: OMTP TR1' standard, defining it as a "set of hardware and software components providing facilities necessary to support Applications" which had to meet the requirements of one of two defined security levels. The first security level, Profile 1, was targeted against only software attacks and whilst Profile 2, was targeted against both software and hardware attacks. Commercial TEE solutions based on ARM TrustZone technology which conformed to the TR1 standard such as Trusted Foundations, developed by Trusted Logic, were later launched. This software would become part of the Trustonic joint venture, and the basis of future GlobalPlatform TEE solutions.

Work on the OMTP standards ended in mid 2010 when the group transitioned into the **'Wholesale Applications Community' (WAC)**.The OMTP standards, including those defining a TEE, are hosted by GSMA. In July 2010 GlobalPlatform first announced their own standardization of the TEE, focusing first on the client API (the interface to the TEE within the mobile OS) which was expanded later to include the TEE internal API,a Remote Administration framework, a compliance program and standardized security level.

Today’s mobile phones provide us with many different kinds of services. For example, we can browse the web and run third-party applications; actions which can be achieved even on today’s feature phones. For this reason, mobile operating systems have grown in size and therefore are increasingly susceptible to software vulnerabilities. However, end users still expect a certain level of predictability and reliability so there is a need for additional security. This is the reason

why there is a need to use hardware assisted TEE functionality. Today almost every smartphone and tablet contains a TEE, such as ARM's TrustZone.Mobile devices equipped with TEEs have the potential to replace, amongst others, wireless tokens for opening doors in buildings or cars, traditional credit and debit cards for transactions and even report on a patient's health to a medical center.

Recent standardization efforts in GlobalPlatform could soon make it possible for TEE functionality to be accessed in a standardized manner. GlobalPlatform has announced a number of TEE specifications. The introduction of these standards have the potential to make it easier for different users to learn, practice and study this domain in a safe and controlled manner.

1.1 Tee

A TEE is a secure, integrity-protected processing environment, consisting of processing, memory and storage capabilities. Figure 1.1-1 shows how a device can be visualized as a series of distinct environments with their own set of features and services. What follows, using the terminology introduced by GlobalPlatform, describes the concepts illustrated in Figure 1.1-1.

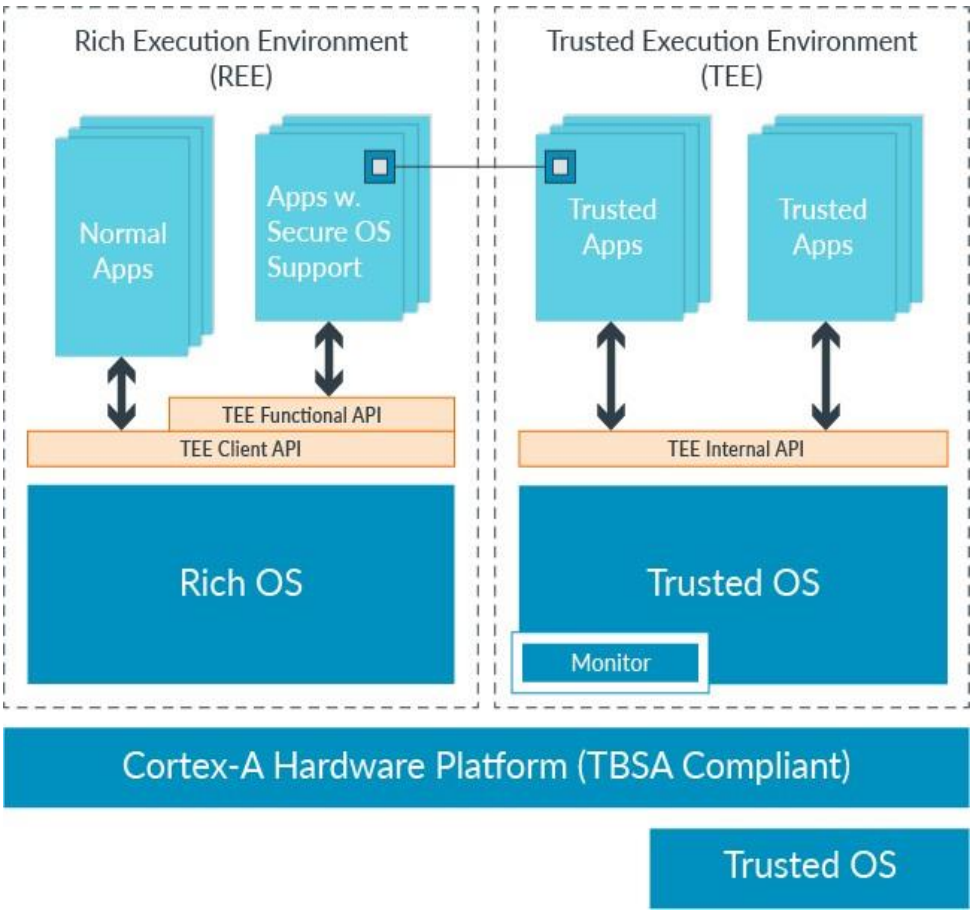

Figure 1.1 -1 TEE Archiitecture

1.1.1 Rich Execution Environment (REE)

"Rich" refers to an operating environment that is feature rich as would expect from modern platforms such as Windows, Linux, Android, iOs, etc. In this environment the majority of applications are being developed and deployed.

1.1.2 Trusted Execution Environment (TEE)

TEE is a combination of features, both software and hardware, that isolate it work done by REE. These environments have a limited set of features and services, since they are only intended to address the critical security subsystem such as unloading some cryptographic features or keys Management.

1.1.3 Trusted Application (TA)

An application encapsulating the security-critical functionality to be run within the TEE. This may be a service style application that provides a general feature, such as a generic cryptographic keystore, or it could be designed to offload a very specific part of an application that is running in the REE, such as a portion of the client state machine in a security protocol like TLS.

1.1.4 Client Application (CA)

CAs are ordinary applications (e.g. browser or e-mail client) running in the REE. CAs are responsible for providing the majority of an application's functionality but can invoke TAs to offload sensitive operations. Examining a typical TEE application workflow sequence, using GP terminology, let's consider a common use case for TEEs: the offloading of DRM protected content. The CA would be responsible for the majority of the tasks associated with viewing the content i.e. opening the media file, providing a region in the display into which it can be rendered (the window) and providing a mechanism by which to start, stop and rewind the media. A TA would be used to decrypt the protected media stream and make the decrypted content available directly to the graphics hardware that is responsible for rendering and displaying the stream.

1.2 Uses

There are a number of use cases for the TEE:

Premium Content Protection / Digital Rights Management

Note: Much TEE literature covers this topic under the definition "premium content protection" which is the preferred nomenclature of many copyright holders. Premium content protection is a specific use case of Digital Rights Management (DRM), and is controversial among some

communities. It is widely used by rights holders to manage the ways in which end users can consume content such as 4K high definition films.

