TÉCNICO LISBOA

SmartZone: Enhancing the security of TrustZone with SmartCards

Afonso Maria Viegas Bota Carreira Caetano

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor(s): Prof. Ricardo Jorge Fernandes Chaves
Prof. Miguel Nuno Dias Alves Pupo Correia

Examination Committee

Chairperson: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Supervisor: Prof. Ricardo Jorge Fernandes Chaves
Member of the Committee: Prof. Miguel Ângelo Marques de Matos

September 2021

Acknowledgments

First of all, I would like to thank my mother for accompanying me throughout the course, but mainly in this most challenging phase that was the realization of this thesis. Thank you for always encouraging me in all my adventures and for always supporting my ideas.

I would like to express my gratitude to my supervisors, Professor Ricardo Chaves and Professor Miguel Pupo Correia, for the support, guidance, and help provided along this entire journey. Their knowledge and experience provided me the right direction to achieve this work. Their different views allowed interesting and important discussions which led this work to a better direction.

Additionally, I would like to thank all my friends for turning the college years into an amazing experience. A huge part of my journey in Instituto Superior Técnico (IST) were the colleagues that have done this journey with me, that I'm lucky enough to assure that some of them became great friends, and that they are coming with me for life. I also want to thank my high school friends who never hesitate to help me and are always and always present in my adventures and life experiences, without them nothing would be possible.

To each and every one of you – Thank you.

Resumo

Os serviços atuais de armazenamento na nuvem têm segurança limitada e exigem que os utilizadores concedam privilégios de acesso total a contas individuais de armazenamento na nuvem. Com os avanços recentes, a necessidade de dispositivos móveis que possam gerir identidades e fornecer autorizações de acesso convenientes de maneira confiável e segura resulta numa necessidade e preocupação cada vez maiores. Para isso, várias soluções foram propostas e projetadas pela indústria. Em relação aos processadores de uso geral, o ARM TrustZone fornece uma plataforma e interação com o usuário muito mais confiável do que as aplicações de "mundo normal". Por outro lado, também existem sistemas restritos dedicados, como *smartcards* (SC), que são considerados muito seguros devido ao seu isolamento e especificações.

O trabalho proposto consiste na criação de um serviço agregador de armazenamento na nuvem altamente seguro, juntando as funcionalidades do ARM TrustZone (agora presente na maioria dos dispositivos móveis), com um *smartcard* embutido num cartão microSD, e aplicando essas funcionalidades ao já desenvolvido serviço agregador de armazenamento na nuvem Storekeeper [PASC16]. Assim, será possível fornecer uma autenticação de utilizador muito forte e funcionalidades de gestão de chaves em dispositivos móveis.

Esta nova abordagem combinará a funcionalidade do agregador de armazenamento na nuvem com a interface segura e o desempenho dos processadores ARM que suportam TrustZone, e a segurança de alto nível e robustez dos *smartcards*.

Palavras-chave: ARM TrustZone, TEE, Mobile Devices, Smartcard, Cloud Storage

Abstract

Current cloud storage services have limited security and require users to grant full access privileges to individual cloud storage accounts. With recent advances in cloud storage technology, the need for mobile devices that can manage identities and provide convenient access authorization reliably and securely is an ever-growing need and concern. Towards this several solutions have been proposed and designed by the industry. In regard to general use processors, ARM TrustZone provides a much more reliable platform and user interaction than 'normal world' applications. On the other hand, dedicated constrained systems such as smart cards (SC) also exist, which are considered very secure due to their isolation and specifications.

The proposed work consists in the creation of a highly secure cloud storage aggregator service, by merging the functionalities of the ARM TrustZone (now present in most mobile phones), with a smart card embedded on a microSD card, and applying these functionalities to the already developed cloud storage aggregator service Storekeeper [PASC16]. With this, it will be possible to provide very strong user authentication and key management functionalities on mobile devices.

This new approach will combine the functionality of the Enhanced Cloud Storage Aggregation Service with the secure interface and performance of the ARM processors supporting TrustZone, and the high-level security and robustness of smart cards.

Keywords: ARM TrustZone, TEE, Mobile Devices, Smartcard, Cloud Storage

Contents

List of Tables

List of Figures

Nomenclature

AAM Multos Application Abstract Machine

AES Advanced Encryption Standard

APDU Application Protocol Data Unit

API Application Programming Interface

AT Access Token

AVD Android Virtual Device

AXI AMBAAdvanced eXtensible Interface

CPU Central Processing Unit

DMA Direct Memory Access

DPA Differential Power Analysis

DRAM Dynamic Random Access Memory

EMV Europay-MasterCard-Visa

GPCS GlobalPlatform Card Specification

I/O Input/Output

IC Integrated Circuit

ID Identification Card

IT Information Technology

JCA Java Cryptography Architecture

KF File Key

KL Access Key

KS Shared Key

KT Device Key-Pair

KU User Key-Pair

MEL Multos Executable Language

microSD Micro Secure Digital

NS Non-secure

OMAPI Open Mobile API

OS Operating System

PC Personal Computer

PIN Personal Identification Number

PKI Public Key Infrastructure

REE Rich Execution Environment

RSA Rivest-Shamir-Adleman

RTE Run Time Environment

SCR Secure Configuration Register

SC Smart Card

SDK Software Development Kit

SDS Storekeeper Directory Server

SGX Software Guard Extensions

SIM Subscriber Identification Module

SMC Secure Monitor Cal

SoC System-On-Chip

SPA SimplePower Analysis

SSL Secure Sockets Layer

TA Trusted Application

TCB Trusted Computing Base

TCG Trusted Computing Group

TEE Trusted Execution Environment

TLB Translation Lookaside Buffer

TPM Trusted Platform Module

UI User Interface

USB Universal Serial Bus

VM Virtual Machine

Chapter 1

Introduction

Cloud storage services are currently a commodity that allows users to store data persistently, access it from everywhere, and share it with friends or co-workers. Due to the proliferation of cloud storage accounts and lack of interoperability between cloud services, managing and sharing cloud-hosted files is a complex task for many users. Cloud aggregator systems provide users with a global view of all files in their accounts and enable file sharing between users from different cloud services. In spite of a considerable usability improvement, with existing cloud aggregator services, users incur additional security risks. In particular, since cloud infrastructures are owned and managed by the cloud service provider, the users have to grant to the cloud aggregator full access permissions to users' cloud storage accounts so that the cloud aggregator can automatically exchange file updates between different cloud storage providers. This results in the customer data being outside of its control and introduces significant security and privacy risks concerning the confidentiality and integrity of data.

The need for mobile devices that can manage identities and provide convenient access authorization reliably and securely is an ever-growing need and concern. Towards this, several solutions have been proposed and designed by the industry. In regard to general use processors, ARM now provides TrustZone and Intel provides SGX secure computational environments. On the other hand, dedicated constrained systems such as Smart Cards (SC), which are characterized by having an integrated circuit embedded in a card body, and the components for transmitting, storing, and processing data also exist, which are considered very secure due to their isolation and specifications [PS19a].

1.1 Motivation

In spite of all the research that has been done over the past years on cloud security, the problem of providing secure cloud storage aggregation has been overlooked. Storekeeper [PASC16] is a security-enhanced cloud storage aggregator service that enables file sharing on multi-user multi-cloud storage platforms while preserving data confidentiality from cloud providers and from the cloud aggregator service. To provide this property, Storekeeper decentralizes most of the cloud aggregation logic to the client-side enabling security-sensitive functions to be performed only on the client endpoints. Even though Storekeeper provides privacy-preserving cloud aggregation by pushing the security logic from the cloud aggregator to trusted clients' endpoints, it creates multiple security risks if the client endpoint is attacked. If the Storekeeper client mobile device is compromised, a regular file-based Keystore can be copied and the device can be a target of attacks where this file can be easily exposed, if this occurs it results in the compromise of the user private keys, resulting in the violation of the users' identity.

When dealing with highly sensitive data, Storekeepers' traditional operating system file-based Keystore approach to protect private cryptographic keys of identity certificates is not secure enough for more critical settings where stronger security properties are required.

1.2 Topic Overview

To address the concerns outlined above, the proposed work consists on the creation of a highly secure system, denominated SmartZone, by merging the functionalities of the ARM TrustZone, now present in most mobile phones, with a smart card embedded on a microSD card [SD 14]. The target application for these functionalities is the distributed cloud storage service Storekeeper that has already been developed [PASC16]. SmartZone will combine the security of the Enhanced Cloud Storage Aggregation Service with the secure interface and performance of the ARM processors supporting TrustZone, and the high-level security and robustness of smart cards, which provides a much more reliable platform than 'normal world' applications, alongside very strong user authentication and key management functionalities on mobile devices.

1.3 Objectives

This research aims to address the problems of dealing with highly sensitive data on a vulnerable mobile device that is being used as a client of a distributed cloud storage system. The goal is to provide strong user authentication and key management functionalities by using a hybrid solution marrying trusted execution environments with smart card technology.

The requirements for this system are:

1. Allow for greater isolation between the operating system and the critical operations performed by Storekeeper;
2. The user must be able to use the system independently of his location;
3. The performance overhead must be low enough to not critically impact the system usability;
4. The overall system and the authentication operations must be "user-friendly" and easy for the user.

An important aspect of SmartZone is that the security properties are achieved based on strong cryptographic guarantees as well as physical and access control mechanisms.

1.4 Main Contributions

This thesis presents the design, implementation, and evaluation of SmartZone, a distributed solution, consisting of an enabled ARM TrustZone mobile application and a smart card applet. The target application is the privacy-preserving cloud aggregation service Storekeeper. SmartZone addresses the problems with dealing with highly sensitive data on a vulnerable mobile device that is being used as a client of a distributed cloud storage system. Using smart card technology for key management and storage provides tamper-proof storage and secure computation capabilities. ARM TrustZone provides clear isolation from a possibly compromised normal world OS and guarantees that sensitive data is only handled in the trusted execution environment, being used for file encryption and decryption operations it provides security advantages when compared to using the device storage and normal world environment.

1.5 Thesis Outline

The rest of this document is organized as follows. Section 2 presents an overview of the technology related to this work and methodologies that help to achieve these goals. Section 3 presents a general work overview, which focus on presenting the SmartZone solution, the architecture of the system and the main operations. Section 4 describes the technical details of our system, the implementation challenges, as well as the several components used. Section 5 presents the results of the experimental evaluation, how the desired goals and requirements are achieved, and some insight into security considerations and limitations of our work. Finally, Section 6 concludes this document by summarizing our findings and discussing future work.

Chapter 2

Related Work

While cloud storage has a lot of potential unless the issues of confidentiality and integrity are addressed many users will be reluctant to fully trust these services with their personal data. One way to do it is by using trusted execution environments combined with the high level of security and robustness of smart cards, on mobile devices.

This section addresses these topics in order to get a better understanding of the related work done regarding our proposal. To do so we must first start by understanding the fundamentals of cloud storage, more specifically Storekeeper, and then proceed to explain the benefits and importance of using smart cards and trusted execution environment technology in our work. Section 2.1 defines cloud storage is and how it can be used, with focus on the current Storekeeper solution. Section 2.2 explores current smart card technology and gives some insight into its uses and benefits. Section 2.3 explains what a trusted execution environment is, focusing in particular on the ARM Trustzone, while still mentioning other existing trusted execution environments.

2.1 Cloud Storage

Firstly, cloud providers put a lot of effort into making such services highly available, allowing users to access their files anytime everywhere (as long as there is network connectivity). Multiple file replicas are stored in the cloud, assuring higher fault tolerance than storing files locally on users' desktops or servers. Furthermore, users are provided with friendly interfaces that allow them to access their files and share them with friends. All users have to do is to log into their accounts by providing a username and password before they can navigate through their files and open them.

Highly available cloud storage services such as Dropbox, Google Drive, or Microsoft OneDrive have become an invaluable resource for consumers by allowing them to access their files anytime

everywhere (as long as there is network connectivity). Since the amount of free space provided by default per cloud storage account is relatively small, and the lack of interoperability between cloud services for sharing files between cloud providers is not directly permitted, managing and sharing cloud-hosted files can be very difficult for users. To address this hurdle, services like Boxcryptor [Box], which provides end-to-end file encryption at the client-side, and Odrive [Odr] provides a multi-user multi-cloud file sharing layer that exposes to the users a unified view of all files and enables seamless file sharing across cloud accounts. However, users take security risks since in all these systems the cloud aggregator has full access to the user cloud storage accounts, resulting in users giving away their privacy and entrusting access credentials to the cloud aggregator.

Systems like BlueSky [VSV12] focus on securing single-user single-cloud platforms, acting like a local storage server that backs up data to the cloud. Security is normally achieved by interposing an encryption layer between the user's client and the cloud provider. Therefore, all client data is individually encrypted and protected with a keyed message authentication code by a proxy server before sending it over the network, so the cloud provider cannot read private data. SPORC [FZFF10] also focuses on a single cloud service, but allows for secure file sharing between users, with the help of a central untrusted server whose sole purpose is to order and store client-generated operations. Data confidentiality is achieved by encrypting all operations under a symmetric key, unknown to the central server.

Storekeeper is a distributed system comprised of a client application (installed on the user device) and a centralized cloud aggregator server, which provides a secure cloud aggregation service for multi-user multi-cloud storage platforms, resorting to mechanisms for user identity management between different cloud providers, and key management and storage solutions. By having the security-sensitive logic on the client endpoint it is possible to provide end-to-end data confidentiality without entrusting sensitive user account credentials to the cloud aggregator server.

The two main components of Storekeeper are: the client application and the Storekeeper Directory Server (SDS). On the user device the client application along with a local cache of the files stored on the cloud store, serve as an interface in order to interact with the system, while on the other hand the SDS runs on a dedicated server and is responsible for managing all the meta-data associated with the users, files and stores. The cloud accounts are represented by stores that are provided by the user and stored on the SDS, by having the files stored on these stores results in the files not being directly stored on the SDS. The SDS server must be managed by a trusted and dedicated administrator which is responsible for registering the users

in the system.

As depicted in the Figure 2.1, illustrating a simple deployment scenario, a user must be registered in the platform by the dedicated administrator so that it can be able to log into the system via a username and password, in order to register their personal cloud accounts. Storekeeper will interpret them as stores that are only accessible through the client-side only, in order to ensure that the user retains control of their accounts. The user will have a unified view of all of his files since the file encryption is performed on the client-side.

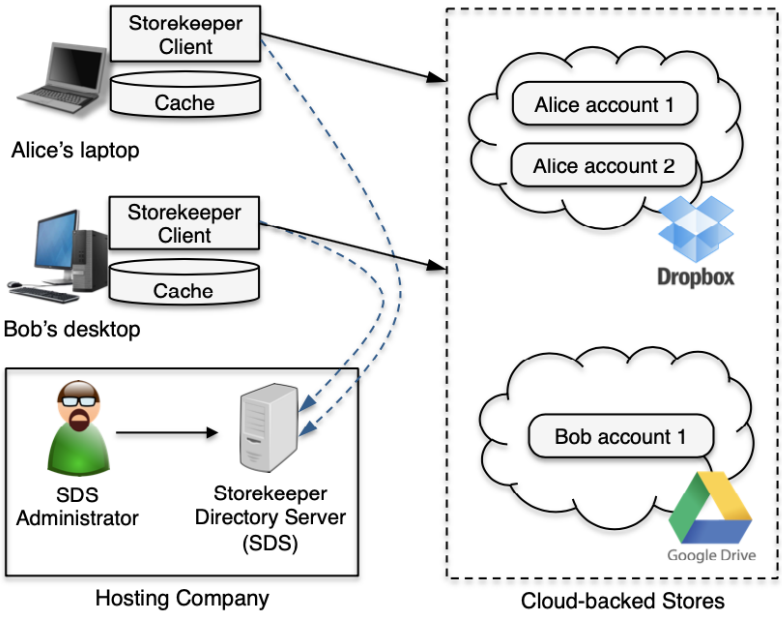


Figure 2.1: Storekeeper system overview [Jam12].

Storekeeper depends on access tokens and user keys, as a basis to overcome the challenges involving user authentication when accessing cloud stores and confidentiality protection of user files, respectively. In order to access a cloud store through the respective API, the Storekeeper clients provide an *access token* (AT), granted by the cloud service, to authenticate itself towards the cloud service, preventing this way the need for the user to interactively input username and password. These access tokens are stored securely by the Storekeeper since the leakage of such tokens would allow for unrestricted access to the users' stores.

When it comes to end-to-end confidentiality, Storekeeper generates a symmetric key, designated *file encryption key* (KF), for each file before sending them to the cloud. In order to protect this key, Public Key Infrastructure (PKI) is used by having the user generating a public key-pair, designated *user key* (KU), and encrypt the file encryption key with the user public key. Assuming that the private user key is maintained private the access to the file encryption key is restricted. In order to prevent the user from having the responsibility to maintain the

user credentials (access token and user keys), Storekeeper resorts to another symmetric key designated *access key* (KL), which it uses to generate a cryptogram, containing the username, user private key (KU) and access tokens (AT) for the respective user, and store it on the SDS:

$$u, \left\{u, KU^{-}\right\}_{KL}, [\{u, AT_{s0}\}_{KL}, ..., \{u, AT_{sn}\}_{KL}] \tag{2.1}$$

This tuple corresponds to: username u, encrypted user key, and list of encrypted access tokens ATs, one for each store s (from 0 to n) that the user has registered in the server. When the user wishes to access the credentials it needs to authenticate himself with a username and corresponding password, this way the user only needs to memorize the username, login password, and the access key.

When a user creates a file, the local client must first encrypt the file with a file encryption key and then upload the resulting ciphertext to the file's home store. The file key is randomly generated and is specific to that file. To protect the file key, the client encrypts it with the public part of the user key and sends it to the SDS. To read the file in the future, the client downloads the encrypted file from the home store, and fetches from the SDS the encrypted credentials: file encryption key and private part of the user key. Next, based on the access key, the private part of the user key is obtained, which in turn uses to decipher the file encryption key and consequently decipher the file itself. Since both file and file keys are encrypted, neither SDS nor cloud provider can read the file contents. In order to share a file with another user, the file encryption key is encrypted with the other user public key.

Storekeeper is mainly implemented in Java and uses 256-bit AES symmetric encryption keys and 1024-bit RSA asymmetric encryption keys alongside the JCA framework in order to perform cryptographic operations. All the communication between the client and the SDS is performed over SSL which provides authentication, integrity, and confidentiality of the exchanged messages. When it comes to performance evaluation, Storekeeper adds a performance overhead between 6 and 10%, according to the files size, due to the system interactions and cryptographic operations.

Storekeeper consider cloud providers to be potentially malicious with respect to violation of data confidentiality, and honest but curious regarding to passively listening to the exchanged messages in order to learn any sensitive data. Communication channels are assumed to be insecure, since they can be eavesdropped or manipulated but, side-channel attacks are not taken into account. By not handling or storing the keys in a secure way, an attacker can obtain the access key, followed by the private key-pair of a user, and consequently obtain the file encryption keys of a file or the access token, violating the end-to-end confidentiality and authentication premises. The integrity of data and meta-data is also not guaranteed by Storekeeper when

intentional modifications are performed by the cloud aggregator or cloud providers [PASC16].

2.2 Smart Cards

Smart cards are perhaps some of the most widely used and underestimated electronic devices in use today. In many cases these devices are in the front-line, defending citizens and systems alike against attacks on information security [May17]. A smart card is a portable, tamper-resistant platform that provides secure storage of confidential data and can protect against unauthorized access and manipulation. It is used in many different industries that come in different sizes and forms such as SIM cards, identification cards, USB sticks, etc. Most smart cards are embedded with multiple silicon integrated circuit (IC) chips, that form a microprocessor, which provides numerous functions like, data storage, manipulation of data, and persistent secure storage in order to store data to identify a user or perform cryptographic operations. An essential requirement for this is the availability of chip hardware designed and optimized for this purpose, along with suitable cryptographic algorithms for protecting confidential data. Since the data can only be accessed via a serial interface that is controlled by the operating system or security logic, this prevents the ability to read the stored data from outside the card. In principle, hardware and software mechanisms can both be used to restrict the use of the memory functions (reading, writing, and erasing data) and subject them to specific conditions. This provides flexibility to construct a variety of security mechanisms, which can also be tailored to the specific requirements of a particular application [Tar09].

These cards are used in many current and real-world systems and are proposed for many future applications. In Telecommunications on the form of Subscriber Identity Module ((U)SIM's) cards that provide authentication and confidentiality to mobile communications [ME17]. On Payment systems with the Europay-MasterCard-Visa (EMV) chip and PIN standard developed by Europay, Mastercard, and Visa, in order to securely store sensitive information within the card responsible for realizing payments, which was not possible with the previous carrying mediums [MM17]. When it comes to the Transportation sector there is a growth since smart card technology is able to offer flexibility, fraud protection, and efficiency making a good solution for public transportation in ski areas, trains or buses [May17]. Concerning Identity Cards/Passports and Health Care, smart cards provide a more secure identification of citizens and storage of highly sensitive personal health data, respectively [Lie17]. In regards to IT and Access Control, it is quite common for employees to be given an ID card to gain access to their place of work and perform signature operations, which among other things are well suited to protecting all sorts of business processes in public networks [RE10].