The TEE is a suitable environment for protecting digitally encoded information (for example, HD films or audio) on connected devices such as smart phones tablets and HD televisions. This suitability comes from the ability of the TEE to securely store cryptographic keys, and the fact that there is often a protected hardware path between the TEE and the display and/or subsystems on devices.

The TEE is used to protect the content once it is on the device: while the content is protected during transmission or streaming by the use of encryption, the TEE protects the content once it has been decrypted on the device by ensuring that decrypted content is not exposed to the REE (that is, the Operating System).

Mobile financial services

Mobile Commerce applications such as: mobile wallets, peer-to-peer payments, contactless payments or using a mobile device as a point of sale (POS) terminal) often have well-defined security requirements. TEEs can be used, often in conjunction with near field communication (NFC), SEs and trusted backend systems to provide the security required to enable financial transactions to take place.

In some scenarios, interaction with the end user is required, and this may require the user to expose sensitive information such as a PIN, password or biometric identifier to the mobile OS as a means of authenticating the user. The TEE optionally offers a trusted user interface which can be used to construct user authentication on a mobile device.

Authentication

The TEE is well-suited for supporting biometric ID methods (facial recognition, fingerprint sensor and voice authorization), which may be easier to use and harder to steal than PINs and passwords. The authentication process is generally split into three main stages:

- Storing a reference 'template' identifier on the device for comparison with the 'image' extracted in next stage.
- Extracting an 'image' (scanning the fingerprint or capturing a voice sample, for example).
- Using a matching engine to compare the 'image' and the 'template'.

A TEE is a good area within a mobile device to house the matching engine and the associated processing required to authenticate the user. The environment is designed to protect the data and

establish a buffer against the non-secure apps located in mobile OS. This additional security may help to satisfy the security needs of service providers in addition to keeping the costs low for handset developers.
The FIDO Alliance is collaborating with GlobalPlatform to standardize the TEE for natural ID implementations.

Enterprise and government

The TEE can be used by governments and enterprises to enable the secure handling of confidential information on a mobile device. The TEE offers a level of protection against software attacks generated in the mobile OS and assists in the control of access rights. It achieves this by housing sensitive, 'trusted' applications that need to be isolated and protected from the mobile OS and any malicious malware that may be present. Through utilizing the functionality and security levels offered by the TEE, governments and enterprises can be assured that employees using their own devices are doing so in a secure and trusted manner.

Utilizing a TEE is not a silver bullet for securing a device. It provides defense in depth and helps narrow down the attack vectors that an attacker can leverage to compromise a device or user. Properly offloading tasks to TEE can efficiently protect sensitive data from leaking, however, if we assume the following scenario: a user has downloaded a new update for their device which contains malicious code.
If that malicious application can masquerade as a legitimate CA, then the attacker could have free use of the sensitive data stored in the TEE. That is they would be able to e.g. decrypt data or sign messages as a legitimate user, however, the TEE would still ensure that the key was not revealed. Even with this limitation the benefits of using a TEE far outweigh the risks of not using it and a TEE is critical to the proper functioning of certain use cases that have become commonly available in mobile devices, such as:

Keystore:
Used for storing cryptographic tokens, keys or certificates, into a TEE to make it more difficult to extract them from the device. When tokens are deployed to a TEE, a CA can make use of them through a keystore Application Programming Interface (API) and these tokens are thus not exposed to user space or Random Access Memory (RAM).

Secure storage:

This can serve as a multi-purpose facility which could allow an application to store everyday information such as user identity, pictures or documents. Most implementations provide both confidentiality and integrity protections. In fact modern storage media such as embedded MultiMediaCard (eMMC) may contain a special partition, called the Replay Protected Media Block (RPMB), to assist with integrity protection. It relies on a keyed-Hash Message Authentication Code (HMAC) for its operation and this key is protected by the TEE. Mobile phone vendors have used secure storage for calibration data and firmware configurations for many years to protect their assets and the safety of end users. For example an attacker, in this case the legitimate owner of a device, may wish to alter the modem configuration to allow them to have a greater share of the bandwidth or a higher priority on the network. This can lead to poor service for other users or be potentially harmful to the user as the new settings may pose a health risk. In order to mitigate this risk the manufacturer would store these critical configurations in secure storage and would potentially disallow the device to boot if the calibration data has been tampered with.

Secure boot

Is an extension to what has just been discussed, it provides the ability to measure the integrity of certain code and data during the boot and can be designed to disallow boot if any of the components have been altered. It does this by storing a list of known good configurations, i.e. the signatures of firmware and software components and comparing these to the components as they are prepared for loading. Modern implementations of secure boot extend from the hardware all the way to the user space. Non-Volatile Memory (NVM) such as the Read Only Memory (ROM) code or key hashes stored in physical write once fuses can be used to provide the basis of security. The systems can be thus designed that each verified component can be relied upon to provide the verification of the layer(s) above it. A simplified example would be that the ROM code verifies the bootloader, which in turn verifies the main OS, which in turn verifies the overall REE before launching the first user space application e.g. init. Secure boot does not

guarantee that the device is free of security issues; rather it can certify that the components that have booted are the best known configuration as provided by a trusted source, e.g. the OEM.

Digital Rights Managements (DRM):
One of the driving forces behind the wider adoption of TEEs has been the media industry. DRM content is becoming ubiquitous, common examples are music files from iTunes or a video stream from NETFLIX. In order to protect the media stream from piracy the data is encrypted with a key that is generally device or session specific. The TEE is used to protect this key, perform the decryption of the session and make the data available to other parts of the system so it can be securely displayed.
Although the previous use cases are more prevalent on mobile devices they are now seeing widespread deployment on a variety of devices ranging from desktops to smart watches.

1.3 Benefits of using a TEE

From a business and commercial perspective, the TEE meets the requirements of all of the key players. At a high level:

- **Mobile manufacturers'** security concerns are tied to several factors, not the least of which being the sheer number of stakeholders involved in device and application delivery. A framework (such as GlobalPlatform-certified TEE) that guarantees a minimum baseline for platform security would allow all stakeholders to make updates to devices and applications while minimizing threats to consumers.
- **For MNOs** the TEE delivers a higher level of security than what the Rich OS offers and higher performance than what a Secure Element (SE) typically offers. In essence, the TEE ensures a high level of trust between the device, the network, the edge and the cloud, thereby improving the ability of a MNO to enhance services for root detection, SIM-lock, anti-tethering, mobile wallet, mobile as PoS, data protection, mobile device management, application security, content protection, device wipes, and anti-malware protection.
- **Content and service providers** want the TEE to ensure that their product remains secure and can be deployed to numerous platforms in a common manner and is easily accessible to the end user.

- **Payment service providers** do not want to have to develop different versions of the same application to satisfy the needs of different proprietary TEE environments. E.g. if the ecosystem is not standardized, payment service providers will have to be certified and support different applications and processes. This is time consuming, costly and counterintuitive to the goal of creating a mass market for application deployment. Focusing specifically on security, the TEE is a unique environment that is capable of increasing the security and assurance level of services and applications, in the following ways:
- **User Authentication:** Using the trusted UI, the TEE makes it possible to securely collect a user's password or PIN. This trusted user authentication can be used to verify a cardholder for payment, confirm a user's identification to a corporate server, attest to a user's rights with a content server, and more.
- **Trusted Processing and Isolation:** Application processing can be isolated from software attacks by running in the TEE. Examples include processing a payment, decrypting premium content, reviewing corporate data, and more.
- **Transaction Validation:** Using the trusted UI, the TEE ensures that the information displayed on-screen is accurate. This is useful for a variety of functions, including payment validation or protection of a corporate document.
- **Usage of Secure Resources:** By using the TEE APIs, application developers can easily make use of the complex security functions made available by a device's hardware, instead of using less safe software functions. This includes hardware cryptography accelerators, SEs, biometric equipment and the secure clock.
- **Certification:** Trusted certification is best achieved through standardization of the TEE, which in turn improves stakeholder confidence that the security-dependent applications are running on a trusted platform.

2. STANDARDIZATION

Smart connected devices, such as smartphones, are intrinsic to daily life: they are used for business, social interactions, making purchases and enjoying media content. All of this data, however, is susceptible to attacks from hackers and the millions of downloadable applications represent an even larger opportunity for fraudsters.

Similarly, automotive and home devices are increasingly becoming connected and offering more functionality. On top of this, consumers are increasingly using their devices in new ways: organizing a trip from a smart TV, streaming music while driving or using a smartphone to pay for shopping. These expanded practices create new security vulnerabilities, which highlight the need for mechanisms that allow trusted parties to have access to applications without granting hackers the same opportunity.

Service providers and original equipment manufacturers (OEMs) now need to protect applications on many levels: from attacks originating in a device's operating system, authenticating the correct user to the correct service, offering increased privacy, protecting valuable content, allowing secure access to corporate and personal data and mitigating financial risks. One solution to these security challenges is to provide a small, isolated execution environment that allows service providers and OEMs to improve the user experience while reducing fraud. The GlobalPlatform Trusted Execution Environment (TEE) effectively addresses these concerns.

2.1 Global Platform

While there are many proprietary systems, GlobalPlatform is working to standardize the TEE. Standardizing the TEE is helpful for implementers of mobile wallets, NFC payment implementations, premium content protection and bring your own device (BYOD) initiatives. These following TEE specifications are currently available from the GlobalPlatform website:

- TEE Client API Specification v1.0 outlines the communication between applications running in a mobile OS and trusted applications residing in the TEE.

- TEE Systems Architecture v1.0 explains the hardware and software architectures behind the TEE.
- TEE Internal API Specification v1.0 specifies how to develop trusted applications.
- TEE Secure Element API Specification v1.0 specifies the syntax and semantics of the TEE Secure Element API. It is suitable for software developers implementing trusted applications running inside the TEE which need to expose an externally visible interface to client applications.
- Trusted User Interface API Specification v1.0 specifies how a trusted UI should facilitate information that will be securely configured by the end user and securely controlled by the TEE.
- TEE TA Debug Specification v1.0 specifies the GlobalPlatform TEE debug interfaces and protocols.
- TEE Management Framework v1.0 specifies the GlobalPlatform Remote Administration Framework, which enables trusted applications on a device to be remotely managed by trusted service providers.

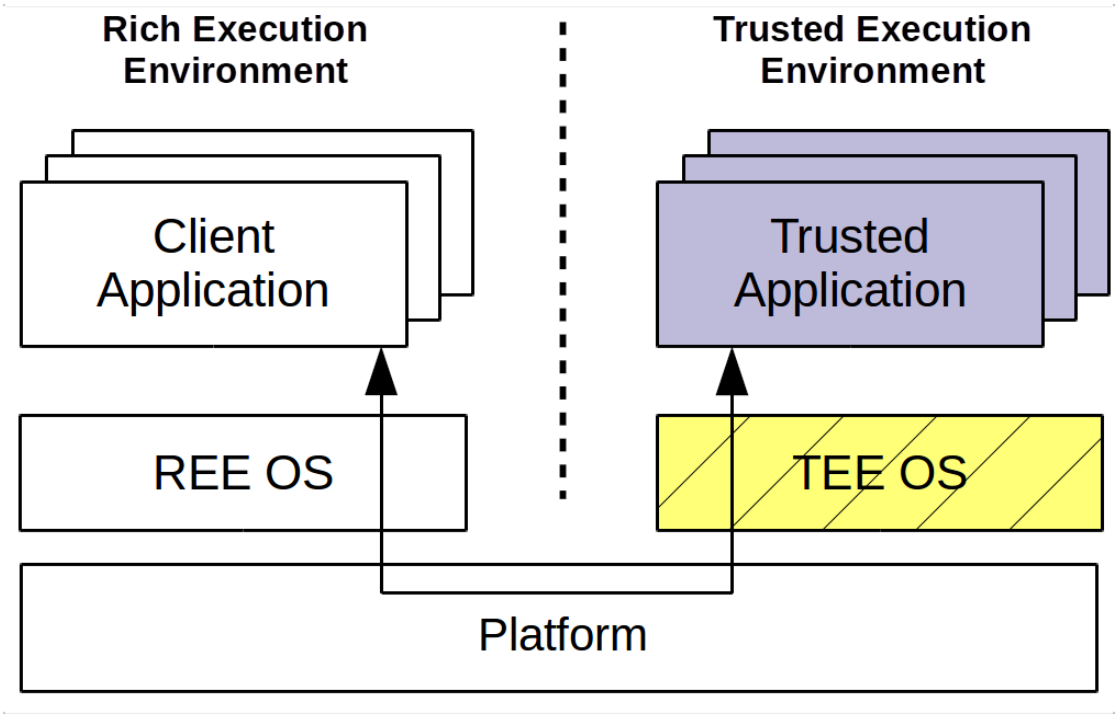

Figure 2.1-2 Computing Device TEE

Trustonic was the company to qualify a GlobalPlatform-compliant TEE product. Since then, a significant number of GlobalPlatform TEE implementations have become available. A list of those which have been formally qualified by GlobalPlatform can be found at, and many other TEE products offer a high level of compatibility with GlobalPlatform standards.

The resulting lack of standardization has presented application developers with a significant challenge to overcome; each proprietary TEE solution requires a different version of the same application to ensure that the application conforms to unique versions of the technology. In addition, if the application provider wishes to deploy to multiple TEE solution environments and have assurance that each environment will provide a common level of security, then a security evaluation will need to be performed on each TEE solution. This leads to a resource intensive development process.