Smart cards can also be classified on the basis of their data transmission method. Data can be transmitted using mechanical contacts on the surface of the card or wirelessly using electromagnetic fields. Also in regards to smart card with chip element, they can be divided into two groups that differ in both functionality and price: memory cards and processor cards, as can be seen on Figure 2.2. The functionality of memory cards is usually optimized for a particular application which severely restricts the flexibility of these cards but, in contrast makes them quite inexpensive. When it comes to processor cards they are very versatile in use.

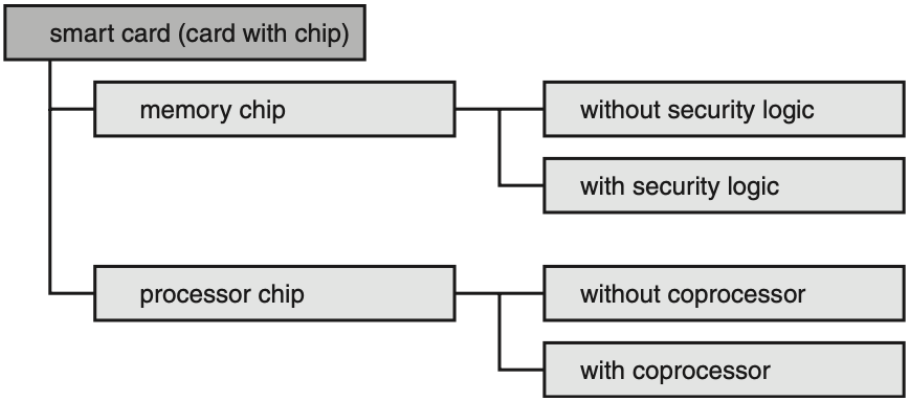


Figure 2.2: Classification of smart cards by chip type [RE10].

In combination with the ability to compute cryptographic algorithms, this allows smart cards to be used to implement convenient security modules that can be carried by users at all times, for example in a purse or wallet. Some additional advantages of smart cards are their high reliability and long life compared with magnetic-stripe cards, whose useful life is generally limited to two to three years [RE10].

The standards that govern the design and functionality of smart cards have been established by the ISO-7816 family of standards [Jam12]. This requirements go from how a smart card should communicate with a terminal or external application to the physical aspects of the card, in order to provide a guideline for how a smart card should be designed.

To communicate with smart cards, a specific type of message must be used. These messages are on the Application Protocol Data Unit (APDU) format, as specified by ISO/IEC 7816-4. The protocols used to communicate with smart cards are client-server based, where the card is the server, which only answers to requests from the client. The messages sent from the client must be in the request APDU format as follows:

- **CLA [1 Byte]:** Class of the instruction to be executed.
- **INS [1 Byte]:** Instruction to be executed.
- **P1-P2 [2 Bytes]:** Instruction parameters for the instruction.

- **Lc [1 Byte]:** Encodes the size of data to follow.
- **Data [Lc Bytes]:** Instruction specific data.
- **Le [1 Byte]:** Maximum number of expected bytes for the response message.

And from the smart card, the messages must respect the format:

- **Data [up to Lc Bytes]:** Response data.
- **SW1-SW2 [2 Bytes]:** Message processing status, where 0x9000 (hexadecimal) represents success.

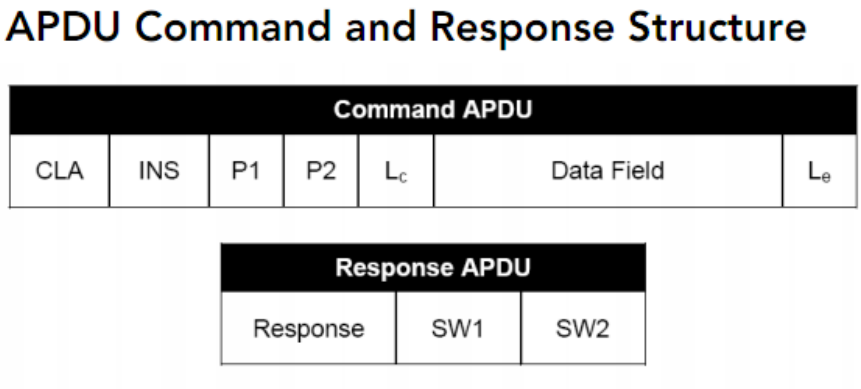


Figure 2.3: Application Data Protocol Unit *(ADPU)*

It is practically impossible to configure a smart card to provide perfect security that can defend against everything and everybody. If sufficient effort and expense are devoted to an attack, every system can be breached or manipulated. However, every potential attacker performs a sort of cost/benefit analysis and the reward of breaking a system must be worth the time, money, and effort necessary to achieve this objective. Most currently known successful forms of attack on smart cards take place at the logical level. These attacks are based on pure contemplation or computation. This category includes traditional cryptanalysis as well as attacks that exploit known weaknesses of smart card operating systems and Trojan horses in the executable code of smart card applications. There are two main types of attack that are considered in smart card security, passive/non-invasive and active/invasive attacks. In a passive/non-invasive attack, the attacker will attempt to derive information by observing, in order to derive knowledge without modifying a smart card. By contrast, an active/invasive attack requires the microprocessor to be exposed or removed in order to directly attack it through a physical mean, typically requiring very expensive equipment and a large investment. This type of attacks involves manipulation of the data transfer or the microcontroller.

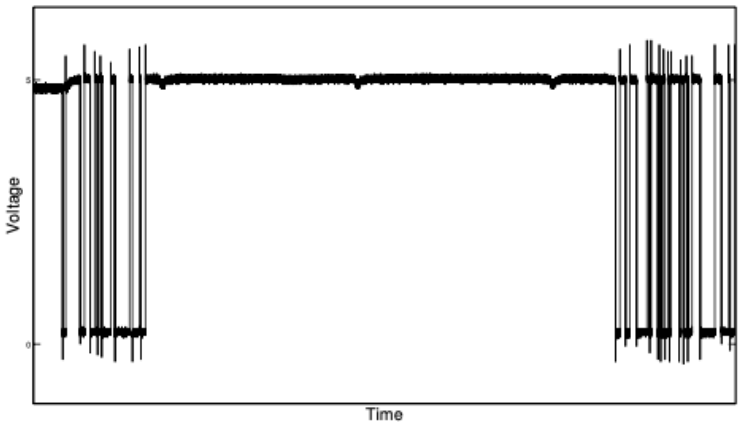


Figure 2.4: The I/O of a smart card command [Tun17].

Side-channel attacks are a class of passive/non-invasive attacks where an attacker will attempt to deduce what is occurring inside a device by observing information that leaks during the normal functioning of the device. If this information can be related to any secret information being manipulated the security of the device can be compromised. An example of this is Timing Analysis, where the attacker observes differences in the amount of time required to perform certain operations, e.g. calculate a RSA signature for different messages, in order to derive the secret key. On Figure 2.4 an example of a trace obtained with an oscilloscope can be seen, where the I/O events on the left side represent the reader sending a command to the smart card, and the I/O events on the right side shows the response. The time required to perform such operation can be determined by observing the amount of time that passes between these two sets of events. The most common form of side-channel attack, when considering smart cards, is the analysis of the instantaneous power consumption, which can be separated in Simple Power Analysis (SPA) or Differential Power Analysis (DPA). On SPA an attacker searches for patterns in order to determine the location of individual functions, then the difference between multiple executions of the same program can be used to determine the data being processed by the program. In contrast on DPA, a measure is firstly performed with known data and then again while it is processing unknown data. By repeating these measurements (Figure 2.5) the noise can be eliminated and the difference between measurements can be used to deduce unknown information. An alternative to measuring the power consumption of a smart card is to measure the instantaneous electromagnetic emanations as a computation is being performed. In order to perform such measurement, a small probe must be used to try to get a strong signal from a specific part of the chip-making it much more complicated to realize, since the tools necessary for capturing such information are more complex when compared to power analysis.

It is also possible to inject faults in microprocessors by varying the supply voltage, and/or

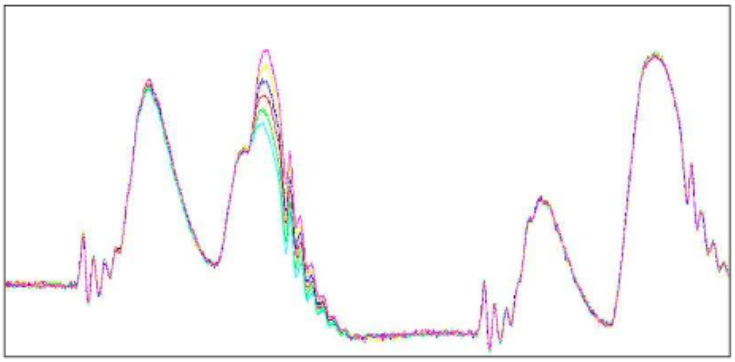


Figure 2.5: Overlaid acquisitions of the power consumption produced by the same instruction but with varying data [Tun17].

external clock, tune chip temperature, use of laser and white light sources, and electromagnetic fluxes. These mechanisms can cause effects like resetting data to a blank state, change data to a random value, or even modifying instructions opcodes executed by the microprocessor.

There are several countermeasures for protecting cryptographic algorithms and microprocessors against side-channel attacks and fault attacks. By using execution and data randomization it is possible to change the order in which operations in an algorithm are executed from one execution to another making it difficult to know what is being manipulated at a given moment in time and manipulate the data in such a way that the value present in memory is always masked with a random value different from each execution, respectively. Also, the introduction of random delays at different points in the algorithm execution, although does not prevent an attack, can be used to increase the time required to attack making it more difficult for injecting a fault in a previous identified target through a side-channel attack. Also, checksums, execution, and variable redundancy prevents data (e.g. key values) or execution results from being modified by a fault by repeating execution of algorithms or reproduce variables in memory in order to be able to verify if these values are correct [RE10].

2.2.1 Formats

The typical smart card has dimensions of 85.6mm by 54mm it is designated as ID-1 and is specified in the ISO 7810 standard. This standard plainly defines an embossed plastic card that is intended to be used for identification, hence the ID-1 format is very easy to handle by being small enough to be able to fit in a wallet but not so small that it is lost very easily. Nevertheless, the demands of modern miniaturization led to a much bigger offer of smart cards formats with a wide range of dimensions that can be used for different purposes, making it more difficult to determine whether a particular card is actually an ID-1 smart card. This resulted in the

development of different formats like ID-00, ID-000, UICC (widely known as SIM card), and Visa Mini but also even extended to other forms factors such as USB plugs with integrated smart card microprocessor or even SD cards. Modern smart card microcontrollers typically work with the USB Full-speed option, which provides a data rate of 12 Mbit/s [RE10].