There are two central reasons why the TEE exists:

- **An increasing number of mobile services, which require a greater level of security, are emerging.**
- **With a growing number of users, there is a greater need for protection against software attacks**. Applications with higher security requirements, and therefore heightened ramifications if compromised, require more protection than can be offered by rich OS solutions alone.

Enterprise IT environments, delivery of premium multimedia content, mobile payments, the Internet of Things, government identification programs and more seek to balance a consumer's desire for a rich experience with the security concerns shared by consumers and service providers. The TEE isolates trusted applications and keeps them away from any malware which might be downloaded inadvertently. Because of this, the TEE will become an essential environment within all devices as the secure services market evolves.

Since GlobalPlatform is handset and Rich OS agnostic, it is well placed to bring forward specifications for the TEE that can be embraced by all suppliers and reside comfortably alongside each of their rich OS environments. Interoperability in both functionality and security will be enhanced by the standardization of the TEE. This will simplify application development and deployment for all concerned, saving costs and time to market.

The GlobalPlatform TEE Protection Profile 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 will have to 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 Common Criteria portal has officially listed the GlobalPlatform TEE Protection Profile on its website, under the Trusted Computing category. This important milestone means that industries using TEE technology to deliver services such as premium content and mobile wallets, or enterprises and governments establishing secure mobility solutions, can now formally request that TEE products are certified against this security framework.

GlobalPlatform is committed to ensuring a standardized level of security for embedded applications on secure chip technology. It has developed an open and thoroughly evaluated trusted execution environment (TEE) ecosystem with accredited laboratories and evaluated products. This certification scheme created to certify a TEE product in 3 months has been launched officially in June 2015.

2.2 Importance of Standardization

TEE standardization is essential to avoid fragmentation. The proliferation of proprietary TEE solutions would lead to the following:

- Higher costs to develop or change applications/solutions when creating or adapting to proprietary platforms
- The need for very specialized skills
- Extended time-to-market due to longer development times and potential integration issues.

Standardization, by contrast, enables simplified and unified implementation and improves interoperability between stakeholders. Furthermore, standardization allows a large ecosystem to thrive and blossom, allowing for multiple business partners and, because it ensures long-term stability and survivability, protects investment in a way that proprietary solutions cannot. It also defines a basis for evaluating and comparing different solutions. Lastly, standardization creates a foundation for a uniform certification process.

3. OPEN-TEE

Open-TEE is an open source project whose main goal is to implement a virtual TEE compliant with the latest GP TEE specifications. The Open-TEE framework tools removing the need for specialized hardware, the overheads that it incurs and the elimination of cost for purchasing hardware TEEs. Furthermore, provides to allow developers access a TEE, even a virtual, that so far is restricted from chip and device manufacturers.
The most common debugging Trusted applications methods are either expensive or to resort to primitive print tracing by inserting diagnostic output in the source code, that provides detailed instruction level of debugging, but the costs associated with these debuggers can be prohibitively expensive and complex. Another problem encountered by the TEE developers is that if a TA, running on actual device hardware, crashes then a hard reset of the device may be required. Thereby the time and effort of the debugging process is significantly increased.
Safely exposing TEE functionality to application developers provides them with a means to improve the security and privacy of their applications. Also exposing TEE technology to a wider audience does not guarantee that all security threats will disappear, however, a large community can provide plentiful ideas and a more in-depth research, than a small group of people. The financial and technical aspects of accessing a hardware TEE make it infeasible, for this reason the creation of a TEE framework such as Open-TEE, that is not bound to any hardware or any particular vendor, yet tries to conform to one standardization effort (for which GP has been chosen) is a great improvement.

3.1 Architecture

The following section describes the design and implementation of such a software framework which is called Open-TEE. It will begin with an overview of the structure of the Open-TEE environment. Figure 3.1 identifies the main components and their relationships. The color code used in Figure 3.1 is the same as that used for Figure 2.3 to make the correspondence between

the Open-TEE implementation architecture and the GP conceptual architecture is clear. Each component is described in detail below.