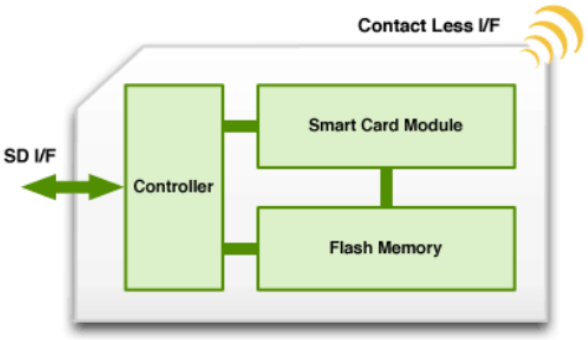


Figure 2.6: An SD card containing an integrated smart card element [MC12].

SmartSD cards [SD 14] are a subset of Advanced Security SD cards that integrates smart card functionality with other components like a microSD card, allowing for the implementation of a secure storage on any mobile phone that supports the ASSD specification. Using this specification brings advantages like flexibility, support and backward compatibility, but also allows an host device to select and communicate with a security system in the SD card while maintaining its most basic functionality, data storage. SD cards in the microSD format are a widespread format included in most mobile phones allowing to bring some of the advantages of smart card element usage to mobile phones [MC12].

Integrating smart card functionality with other components like a microSD card, which is a widespread format in most mobile phones, brings some of the advantages by providing flexibility, support, and backward compatibility, combined with a stable increase in SD cards read and write speeds and storage capacity throughout the years [SD 19]. In the latest years microSD cards read/write speeds and storage capacity have been increasing [SD 19], combining this with the widespread in most mobile phones allows bringing some of the advantages of smart card element usage to mobile phones. Figure 2.6 illustrates the components of a SD card containing an integrated smart card element.

2.2.2 Technologies and Platforms

Smart card hardware and software are very closely coupled together and they both contribute, along with other factors, towards the success of smart card technology. The development of smart card operating system followed a similar path to the development of operating systems

in traditional computing devices. The main efforts for the provision of multi-application smart card platforms was with the introduction of distinct smart card platforms, where Java Card [Orab], Multos [Mul] and GlobalPlatform [Glo] are the most widely utilised smart card platforms [MA17].

A Java Card is a smart card capable of running Java programs, taking advantage of the modularity and encapsulation encouragement brought by the object-oriented programming model of the Java programming language, which relates to more independence between objects and their interaction using well-defined interfaces [VV00]. It also allows more than one applet to be deployed on the card. To develop Java Card applications or applets, the programmers can use a subset of the Java language. Since applications run in a virtual environment, the process of debugging applications is also simplified, since in the development phase they can run into a Java Card virtual machine in the developer's computer. Before deploying the applet, it must be translated into a CAP file which is a specific file format runnable by the Java Card virtual machine on the card.

The Java Card platform is at its core a very minimal subset of Java, complemented with the following specific features catered for the development of secure elements [Ora19]. The first feature is the interoperability between Applets developed with Java Card and any Java Card enabled product, in order to enable reuse across platforms and provide migration capabilities if security requirements evolve. The second feature is the security provided by Java Card technology since it relies on the inherent security of the Java programming language in order to provide a secure execution environment by supporting the latest security standards and state-of-the-art cryptography algorithms, modes, and protocols. The capacity to enable multiple applications from multiple vendors to coexist securely on a single platform offering flexibility is one of these features provided by Java Card technology. The fourth feature correlates to the extensibility and updatability of new services that can be developed during the life-cycle of the smart card, allowing for service providers to constantly adapt to security threats. The last feature is the compatibility of the API with multiple international standards such as ISO 7816 as well as major industry-specific standards like GlobalPlatform. The Java Card architecture holds limited resemblance to the normal Java specifications as can be seen in Figure 2.7.

The GlobalPlatform [Glo] is an independent, non-profit association that is responsible for promoting the GlobalPlatform standards and smart card technology so that interoperability, availability and security of multi-application smart card technology is enhanced. The GlobalPlatform card specification (GPCS) defines a set of logical components, agnostic of the underlying smart card platform, that aim to enhance multi-application smart card security, portability

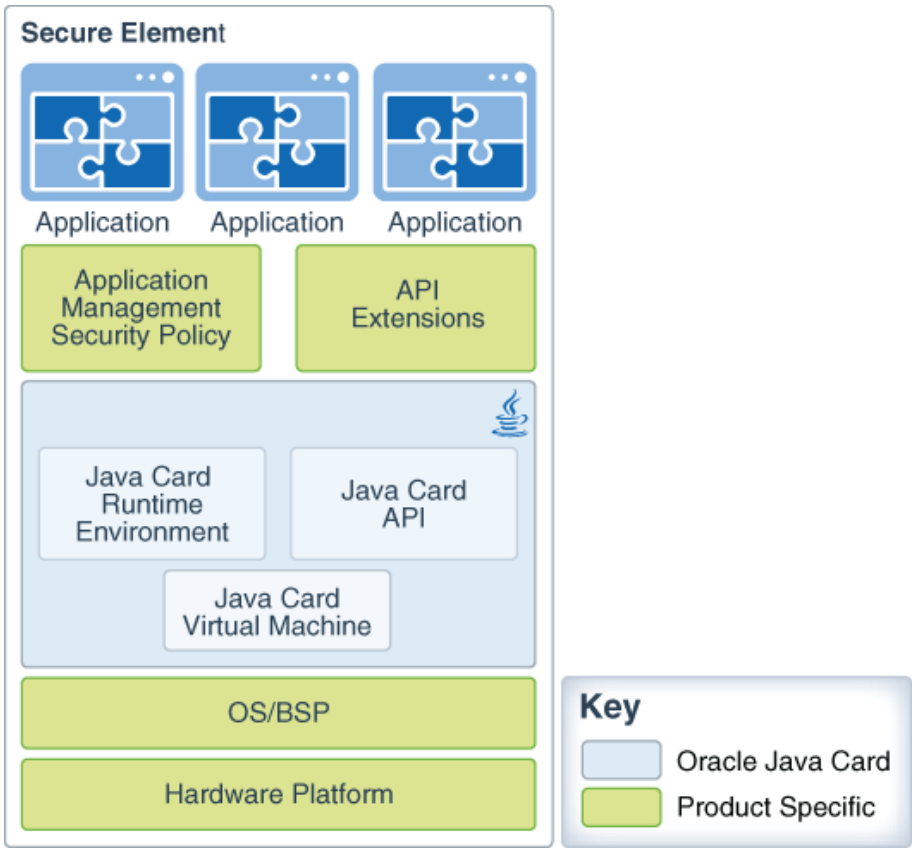


Figure 2.7: Overview of the Java Card Architecture [Oraa].

and interoperability. At the bottom of the suggested architecture we find the smart card microprocessor, on top of the smart card hardware we have the Run-Time-Environment (RTE) (e.g. a smart card supporting the Java Card specification) which is composed by the operating system, a virtual machine (VM) and an application programming interface (API). The RTE is considered as an abstraction layer between the GPCS and the underlying hardware. The API offers application programmers the ability to access the basic functionality defined within the GPCS. The association is putting great effort to make sure that GlobalPlatform implementations are properly tested and that GlobalPlatform developers provide accurate product implementations [MA17].

Multos [Mul] is a multiapplication smart card operating system originating from the attempt to provide a secure and reliable platform for the Mondex system for electronic purses. It was designed from scratch by focusing on security along with the specific details of the underlying smart card microprocessors since it was primarily optimized to meet the requirements of electronic payment systems. Multos is a ISO/IEC 7816-4 compliant operating system and the program code is typically developed in C or Java and translated into the Multos Executable Language (MEL) using a special compiler. The internal architecture is composed by the smart card microprocessor on the bottom, followed by the Multos operating system (OS) offering

the basic required functionality (e.g. I/O, file management, cryptographic operations) and the Multos Application Abstract Machine (AAM) is located between applications and the Multos OS that provides an API that will enable interoperability between different Multos implementations. Implementations of the Multos OS must always adhere to the functionality already defined within the Multos standards and applications and must be digitally signed by a licensed Multos certification service before being loaded into a Multos smart card. At an early stage of its development, it was almost impossible to obtain access to the Multos specification resulting in not being as widely utilized as Java Card, especially when compared with the simplicity, familiarity, and portability of Java Card [MA17].

2.3 Trusted Execution Environments

As mobile operating systems grow in size and complexity, they are increasingly susceptible to software vulnerabilities. A Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel, designed to provide protection against attacks that exploit these vulnerabilities by providing a secure, integrity-protected processing environment, consisting of processing, memory, and storage capabilities. This architecture guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory, and sensitive I/O), and the confidentiality of its code, data, and runtime states stored on a persistent memory. The content of TEE is not static; it can be securely updated. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third parties [SAB15]. This secure execution environment is isolated from the Rich Execution Environment (REE) where the OS and the applications are executed. This architecture makes it possible to design REE applications and services that remain secure even in the face of OS compromise by partitioning them so that sensitive operations are restricted to the TEE and sensitive data, like cryptographic keys, never leave the TEE [EKA14]. The scheduling between the trustlets, applications that are executed in the TEE, and the operating system running on the "normal" environment is performed by the TEE OS. This OS should ensure that no data is leaked between both the environments during context switches and even between trustlets running on the TEE [CDRP14].

2.3.1 ARM TrustZone

TrustZone [ARM09] is a hardware based TEE technology incorporated into ARM processors consisting of security extensions to an ARM System-On-Chip (SoC), covering the processor, memory, and peripherals. These mechanisms can be leveraged by system designers to run

secure services in isolation from the OS, providing the capability to virtually separate hardware into two domains referred to as *"worlds"*. In this manner, it is similar to a hypervisor, but it differs in the way that both worlds potentially have full access to all the hardware while it is still possible to restrict access to certain resources for one of the worlds. It consists of security extensions to an ARM SoC covering the processor, memory, and peripherals. These mechanisms can be leveraged by system designers to run secure services in isolation from the OS. The secure services that can be built range from a simple library to a complete OS [SRSW14].

The software that is executed in the secure world may be bare-metal, containing nothing but security libraries, or it can be a full-fledged OS. Since most platforms provide limited resources, mostly in terms of memory, to this environment, it gravitates to be small in size and only contains the bare essentials. The OS in the secure world can be viewed as if it is run in parallel with the OS running in the normal world. On a multi-core processor, this may often be the case as one core could be dedicated to run in secure mode. Execution on the other cores still continues even if there has been a switch to the secure world. Trusted applications can come pre-installed from the factory or they can be installed during the lifetime of the device. Generally trusted applications must be signed by the manufacturer in order to be installed.

The normal world runs the traditional OS, the Rich OS, which the user interacts with. Common examples include Windows 10 [Mic], iOS [App] and Android [Goo]. Applications that intend to utilize secure services can call on functions made public via a TrustZone API. Applications can request that data be encrypted/decrypted by the secure world which in turn, encrypts the data and returns the encrypted/decrypted data to the application in the normal world. This way, the keys used are never exposed to the normal world.