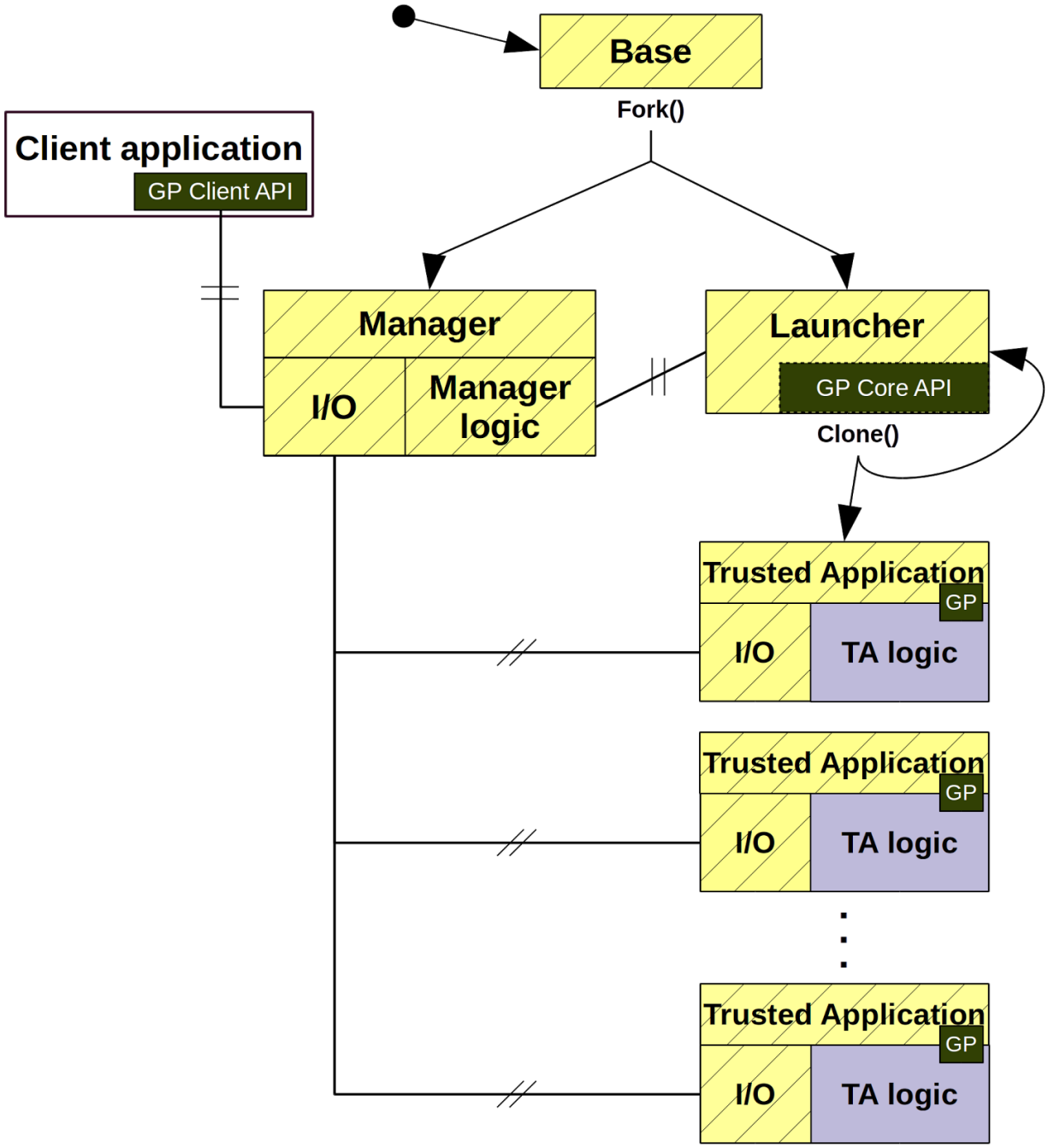

Figure 3.1-3 Open-TEE architecture

3.1.1 Base

Open-TEE is designed to function as a daemon process in user space. It starts executing Base, a process that encapsulates the TEE functionality. Base is responsible for loading the configuration and preparing the common parts of the system. Once initialized Base will fork and create two independent but related processes. One process becomes Manager and the other, Launcher which serves as a prototype for TAs.

3.1.2 Manager

Manager can be visualized as Open-TEE's "operating system". Its main responsibilities are: managing connections between applications, monitoring TA state, providing secure storage for a TA and controlling shared memory regions for the connected applications. Centralizing this functionality into a control process can also be seen as a wrapper abstracting the running environment (e.g. GNU/Linux) and reconciling it with the requirements imposed by the GP TEE standards. GP requirements and the host environment's functionality are not always aligned. For example, GP requirements stipulate that if a TA/CA process crashes unexpectedly, all shared resources of the connected processes must be released. In a typical running environment, this requires additional steps beyond just terminating the process. For example, all shared memory must be unregistered – this needs to be a distinct action from normal process termination.

3.1.3 Launcher

The sole purpose of Launcher is to create new TA processes efficiently. When it is first created, Launcher will load a shared library implementing the GP TEE Core API and will wait for further commands from Manager. Manager will signal Launcher when there is a need to launch a new TA (for example, when there is a request from a CA). Upon receiving the signal, Launcher will clone itself. The clone will then load the shared library corresponding to the requested TA. The design of Launcher follows the "zygote" design pattern (such as that used in Android) of preloading common components. This is intended to improve the perceived performance of

starting a new TA in Open-TEE: because shared libraries and configurations common to all TAs are preloaded into Launcher, the time required to start and configure the new process is minimal. A newly created TA process is then re-parented onto Manager so that it is possible for it to control the TA (so that, for example, it can enforce the type of GP requirements discussed in the paragraph above).

3.1.4 TA Processes

The architecture of the TA processes is inspired by the multi-process architecture utilized in the Chromium Project. Each process has been divided into two threads. The first handles InterProcess Communication (IPC) and the second is the working thread, referred to respectively as the I/O and TA Logic threads. This architectural model enables the process to be interrupted without halting it, as occurs when changing status flags and adding new tasks to the task queue. Additional The architecture of Manager follows the same division benefits of this model are that it allows greater separation and abstraction of the TA functionality from the Open-TEE framework.

3.1.5 GP TEE APIs

The GP TEE Client API and GP TEE Core API are implemented as shared libraries in order to reduce code and memory consumption. In addition, loading the GP TEE Core API into Launcher when it is created will help to reduce the startup times of the TA process removing the need to load it for every TA; this is one of the key benefits of the “zygote” design.

3.1.6 IPC

Open-TEE implements a communication protocol on top of Unix domain sockets and interprocess signals as the means to both control the system and transfer the messages between the CA and TA.

4. GLOBAL PLATFORM: CLIENT API SPECIFICATION

There are two main API's for developing TEE applications. The first one is the Client API that describes all the functions and interfaces that a CA can use to interact with the TEE engine and its corresponding TA. The second one is the TEE Internal Core API Specification that describes all the standards regarding the TA. On this section and considering the scope of this paper, the Client Application API specification and all its components are going to be described. That is to say there is going to be a demonstration of all the Client Application functionality via presenting all the available methods, functions and so on, as they are described in the Client API specification document of Global Platform.
All the following are quoted from the Global Platform Specification documents.

Client API Specification

4.1.1 DATA TYPES

TEEC_UUID	This type contains a Universally Unique Resource Identifier (UUID) type as defined in RFC4122. These UUID values are used to identify Trusted Applications.	typedef struct { uint32_t timeLow; uint16_t timeMid; uint16_t timeHiAndVersion; uint8_t clockSeqAndNode[8]; } TEEC_UUID;
TEEC_Result	This type is used to contain return codes which are the results of invoking TEE Client API functions.	typedef uint32_t TEEC_Result;

TEEC_Context	This type denotes a TEE Context, the main logical container linking a Client	typedef struct { Imp;
	Application with a particular TEE. Its content is entirely implementationdefined.	} TEEC_Context;
TEEC_Session	This type denotes a TEE Session, the logical container linking a Client Application with a particular Trusted Application. Its content is entirely implementation-defined.	typedef struct { imp; } TEEC_Session;
TEEC_SharedMemory	This type denotes a Shared Memory block which has either been registered with the implementation or allocated by it.	typedef struct { void* buffer; size_t size; uint32_t flags; imp; } TEEC_SharedMemory;
TEEC_MemoryReference	This type defines a memory reference that uses a pre-registered or pre-allocated Shared Memory block. It is used as a TEEC_Operation parameter when the corresponding parameter type is one of TEEC_MEMREF_WHOLE, TEEC_MEMREF_PARTIAL_INPUT, TEEC_MEMREF_PARTIAL_OUTPUT, or TEEC_MEMREF_PARTIAL_INOUT.	typedef struct { TEEC_SharedMemory* parent; size_t size; size_t offset; } TEEC_MemoryReference;

TEEC_Parameter ;	This type defines a Parameter of an operation. It is either a temporary memory reference or an indirect memory reference.	typedef union { TEEC_TempMemoryReference tmpref; TEEC_MemoryReference memref; } TEEC_Parameter
TEEC_Operation	This type defines the payload of either an open Session operation or an invoke Command operation. It is also used for cancellation of operations, which may be desirable even if no payload is passed.	typedef struct { uint32_t started; uint32_t paramTypes; TEEC_Parameter params[4]; imp; } TEEC_Operation;

4.1.2 CONSTANTS

4.1.2.1 TEEC_InitializeContext

TEEC_Result TEEC_InitializeContext(
const char name, TEEC_Context*
context)

This function initializes a new TEE Context, forming a connection between this Client Application and the TEE identified by the string identifier name. The Client Application MAY pass name with a value of NULL, which means that the Implementation MUST select a default TEE to connect to. The supported name strings, the mapping of these names to a specific TEE, and the nature of the default TEE are implementation-defined. The caller MUST pass a pointer to a valid TEEC Context in context.