One of the main features of ARM TrustZone enabled processors is an extra 33rd bit called the non-secure bit (NS bit) to signal in which world the processor is currently executing (i.e., the currently active world). This bit can only be set by the system running in the secure world. When the NS-bit is 0 the processor is operating in the secure world and contrarily it operates in the normal world. This security bit is located on the communication bus called the AMBA Advanced eXtensible Interface (AXI) that is used by the main processor in all memory system transactions, including access to system memory and peripherals. These peripherals can either be located in the same package or chip or outside the package. The security bit is added to this bus to communicate to the peripherals whether the transaction they are receiving is either from the secure or the normal world. This way the processor can enforce permissions on access to ensure that no secure world resources can be accessed by the normal world components. Also, all peripherals should check the security status of the transaction and ensure that they do not

leak any sensitive information from the secure world to the normal one [KK14][Yal18].

The processor only executes in one "world" at a time, hence to run in the other world requires a context switch. TrustZone provides the monitor mode, which preserves the processor state during the world switch, in order to control the context switch between the two worlds. If the normal world requests a switch to the secure world, the CPU must first enter this mode. The monitor mode ensures the state of the world that the processor is leaving is safely saved, and the state of the world the processor is switching to is properly restored. Software in the normal world can force a switch to the secure world either by an explicit call via a dedicated assembly instruction called Secure Monitor Call (SMC) or interrupts, e.g., normal interrupts (IRQ) and fast interrupts (FIQ) configured to be handled in the secure world (see Figure 2.8). To return back to the normal world from the secure world, the software in the secure world can perform a return from an exception and set the NS bit in the Secure Configuration Register (SCR), which is a read/write register that is only accessible by the secure world, i.e., access to SCR from the normal world results in an exception [Yal18].

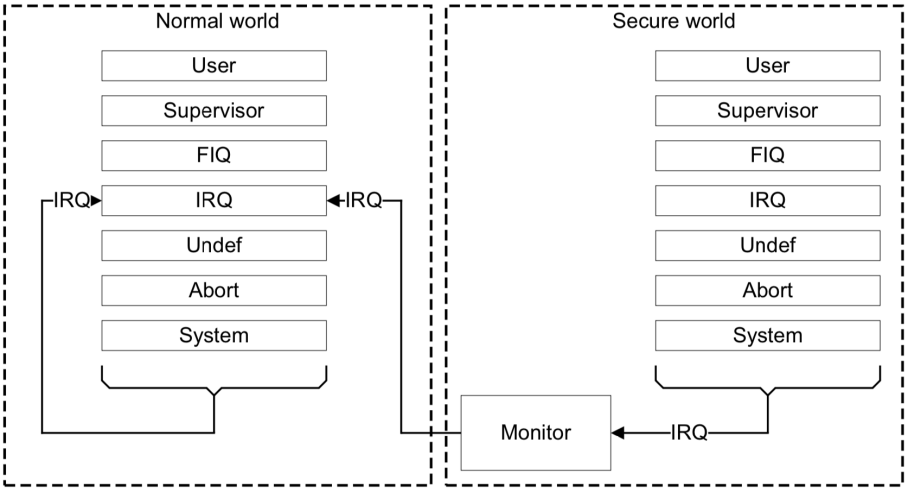


Figure 2.8: One possible IRQ routing in a design with IRQ configured as a non-secure interrupt [ARM09].

As with the separation of the execution environments, the same happens with the physical memory in TrustZone. The memory is partitioned into two distinct regions designated as trusted and untrusted memory. The normal world has access to untrusted memory while the secure world has full access to the memory space, however, the latter generally only uses secure and shared memory. Untrusted memory can also be registered as shared memory, which can be used by both worlds to communicate with each other. What memory belongs to which region can be controlled via the TrustZone Protection Controller which is only accessible from the secure world.

Like memory, devices connected to the TrustZone-aware system are divided into secure and normal devices. The normal world is blocked from using devices designated as secure. In general, secure devices generate FIQs, and non-secure devices generate IRQs. This behavior can be controlled via the TrustZone Interrupt Controller, which can only be accessed while executing in the secure world like the TrustZone Protection Controller mentioned previously [KK14].

The security of the system is achieved by partitioning all of the SoC's hardware and software resources so that they exist in one of two worlds - the Secure world for the security subsystem, and the Normal world for everything else. A design that places the sensitive resources in the Secure world, and implements robust software running on the secure processor cores, can protect almost any asset against many of the possible attacks, including those which are normally difficult to secure, such as passwords entered using a keyboard or touch-screen. This removes the need for a dedicated security processor core, which saves silicon area and power, and allows high-performance security software to run alongside the Normal world operating environment [ARM09].

Despite this, ARM TrustZone is vulnerable to a different set of attacks that can be explored through direct or indirect critical links between the attacker and security-critical applications. These critical links can be broadly categorized into three categories [MBG19]: **Type 1**, where the attacker makes links to security-critical applications through privileged software. **Type 2**, attacker links to security-critical applications through micro-architectural events of hardware, and finally **Type 3** where the attacker makes links to security-critical applications through directly probing hardware. The resilience of ARM TrustZone to set of attacks is as follows:

- **Type 1** - Secure against *Memory Mapping Passive/Active Attacks* because secure world's page-tables and TLBs are divided into two separated spaces, and managed by trusted system software. Also secure against *Direct Memory Access (DMA) based Attacks* and *Firmware based Attacks*, since DMA accesses are rejected in the secure world and the firmware is part of the secure world. When it comes to *Denial of Service Attacks*, ARM TrustZone is weakly secure because although trusted operating system manages the page tables and TLB, monitor mode still has a link with untrusted system software.

- **Type 2** - Not secure against *Cache Bases Side-Channel Attacks* because, albeit cache lines are extended with a NS-bit to distinguish them from belonging to secure or normal world, the allocation of these are performed on the basis of memory access demand from the world at runtime, making it possible for the attacker to evict the desired cache line. When it comes to *Row-hammer Attacks* ARM TrustZone is secure if all secure data and code are in the memory of SoC. In the case of external DRAM, the security depends if

the developer includes, or not, an integrity check mechanism in order to check if data on DRAM has been accessed and modified without authorization.

- **Type 3** - For *Port Attacks* and *Bus Tapping Attacks*, ARM TrustZone is safe if debug ports are disabled and if secure data is limited to memory within the Soc respectively. Finally, ARM TrusZone is not secure against *Chip Attacks* or *Power Analysis Attacks*.

2.3.2 Other Implementations

While the main focus of this project is the ARM Trustzone, it is worth noting that there have been other companies that have been exploring the trusted execution technology in order to improve the security of their systems.

The Trusted Platform Module (TPM), Specified by the Trusted Computing Group (TCG), allows a system to provide evidence of its integrity and to protect cryptographic keys inside a tamper-evident hardware module. It consists of a coprocessor, which is typically located on the motherboard. Its primary purpose is to serve for bootstrapping trust on the local platform: it is responsible for storing the software measurements computed during the trusted boot process of the system, and for securely storing cryptographic keys for remote attestation and data sealing operations. Today, Trusted Platform Modules (TPMs) are available for nearly every PC platform, ranging from desktop machines to notebooks. These TPMs provide the basic building blocks for different security services like storing integrity measurements of the installed and loaded software during boot, authenticated reporting of these stored values to remote verifiers or binding certain data like keys to certain platform configurations [DW09]. In itself, the TPM does not provide the means for executing security-sensitive code in isolation. Instead, it is typically used in tandem with trusted hypervisors or OSes, which will then be responsible for providing confidentiality and integrity protection of such applications [PS19b].

However, TPMs are deprecated for mobile and embedded applications, from a certain point of view, it seems simple to put a microcontroller-based TPM like the ones used on desktop machines on a mobile platform. However, each new chip on a phone's motherboard increases the cost of this device, not to mention the additional power consumption of the extra chip. To add to this, the main shortcoming of the TPM is that it does not provide an isolated execution environment for third-party, thereby reducing its functionality to a predefined set of APIs [SAB15].

Intel Software Guard Extensions (SGX) is a set of extensions for the Intel architecture, introduced in 2015 with the sixth-generation Intel Core microprocessors based on the Skylake microarchitecture, that aims to provide integrity and confidentiality guarantees to security-sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc)

is potentially malicious. Intel SGX implements secure containers for applications without making any modifications to the processor's critical execution path, and the remote computation service user uploads the desired computation and data into this container. The trusted hardware protects the data's confidentiality and integrity while the computation is being performed on it. SGX does not trust any layer in the computer's software stack (firmware, hypervisor, OS). Instead, SGX's Trusted Computing Base (TCB) consists of the CPU's microcode and a few privileged containers. SGX processor does not provide orthogonal privileged levels to secure application as in the TrustZone, instead, the application executes on the same untrusted operating system but security is achieved by matching with hardware managed meta-data, which OS cannot read or write [CD16].

Intel SGX and ARM TrustZone are popular trusted execution environments. Both Intel SGX and ARM TrustZone are hardware-assisted trusted execution environments but the mechanism behind making the trusted environment for trusted applications are different. Intel SGX creates a trusted environment for trusted applications such that it executes over existing untrusted system software. Whereas, ARM TrustZone creates a new trusted world for trusted applications that execute over trusted system software and hardware that is only visible to the trusted world.

In terms of vulnerabilities Intel SGX shares almost the same resilience as ARM Trustzone, only differing when it comes to *Memory Mapping Passive Attacks* where SGX is not secure because enclave's page-tables and TLB's are managed by untrusted system software [MBG19]. Also, Intel SGX is not secure against *Denial of Service Attacks* since these containers are executing over untrusted system software.

2.3.3 Android Keystore

Android OS [Goo] is an operating system developed by the Open Handset Alliance consortium led by Google. The operating system is based on a Linux kernel that is modified to better fit a mobile operating system. Although the Android operating system and its packages are open source, only upon release of a new final version is the source code released. Most apps are developed in Java, but C++ is also supported, however when it comes to OS services on Android these are mostly written on the latter. Android provides the security of the Linux kernel by providing a secure inter-process communication ensuring a good isolation (Application Sandbox) between applications, the operating system or the user. Since applications must declare which permissions they may require, the smart card API takes advantage of this architecture by defining a permission class that the application must request to obtain access to the API [MC12][And].

Android Keystore is the Android Framework API and component used by apps to access

Keystore functionality. It is implemented as an extension to the standard Java Cryptography Architecture APIs, and consists of Java code that runs in the application's own process space. Keymaster TA (trusted application) is the software running in a secure context, most often inside TrustZone on an ARM SoC, that provides all of the secure Keystore operations, has access to the raw key material, validates all of the access control conditions on keys, etc.

2.3.4 Uses of ARM TrustZone

Nowadays anyone has access to wireless networks very easily and has access to computing devices with strong computing power and high portability, so the need for efficient methods to sharing or synchronizing his or her data among several devices has arisen. Cloud storage services are one of the possible solutions and has become very popular, but due to the nature of cloud storage service there are several security problems such as data leakage, modification, or data loss.

To handle these problems, Jaebok Shin *et al.* proposed a secure data access control method of cloud storage named Data Firewall Cloud (DFCloud) [SKPP12]. DFCloud aims to leverage a TrustZone-assisted TEE on users' mobile devices to provide secure access control capability to cloud storage services. This is achieved by using client-based encryption, relying on the use of TPM functionalities to manage encryption keys, assuming that each client mobile device is ARM TrustZone enabled and that DFCloud performs remote attestation on each client in order to prevent credential or data leakage.

DFCloud architecture is comprised of three different entities (see Figure 2.9), described below [SKPP12]:

- **Client** is a TrustZone enabled mobile device where a user can access the data, through the File Explorer running on the normal world, and perform a series of operations like listing either local or cloud files, execute, delete, copy and move them. For file encryption/decryption, key management, and platform attestation operations, the execution is performed in the secure world.

- **DFCloud server** acts as a proxy server and has the responsibility of maintaining the File Metadata Store which consists of file owner information and data block addresses pointing to the location on the cloud storage service.

- **Cloud Storage Services** consists of the File Handler, which manages data block encryption/decryption, the Attestation Component, which measures the integrity of the OS image and bootloader, libraries and process, and finally, the Key Management Component controls the creation and distribution of keys.

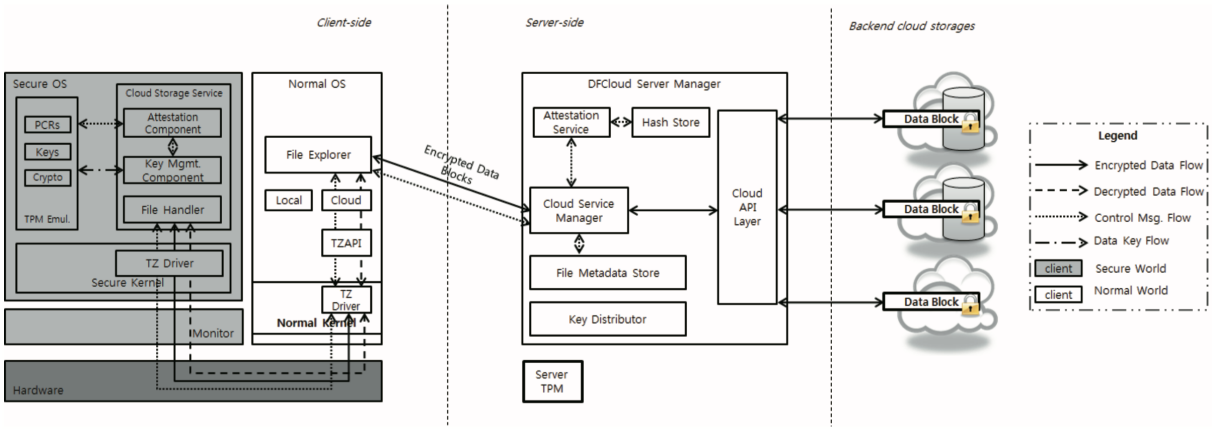


Figure 2.9: DFCloud architecture [SKPP12].

By relying on the TEE, an attacker that manages to compromise the local OS will not be able to recover the keys and the content of the files. However, this system comes with assumptions that may become serious safety problems. For example, this solution does not provide any safety assurances regarding integrity, consistency, or availability of providers. Also by focusing only on data leakage and not taking to account integrity, consistency, or availability of providers. Finally, user authentication is dependent only on user ID and password. An attacker can access all of the content of cloud storage if he or she logs in with valid pair of ID and password. As expected, by performing cryptographic operations when downloading and uploading users' files, the performance is negatively affected due to world switching time increases proportionally to file size [SKPP12].

Beyond relying on client-side TEE, researchers have proposed new applications of TrustZone assisted TEE on the cloud backend itself. Trusted Apps (TrApps) [BGK17], is a platform that targets securing partitioned applications and services in an untrusted cloud environment in order to allow sensitive data processing in the cloud without trusting the cloud provider. This approach is based on the ARM architecture and makes use of TrustZone as an implementation of trusted execution environment.

The architecture of TrApps (see Figure 2.10) comprises the Genode OS, which allows building secure special-purpose operating systems based on a micro kernel architecture, running in secure world and multiplexing the secure world in order to support trusted compartments by various individual cloud customers. In the normal world, we run a standard Linux system that manages the majority of peripherals and especially the network card, and thus, is the only entity with network access. Large parts of secure TrApps applications are supposed to run in normal world, while very small parts of their application logic are offloaded as a secure compartment to the secure world. This requires the partitioning of applications which results in the notion of Divided

Applications (DivAs), that comprise a regular Linux application, called Normal Application (NormA), with one or multiple trusted compartments called Secure Application (SecA).

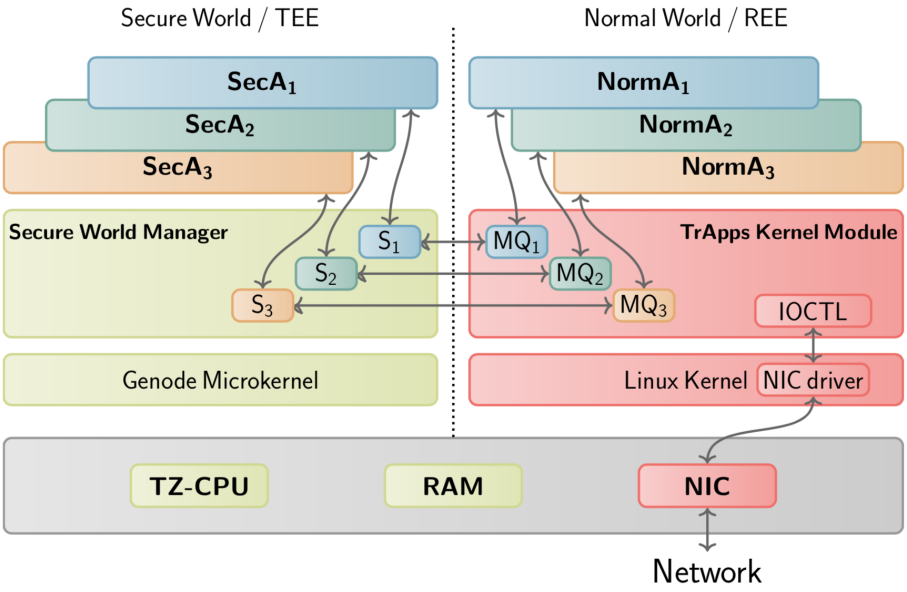


Figure 2.10: TrApps architecture details [BGK17].

When it comes to performance, on average, usage of TrApps imposes an overhead of 36.9%, however, for larger payloads the overhead is lower, as the overhead is mostly due to the constant world switching time of ARM TrustZone [BGK17].

2.4 Summary

This chapter describes the related work regarding cloud storage services, smart card technology, and Trusted Execution Environments. It presents an overview on Cloud Storage services, when it comes to confidentiality and integrity of data, with special emphasis on Storekeeper.

It starts by introducing the concept of Smart Card technology, which solves the problem of secure storage whilst also providing personal identification, authentication, data storage, and application processing. We explore different card types and formats and also reflected on the security provided by this solution, but also grasped on different smart card technologies and platforms from JavaCard to Multos.

We then describe the concept of Trusted Execution Environments that offers an execution space that provides a higher level of security, that allows the processing of sensitive data. Different implementations were analyzed, like Intel SGX and Trusted Platform Modules. Due to our choice of using ARM TrustZone as our hardware-based isolation technology, we decided to focus on the uses of that technology, especially when it comes to the Android operating system.

a

Chapter 3

SmartZone

This chapter describes the design of our system, which consists on a highly secure solution by merging the functionalities of the ARM TrustZone with a smart card embedded on a microSD card. SmartZone is a distributed solution, consisting on an enabled ARM TrustZone mobile application and a smart card applet. The target application for these functionalities is the distributed cloud storage service Storekeeper, where SmartZone will be responsible for the client-side security-critical operations.

In this chapter, we start by giving an overview of the solution and desired functionalities as well as presenting the application use cases (Section 3.1). Afterward, we describe the assumptions and threat model taken by the proposed solution regarding its environment (Section 3.2). We then describe the architecture of our system and its main components (Section 3.3), followed by the basic operations and communication security details between the mobile device and the smart card (Sections 3.4 and 3.5). We then conclude this chapter by summarizing the proposed solution (Section 3.6).

3.1 Overview and Use Cases

The proposed solution addresses the problems with dealing with highly sensitive data on a vulnerable mobile device that is being used as a client of a distributed cloud storage system, using smart card technology for key management and ARM TrustZone for file encryption and decryption operations. Instead of generating and storing keys on the device's normal world environment, a smart card is used to generate and store these keys, granting the security advantages provided by this technology when compared to using the device storage. Since there is no set limit for filesize, operations regarding cipher and decipher of files must be performed on the TEE of the mobile device due to smart card hardware constraints. For this, we use ARM TrustZone,

for cipher and decipher of files when provided with the respective key, in order to perform these operations on an isolated environment from the normal OS.

The target application for these functionalities is the distributed cloud storage service Storekeeper, so two main uses cases must be addressed:

1. **Add File** - In this first scenario a new (unprotected) file is added by the user to the SmartZone app, either this file was created by the user on its mobile device or the user has received this file, for example, via e-mail. Therefore, the respective file keys must be generated, and the file must be ciphered (protected) by the user device, before sending it to the cloud storage service with all respective metadata.

2. **Read File** - For this scenario, the user must request to view a ciphered file (protected) that was previously added to the SmartZone app (via Storekeeper). For this, the solution must decipher the protected file and display its contents to the user. After that, the file must be protected again.

These two uses cases were designed with Storekeeper algorithm in mind, hence special attention was given to the integration between the SmartZone application and Storekeeper.

3.2 Assumptions and Threat Model

We distinguished 3 levels of security regarding our solution that can be correlated with each environment. For all the software running on the normal world, i.e. SmartZone app, we consider it as **Level 0**, and consequently unsafe, so no keys can be disclosed while executing on this level. On **Level 1** layer occurs all the execution that is performed on ARM TrustZone, and consequently file contents and file keys are disclosed at this level. Finally, all the execution regarding key management and storage happens on the smart card, resulting in this environment being classified as **Level 2**, the higher secure level of our solution.

In order to justify our design and implementation, we have to describe the assumptions and threat model considered for our solution that can be separated by each security level described above:

Level 0 - SmartZone Application

- We assume that the normal world runs an untrusted kernel and is used by untrusted users, which however do not have access to the secure world resources and configuration.

- The username and password are shared securely with the smart card on the initialization step. The attacker cannot have access to these fields.
- The SmartZone app does not provide confidentiality of the file when opened.