4.1.2.2 TEEC_FinalizeContext

```
void TEEC_FinalizeContext(
 TEEC_Context* context)
```

This function destroys an initialized TEE Context, closing the connection between the Client Application and the TEE. The Client Application MUST only call this function when all Sessions inside this TEE Context have been closed and all Shared Memory blocks have been released. The implementation of this function MUST NOT be able to fail; after this function returns the Client Application must be able to consider that the Context has been closed. The function implementation MUST do nothing if the value of the context pointer is NULL.

4.1.2.3 TEEC_OpenSession

```
TEEC_Result TEEC_OpenSession (
TEEC_Context* context,
TEEC_Session* session,  const
TEEC_UUID* destination,
uint32_t connectionMethod,
const void* connectionData,
TEEC_Operation* operation,
uint32_t* returnOrigin)
```

This function opens a new Session between the Client Application and the specified Trusted Application. The Implementation MUST assume that all fields of this session structure are in an undefined state. When this function returns TEEC_SUCCESS the Implementation MUST have populated this structure with any information necessary for subsequent operations within the Session. The target Trusted Application is identified by a UUID passed in the parameter destination.

4.1.2.4 TEEC_CloseSession

```
void TEEC_CloseSession (
TEEC_Session* session)
```

This Function closes a Session which has been opened with a Trusted Application. All Commands within the Session MUST have completed before this function can be called. The Implementation MUST do nothing if the input session parameter is NULL. The implementation of this function MUST NOT be able to fail; after this function returns the Client Application must be able to consider that the Session has been closed.

4.1.2.5 TEEC_RegisterSharedMemory

```
TEEC_Result TEEC_RegisterSharedMemory(
 TEEC_Context* context,
 TEEC_SharedMemory* sharedMem)
```

This function registers a block of existing Client Application memory as a block of Shared Memory within the scope of the specified TEE Context, in accordance with the parameters which have been set by the Client Application inside the sharedMem structure. The input context MUST point to an initialized TEE Context.

4.1.2.6 TEEC_InvokeCommand

```
TEEC_Result TEEC_InvokeCommand(
TEEC_Session* session, uint32_t commandID,
TEEC_Operation* operation,  uint32_t*
returnOrigin)
```

This function invokes a Command within the specified Session. The parameter session MUST point to a valid open Session. The parameter commandID is an identifier that is used to indicate which of the exposed Trusted Application functions should be invoked. The supported command identifier values are defined by the Trusted Application‟s protocol.

4.1.2.7 TEEC_ReleaseShared Memory

void TEEC_ReleaseSharedMemory (TEEC_SharedMemory sharedMem)*

This function deregisters or deallocates a previously initialized block of Shared Memory. For a memory buffer allocated using TEEC_AllocateSharedMemory the Implementation MUST free the underlying memory and the Client Application MUST NOT access this region after this function has been called. In this case the Implementation MUST set the buffer and size fields of the sharedMem structure to NULL and 0 respectively before returning. For memory registered using TEEC_RegisterSharedMemory the implementation MUST deregister the underlying memory from the TEE, but the memory region will stay available to the Client Application for other purposes as the memory is owned by it. The implementation MUST do nothing if the value of the sharedMem parameter is NULL.

4.1.3 TA Interface & Effect of Client Operation on TA Interface

Each Trusted Application exposes an interface (the TA interface) composed of a set of entry point functions that the Trusted Core Framework implementation calls to inform the TA about life-cycle changes and to relay communication between Clients and the TA. Once the Trusted Core Framework has called one of the TA entry points, the TA can make use of the TEE Internal Core API to access the facilities of the Trusted OS. Each Trusted Application is identified by a Universally Unique Identifier (UUID) as specified in [RFC 4122]. Each Trusted Application also comes with a set of Trusted Application Configuration Properties. These properties are used to configure the Trusted OS facilities exposed to the Trusted Application. Properties can also be used by the Trusted Application itself as a means of configuration.
The following table presents the TA functions that are called, whenever one of the earlier presented functionalities of the CA are executed:

Client Operation	Trusted Application Effect
TEEC_OpenSession	If a new Trusted Application instance is needed to handle the session, TA_CreateEntryPoint is called. Then, TA_OpenSessionEntryPoint is called
TEEC_InvokeCommand	TA_InvokeCommandEntryPoint is called.
TEEC_CloseSession	TA_CloseSessionEntryPoint is called. For a multi-instance TA or for a single-instance, non keep-alive TA, if the session closed was the last session on the instance, then TA_DestroyEntryPoint is called. Otherwise, the instance is kept until the TEE shuts down
TEEC_RequestCancellation	Cancellation process is triggered

5. IMPLEMENTATION: Password Manager

5.1 High level description

The application is the implementation of a password manager application in the context of TEE, where the user's passwords are stored in the secure trusted application's storage. In its current form it is a command line program running under a linux-based operating system, using the open-tee engine.

When the application starts the user is greeted with the logo and the master password is requested so as to get access to the main program functionality:

Figure 5.1-4 Application Login Screen

When the password is given it is hashed and compared to the hash of the actual password that is stored in the Trusted Application's storage.

After a successful authentication the user is presented with the application's main menu:

Figure 5.1-5 Application main menu

It is noted that the application exits after a failed authentication attempt.

The five available options of the main menu as seen above, are the following:

1)Create /Overwrite a password:

This allows the user to create a new unique password entry in the Trusted Storage. Each entry consists of 2 fields, password id and the actual password. Whenever a user creates a password he is asked to provide both id and the actual password. It is also important to remember the id of the password to be able to retrieve it later on.

Figure 5.1-6 Application Password Retrieving

2)Show a password:

With this option the user has the ability to retrieve the password from the Trusted environment, giving the password's unique id.

3)Delete a password:

This option deletes the password entry for the given password id.

4)Change master password:

This option is used to change the master password that is used to lock the application. It is important to note that the user has to provide the old master password to be able to change it.

5)Exit Secure Password Manager:

Exits the program

As mentioned earlier, the application uses the trusted execution environment secure storage to save the user's passwords. It does so by sending the data over the channel that is setup upon applications' start, that is nothing more than a part of temporarily shared memory between the two applications.

A master password is used for accessing the application. This password is also stored in the TA memory and is hashed using SHA256 algorithm along with salting.