Level 1 - Secure World

- An attacker can trigger an SMC call and attempt to pass fake data onto the secure world. The attacker may do this call repeatedly to cause a local denial of service.
- We assume that the hardware is correct, i.e., that all TrustZone security features supported by the processor are correctly implemented and cannot be compromised or circumvented by an attacker.
- A symmetric key is securely imported to TrustZone beforehand and shared with the smart card, in order to provide confidentiality between both parties.

Level 2 - SmartZone Applet

- The smart card is tamperproof, no hardware attacks are taken into account.
- No other application internal to the smart card can access the keys resorting to a properly implemented firewall between applications.
- Only the authenticated user has access to the smart card API. If someone authenticates with the user password, he is the expected user.
- The SmartZone smart card applet is reliable, so the client application trusts in the smart card applet.

3.3 Architecture

The proposed architecture is divided among two environments, the device side and the smart card side. As can be seen on Figure 3.1, in the device side we have represented the SmartZone application (client-side) component, running on the user device on top of Android OS. On the smart card side we have the SmartZone applet component running inside the smart card. In green we have the components related or native to Android that are used by SmartZone application, while on blue we have the SmartZone specific components.

In this picture, it is possible to see that in the "normal world" we have the SmartZone Application component. This component interacts with *AndroidKeystore* via the Android Framework

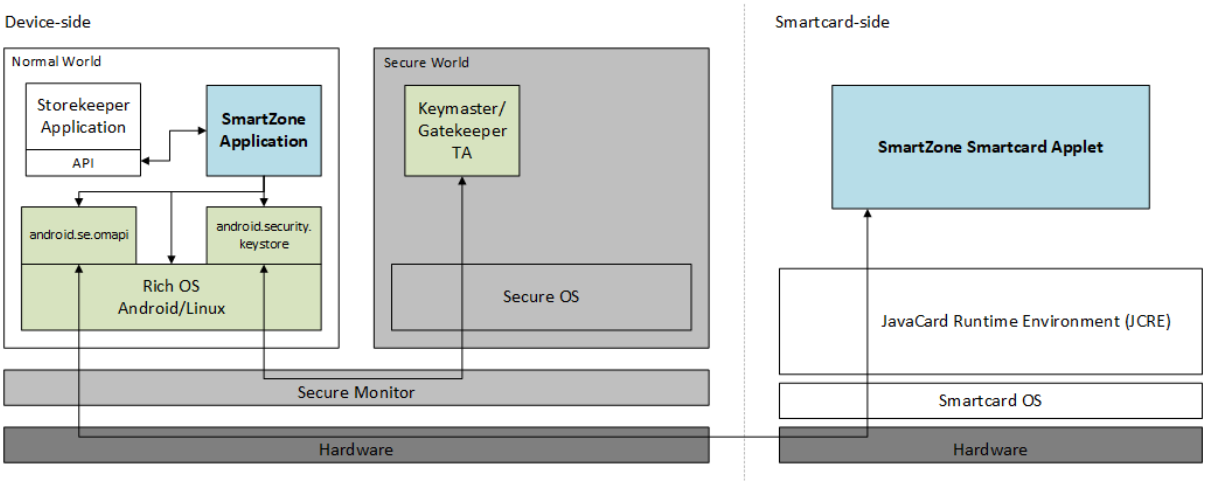


Figure 3.1: SmartZone architecture.

API, by using the Android-specific extension "android.security.keystore" in order to perform secure operations like the cipher and decipher of the files. These operations are performed by the secure software running inside ARM TrustZone (Keymaster TA and Gatekeeper TA). The SmartZone Application component also interacts with Storekeeper Application (Client-side) via specific API in order to provide integration with the cloud storage service.

The SmartZone applet running on the smart card is running on top of JavaCard Platform and implements the desired behaviors of key generation and management. The interaction between both SmartZone components its achieved via the Android-specific extension "android.se.omapi", which is going to be explained in more detail in Section 4.2.

In order to guarantee confidentiality between the two SmartZone components, it is assumed that during the initialization phase a shared symmetric-key KS is generated by the smart card and imported securely to the ARM TrustZone via secure key import mechanisms currently available on Android 9 (API level 28) that allows importing encrypted keys securely into the Keystore using an ASN.1-encoded key format. This is possible by encrypting KS with the user's device public key KT which is generated by Android Keystore. The encrypted key in the *SecureKeyWrapper* format, which also contains a description of the ways the imported key is allowed to be used, can only be decrypted in the Keystore hardware belonging to the specific device that generated the wrapping key. Keys are encrypted in transit and remain opaque to the application and operating system, meaning they're only available inside the secure hardware into which they are imported.

When it comes to the keys used by SmartZone and their exposure, they are listed below and is illustrated on the Table 3.1, respectively:

- KF and KR - File-specific keys.
- KU - User key-pair.
- KS - Shared key, securely shared between the client application and the smart card applet for confidentiality between both parties.
- KT - Device key-pair, used in order to securely import keys into the Keystore.

Keys	Normal World	Secure World (TZ)	Smart Card
KU	-	-	✓
KF	-	✓	✓
KR	-	-	✓
KS	-	✓	✓
KT	-	✓	-

Table 3.1: Keys exposure.

We use two file-specific keys for all the required file operations, KF and KR.

3.4 Basic Operation

To present the basic operation of SmartZone solution, we first consider the initialization phase which comprises the user authentication phase, and then the main file operations which are performed when the user wants to open a specific file.

3.4.1 User Authentication

In the proposed solution, where file keys are stored inside the card, there is a need to authenticate the user, in order to assure that no keys will be handed to an attacker. To achieve this, the proposed solution uses Challenge-Response as the authentication protocol. The user, via the SmartZone application, initializes the authentication protocol by sending the username and password to the smart card. After this information is validated by the smart card applet, a challenge is generated and sent back to the SmartZone application. After challenge computation and validation by the smart card applet, a user key-pair KU is generated inside the smart card. The public key of this pair must be used to cipher the file keys while the private key never leaves the smart card. See Figure 3.2.

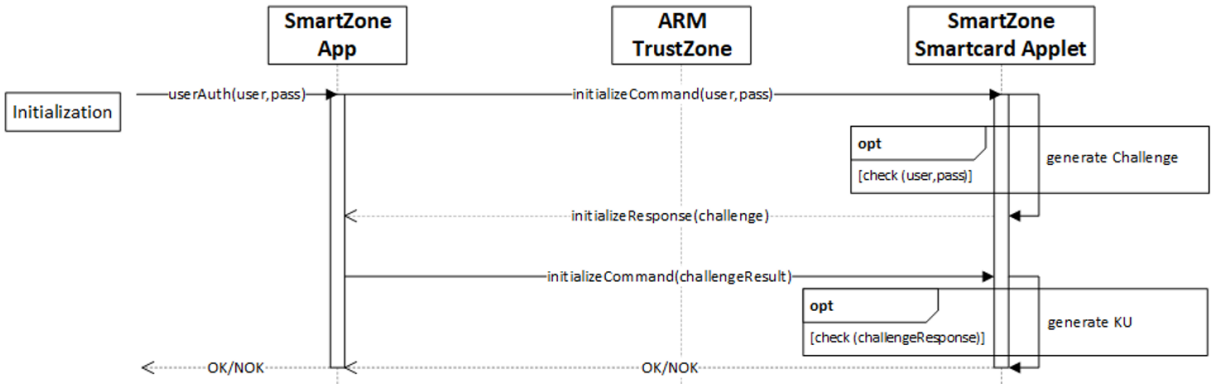


Figure 3.2: SmartZone user authentication.

This authentication mechanism respects Storekeeper user credentials handling protocol when it comes to the fields *username* and *password* for authentication. The user username and password must be sent by the user to the smart card, and it is important to ensure its confidentiality. To enable this confidentiality, the communication between these two parties is ciphered by the shared key KS that was shared previously.

3.4.2 File Operations

When a new file is created/added to SmartZone, the client application running on the device takes care to send a command to the smart card applet in order to generate the file-key KF and read-key KR for the file in question. The key KR is basically an intermediate symmetric key that is shared between all users that can read the file in order to revoke access to the file without re-encrypting it (See Storekeeper). SmartZone uses key wrapping for encrypting one key using another key, in order to securely store it or transmit it over an untrusted channel. This way, the smart card applet wraps KF with KR, and stores the tuple $\{KF\}KR$ on its memory. Then it encapsulates KR with the public part of the user key-pair KU in order to generate the tuple $\{KR\}KU+$. Finally, the smart card applet returns the file-key KF and the wrapped read-key $\{KR\}KU+$ to the SmartZone Application.

After receiving the file-key KF from the smart card applet, this key is imported to the Android Keystore alongside the file to be ciphered. When the file is ciphered by ARM TrustZone with KF, it is returned to the client application in order to be sent with the tuple $\{KR\}KU+$ to Storekeeper. Storekeeper stores the ciphered file $\{file\}KF$ on the cloudstore and the tuple $\{KR\}KU+$ on the Storekeeper Directory Server (SDS).

In order to read the file in the future, either by downloading it from the cloudstore or by having the file ciphered on local storage, all that the user must do is send a command to the

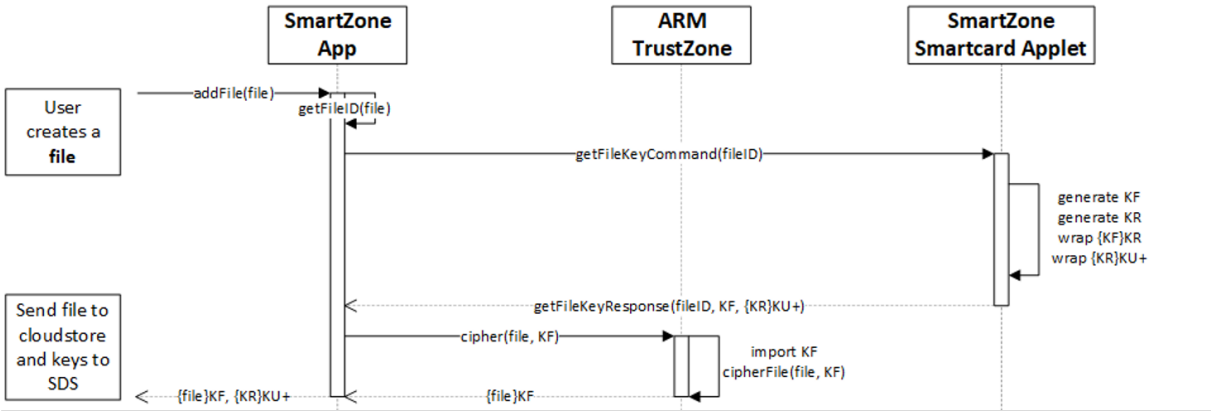


Figure 3.3: SmartZone new file operation.