The application uses an attestation technique, yet not fully developed to be optimal.

5.2 Architecture

The application consists of two components based on the TEE architecture described earlier in this document, the CA (client application) and the TA (trusted application).

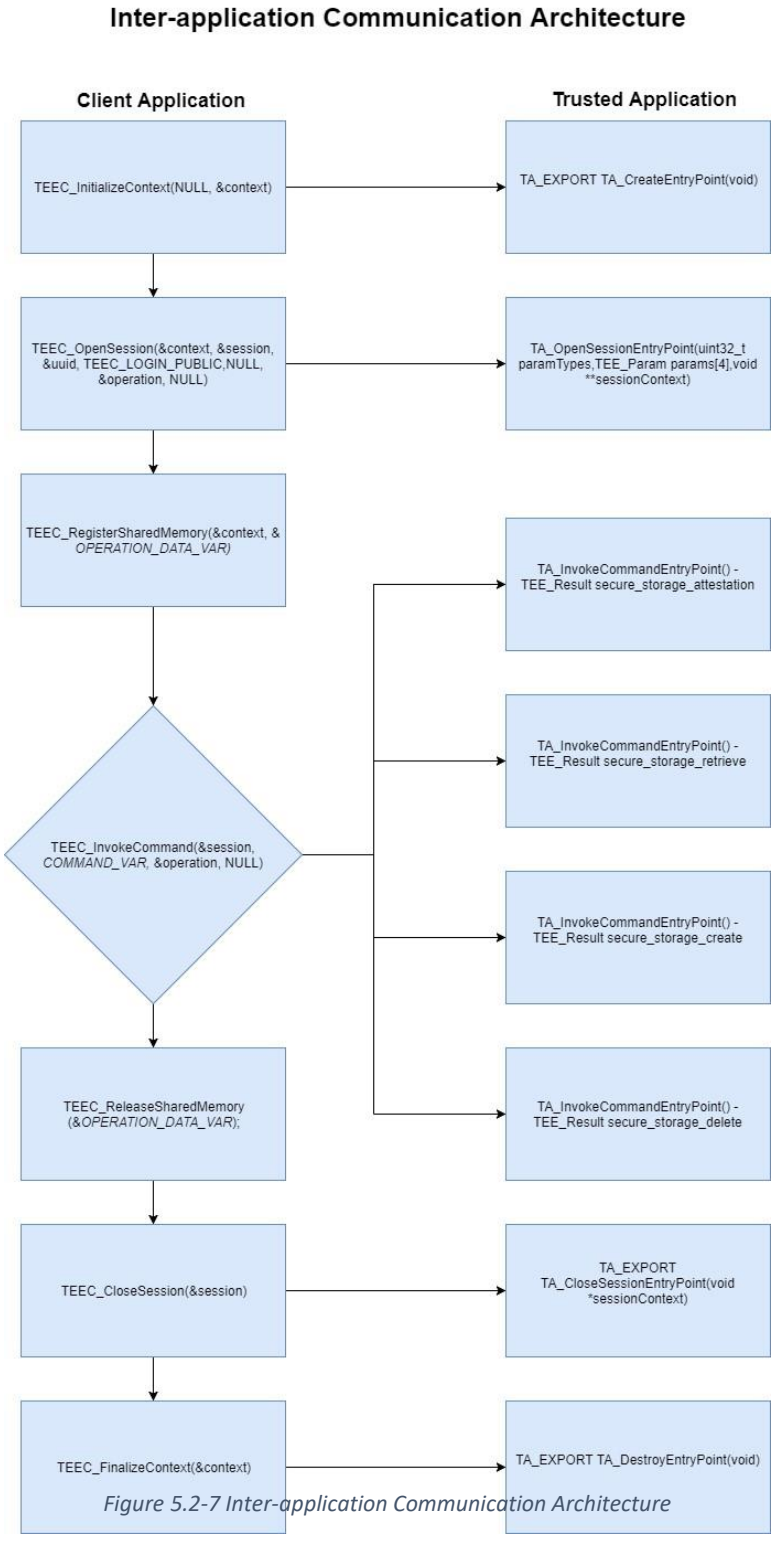

Figure 5.2-7 Inter-application Communication Architecture

5.2.1 Client Application Architecture

The client application includes a struct named TEEC_UUID that contains the unique id of the application that needs to be included in the Trusted application too. This is used by the tee engine to identify the correlation between trusted and client applications and which CAs should have access to communicate with the equivalent TAs.

```
static const TEEC_UUID uuid = {
    0x12345678, 0x8765, 0x4321, {'P','A','S','S','L','O','C','K'}
};
The CA invokes commands  from the TA and for this purpose the available commands are defined as
constants at the start of the CA.
//commands
#define CREATE   0x00000001
#define RETRIEVE 0x00000002
#define DELETE     0x00000003
#define ATTESTATION 0x00000004
In the main function of the program multiple objects are being initialized:
TEEC_Result tee_rv;
    TEEC_Context context;
    TEEC_Session session;
    TEEC_Operation operation;
    TEEC_SharedMemory object_mem;
    TEEC_SharedMemory object_id_mem;
TEEC_SharedMemory output_mem;
memset((void *)&object_mem, 0, sizeof(object_mem));
memset((void *)&object_id_mem, 0, sizeof(object_id_mem));
memset((void *)&output_mem, 0, sizeof(output_mem));
memset((void *)&operation, 0, sizeof(operation));
memset(object,0, DATA_SIZE);      memset(objectid, 0,
DATA_SIZE);      memset(output, 0, DATA_SIZE);
```

First of all InitialiazeContext function is invoked to start the communication between the CA and the Open-TEE engine with the variable context (of type TEEC_Context) as parameter:

```
  tee_rv = TEEC_InitializeContext(NULL, &context);
if (tee_rv != TEEC_SUCCESS) {
      printf("TEEC_InitializeContext failed: 0x%x\n", tee_rv);
goto end_1;
```

The next step is to establish the communication between the CA and the TA that is achieved via the UUID that is passed as a parameter to the TEEC_OpenSession function along with session and context variables.

```
   tee_rv = TEEC_OpenSession(&context, &session, &uuid, TEEC_LOGIN_PUBLIC,NULL,
&operation, NULL);     if (tee_rv != TEEC_SUCCESS) {
printf("TEEC_OpenSession failed: 0x%x\n", tee_rv);
goto end_2;
   }
```

To be able to send data to the TA the CA initializes structs of type TEEC_SharedMemory and then the TEEC_RegisterSharedMemory() function is called to assign the data to the shared memory block. This can be illustrated in the following example code.

```
 BYTE objectid[DATA_SIZE];  memset((void
*)&object_id_mem, 0, sizeof(object_id_mem));
memset(objectid, 0, DATA_SIZE); object_id_mem.buffer =
objectid;  object_id_mem.size = DATA_SIZE;
object_id_mem.flags = TEEC_MEM_INPUT;
   tee_rv = TEEC_RegisterSharedMemory(&context, &object_id_mem);
if (tee_rv != TEE_SUCCESS) {


      printf("Failed to register DATA shared memory\n");
goto end_3;
```

```
}
```

After the share memory registration the program confirms its authenticity to the TA with the attestation feature that is developed in the CA that sends some data hashed into a sha256 hash and sends it to the TA. The TA application has stored the valid hash and compares it to the one that is sent to the CA. If the two hashes do not match then the TA shuts down the connection immediately.
After the attestation check is succedfull the user is prompted to provide the master password. Once again, the CA hashes the password with sha256 algorithm and then sends it to the TA for validation.

After a successful login, the program flow continues to the main menu. The functionality is explained earlier in this paper. However, it is worth noting that each option of the program is correlated to a specific "operation" that is called to be performed between the CA and the TA. Each operation takes as input four parameters, that describe the type of the data that are going to be sent to the TA.
operation.paramTypes = TEEC_PARAM_TYPES
(TEEC_MEMREF_WHOLE,TEEC_MEMREF_WHOLE,TEEC_NONE, TEEC_NONE);

After initiating "paramTypes" the actual parameters are given to the operation struct. Those parameters are the registered shared memory items initiated earlier. To call the correlated command of the TA, the CA uses the TEEC_InvokeCommand function with four parameters:

- The session
- The name of the command
- The populated operation struct (contains data regarding the invoked command)
- And uint32_t* returnOrigin

The following example code demonstrates the aforesaid functionality. More specifically this code section calls the authentication command for the master password of the TA. Afterwards if the authentication fails the user is informed with the appropriate message.

```
if (tee_rv != TEEC_SUCCESS) {
            if (tee_rv==TEEC_ERROR_GENERIC){
printf("Object/Password does not exist!\n");                    sleep(2);
               memset(objectid, 0, DATA_SIZE);
memset(output, 0, DATA_SIZE);
               memset((void *)&operation, 0, sizeof(operation));
            }
else{
               printf("TEEC_InvokeCommand failed: 0x%x\n", tee_rv);
goto end_6;
            }
```

Each command is initiated and invoked in the same way that is mentioned above, e.g. the operation struct is populated and given as a parameter to the TEEC_InvokeCommand along with the appropriate command name and the session parameter.

At the end of the CA it is important to finalize all components in the right order. That is to say the registered shared memory should be first released, then the session between the applications should be terminated and finally the context should be shutdown to close the connection to the tee engine.

```
end_6:
  printf("Exiting TEE..\n");
exitflag=1;
  TEEC_ReleaseSharedMemory(&output_mem); end_5:
  if (exitflag==0){
     printf("Exiting TEE..\n");
exitflag=1;
  }
  TEEC_ReleaseSharedMemory(&object_mem); end_4:
```

```
  if (exitflag==0){
     printf("Exiting TEE..\n");
exitflag=1;
  }
  TEEC_ReleaseSharedMemory(&object_id_mem); end_3:
  if (exitflag==0){
     printf("Exiting TEE..\n");
exitflag=1;w
  }
  TEEC_CloseSession(&session); end_2:
  if (exitflag==0){
printf("Exiting TEE..\n");
exitflag=1;
  }
   TEEC_FinalizeContext(&context); end_1:
   if (exitflag==0){
printf("Exiting TEE..\n");
exitflag=1;
   }
exit(tee_rv);
}
```

6. CONCLUSION

This thesis deals with an application that was built on top of Open-TEE framework. It is a password manager that stores the passwords of the user in a TEE. Thus, the way it stores the passwords, is more secure than a typical password manager. Open-TEE framework is compliant with GP specifications, so the application can be integrated and ported to other GP compliant solutions, even to a hardware TEE.

Open-TEE is an open source project, its creation helps the developers implement solutions in the context of TEE, without the need of costly hardware components that most of them have not access to. This solution also provides a means for testing and debugging of applications and systems built over TEE, in direct way that is also free of cost. As a result, the security and usability of those applications and services can be increased when developed with Open-TEE.

From our experience using Open-TEE framework, we can conclude in following:

- Although coding in Open-Tee is not an easy task, the framework provides tools that make it easier than using hardware based or other solutions.
- Building the framework has limitations and prerequisites, which requires a lot of time on installation, debugging and testing to make it reliable and stable.
- The debugging process, Open-TEE provides, is straight forward and simple to use.
- Little knowledge of Hardware programming needed to develop a complex application on top of TEE

This application is a strong a secure password manager at this point, however, it can be enhanced with more features and improvements in the future, providing more usability and security.

7. BIBLIOGRAPHY

1. https://www.gsma.com/digitalcommerce/mobile-security-technology-overview-trustedhttps://www.gsma.com/digitalcommerce/mobile-security-technology-overview-trusted-execution-environment-teeexecution-environment-tee
2. https://www.embedded.com/design/connectivity/4430486/Trusted-executionhttps://www.embedded.com/design/connectivity/4430486/Trusted-execution-environments-on-mobile-devicesenvironments-on-mobile-devices
3. http://ieeexplore.ieee.org/document/6799152/?reload=true
4. https://www.researchgate.net/publication/262412456_Trusted_execution_environments_on_mobile_devices
5. GlobalPlatform, " TEE System Architecture," http://www.globalplatform.org/ specificationsdevice.asp.
6. GlobalPlatform, "Device specifications for trusted execution environment." http://www.globalplatform.org/specificationsdevice.asp.
7. The Chromium Projects, "Multi-process architecture," http://www.chromium.org/developers/design-documents/multi-process-architecture. Muthu, "Emulating trust zone feature in android emulator by extending qemu," Master's thesis, KTH Royal Institute of Technology, 2013.
8. https://www.electronicsweekly.com/blogs/eyes-on-android/what-is/what-is-a-trustedexecution-environment-tee-2015-07/
9. The Apache Software Foundation, "Apache license, version 2.0," http://www. apache.org/licenses/LICENSE-2.0.
10. https://arxiv.org/pdf/1506.07367.pdf
11. https://arxiv.org/pdf/1506.07739.pdf
12. https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/23823/Dettenborn.pdf?sequence=1
13. https://www.cs.helsinki.fi/group/secures/CCS-tutorial/tutorial-proposal.pdf
14. http://www.ibm.com/security/cryptocards/
15. J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: an execution infrastructure for TCB minimization," in Proceedings of the 2008 EuroSys Conference, Glasgow, Scotland, UK, April 1-4, 2008, 2008, pp.315–328.

[Online]. Available: http://doi.acm.org/10.1145/1352592.1352625

16. Linaro, " OP-TEE,"https://wiki.linaro.org/WorkingGroups/Security/ OP-TEE.