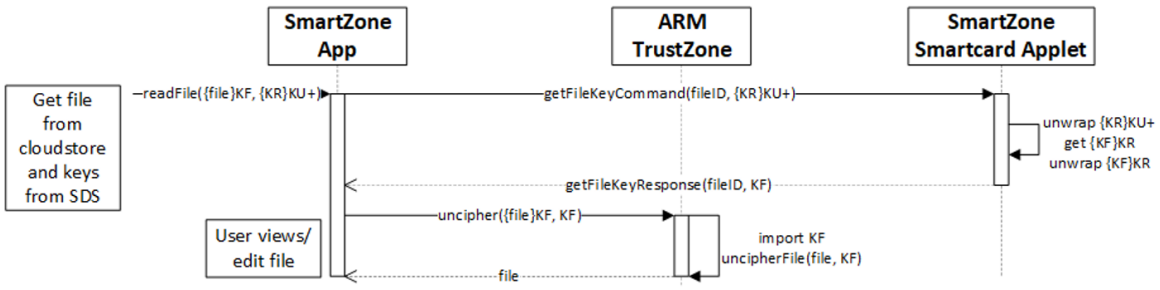


Figure 3.4: SmartZone view file operation.

smart card applet with the fileID and the tuple $\{KR\}KU+$, received from Storekeeper SDS. The SmartZone applet running on the smart card will unwrap the tuple by using the private part of the user key-pair KU in order to obtain KR, and after that, it is going to fetch the wrapped file-key $\{KF\}KR$ from memory, and return the file-key KF, now unwrapped, to the client application. Now all that is left is to securely import KF to the Android Keystore alongside the ciphered file $\{file\}KF$ in order to be unciphered by ARM TrustZone. Finally, the user is able to have access to the unciphered file for viewing or editing purposes.

The received and returned arguments for each operation take into account the Storekeeper API in order for full integration between SmartZone solution and Storekeeper cloud storage service. The full sequence diagram for SmartZone is specified in appendix A.

3.5 Communication

As described, the proposed solution requires a series of messages to be exchanged between the SmartZone application running on the device, and the smart card SmartZone applet, in order for this latter to be able to generate and store new keys. The communication between both

parties is performed in APDU format messages in a command-response style interaction. This format is used to send attributes on the messages which perform operations over the smart card. The operations defined correspond to the User Authentication and File operations. Appendix B specifies in detail the format of the APDUs that correspond to these operations.

During the User Authentication process, two commands are sent consecutively to the smart card being the *ChallengeCommand* and the *AuthenticationCommand* respectively. For these commands, a response is expected by the client application, specifically a *ChallengeResponse* and an *AuthenticationResponse*. When it comes to the File Operations, one single command-response is used, the *GetFileKeyCommand* and *GetFileKeyResponse* messages, being that the only difference is in the fields present in each command that triggers a different behavior on the smart card applet. A third command for getting the smart card status regarding authenticated users and existing file keys also exists and is designated *GetStatusCommand*. On the following Table 3.2 it is possible to analyze each command-response and their data fields.

Commands	**Responses**
ChallengeCommand(String user, String password)	ChallengeResponse(String user, String challenge)
AuthenticationCommand(String user, byte[] challengeResult)	AuthenticationResponse(boolean result)
GetFileKeyCommand(String fileID, byte[] wrappedKey, boolean newFile)	GetFileKeyResponse(String fileID, byte[] fileKey, byte[] wrappedKey)
GetStatusCommand()	GetStatusResponse(boolean init, String message)

Table 3.2: SmartZone commands and responses.

Each command and response is encapsulated inside an APDU Command and APDU Response (Data fields) as specified by ISO/IEC 7816-4 for smart card communication.

3.6 Summary

In this chapter, we described the design of the proposed solution. This solution consists of a highly secure system by merging the functionalities of the ARM TrustZone with smart card technology in order to provide secure protection to file keys used by the cloud storage service Storekeeper.

This solution is divided between two different environments, the user device, and the smart card. In the first we have the client-side application, SmartZone Application, running over the device OS, in this case, Android, which acts as the client in the communication with the smart card. To provide secure storage, this application interacts with the SmartZone applet running on the smart card, which is responsible for all the keys generation and storage. The SmartZone applet running on the smart card is running on the JavaCard Platform, comprising the second component of our solution. To properly design the remaining details of the system, a trust model is proposed. In this trust model assumptions are taken by each component regarding the

underlying system and the communication channels.

To provide a secure communication between the computer and the smart card, several properties are taken into account. The first step is to share a symmetric key between SmartZone Application, running on the device, and the SmartZone applet, running on the smart card. For this, secure key import mechanisms currently available on Android9 (API level 28) are used in order to securely import this key to the Keystore. The second step is to authenticate the user toward the smart card applet, and for this, a challenge-response authentication protocol is used.

When it comes to the functionality, the file operations provided by SmartZone allows for the user to add a new unprotected file to Storekeeper or open a protected file, by having SmartZone deal with all the key generation and file cipher operations with the support of ARM TrustZone for the file cipher and decipher operations. The communication between the client application and the applet running on the smart card uses the APDU command-response format.

The main focus of this solution should be on moving the key generation and management to the smart card due to the high level of security and robustness provided by it, while also use ARM TrustZone to perform cipher and decipher operations with files in question.

Chapter 4

Implementation

The proposed architecture is composed of several components which cooperate to provide the desired features and properties. The implementation, the components, and their communication require technologies matching the required properties. This chapter describes the technologies used to implement the system and the properties obtained from each. It also describes some implementation details regarding the quality properties of the system.

Next, we describe the SmartZone Application that runs on Android OS (Section 4.1). Then we continue this chapter by presenting the implementation for interaction with the smart card (Section 4.2), and also our approach to the applet running on the smart card (Section 4.3). Finally, we finish by providing a brief overview over the different implementation attempts that were made prior to deciding upon the one here presented (Section 4.4).

4.1 SmartZone Application

The SmartZone Application running on the user mobile device was developed for the Android platform. This platform, and consequently OS, was chosen because it is an open-source project and any hardware manufacturer can build a device that runs the Android OS, resulting in an availability of the OS on phones of a large number of manufacturers. Java was the chosen language for the application development using the Android Software Development Kit (SDK). In this kit, there is a variety of tools (debugger, libraries, etc) that were used for the development of the client-side app. When it comes to the execution of the application the native emulator, based on QEMU, was used to run and test the application. The application was developed for the minimum API level of 28 (Android 9 Pie), with build configurations specified in appendix B. Described below is the Android Virtual Device (AVD) configuration user for implementation:

- **AVD Name** : Pixel 3a XL;

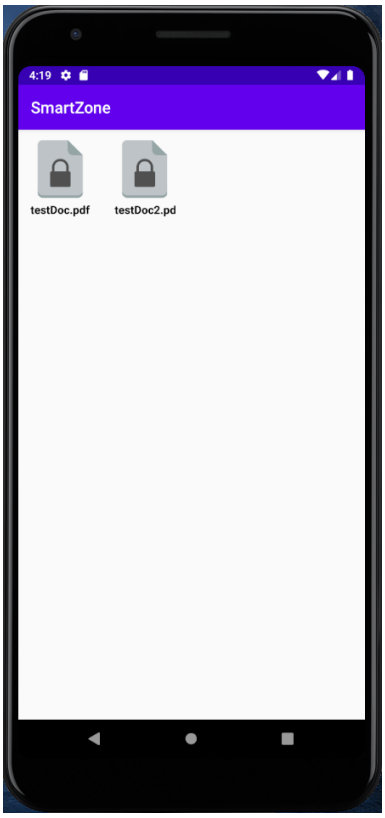


Figure 4.1: SmartZone user interface.

- **API Version** : 28 Android 9.0 (Pie);
- **Screen Resolution** : 1080x2160;
- **CPU Core Count** - 4 (Multi-Core).
- **RAM** : 1536 MB.
- **VM heap** : 256 MB.
- **Internal Storage** : 800 MB.
- **SD Card Storage** : 512 MB.

To perform cryptographic functions the Android security library (androidx.security) was used since it provides strong security that is balanced between noticeable encryption and acceptable performance. This library also promotes the use of the Android Keystore which is relevant as seen in the Section 2.3.3.

When it comes to the application UI, a simple file manager in a grid layout interface was developed in order for the user to easily navigate through the files (See Figure 4.1).

Since the access to ARM TrustZone on Android is very limited, one can use security-oriented APIs provided by Google but can not create his own secure app or use the TrustZone in any

other way. SmartZone uses Android Keystore API (android.security.keystore) to access Keystore functionality. We will be extending its uses and integrating it with ARM Trustzone, since the majority of smartphones and tablets nowadays use ARM processors capable of TrustZone this means that a TEE like ARM TrustZone is deployed in almost every smartphone and tablet. The availability of a trusted execution environment in a system on a chip (SoC) offers an opportunity for Android devices to provide hardware-backed, strong security services to the Android OS, to platform services, and even to third-party apps.

4.2 Smart Card Access

Although smart cards allow several security solutions to be implemented, there is still a burden related with the transportation of the peripherals required to communicate with the smart card. This section proposes the usage of a smart card inside a smartphone in a microSD. This solution brings advantages like flexibility, support, and backward compatibility, but also allows a host device to select and communicate with a security system in the SD card while maintaining its most basic functionality, data storage. SD cards in the microSD format are a widespread format included in most mobile phones allowing to bring some of the advantages of smart card element usage to mobile phones.

For communication between the Android application and the smart card, the Open Mobile API (OMAPI) implementation is used. This mechanism enables an authorized mobile app to communicate with applets within a smart card inside a device. With Android API 28 the Open Mobile API is accessible by the package android.se.omapi.

4.3 Applet Emulation

To implement the desired smart card behavior, and guarantee the successful integration with the client application running on Android, the smart card applet was emulated. The smart card applet was emulated via a Java application while maintaining the APDU command-response message format. The applet behavior was emulated in order for the operations to run in consecutive order. When a command is received from the client application, the smart card applet performs the required operation and returns the response to the client application.

An extra module was added designated *APDUParser*, which is responsible for parsing the APDU commands and responses. When an APDU command is received by the smart card, this module unwraps the SmartZone command from the Data field of the APDU command and passes it along to the applet for execution. When the SmartZone applet is done with the

execution, the response passes through the module in question to be wrapped with the APDU response message. Then the APDU response is sent to the client application running on the device.

To perform cryptographic operations we used the Java library provided by the JCA framework (Java Cryptography Architecture). Regarding symmetric encryption, we use AES cipher and generate 256-bit symmetric encryption keys randomly. For asymmetric encryption, we use 1024-bit RSA asymmetric keys randomly generated.

4.4 Discussion on Different Implementations

We started with the goal of developing three different modules for SmartZone:

1. A **client application**, developed in Java, running on the normal world on top of Android;
2. A **trusted application**, developed in C/C++, running on a open-source Trusted Execution Environment;
3. A **smart card applet**, developed on the JavaCard Platform.

Since third-party developers have no or very limited direct access to ARM TrustZone, the development of a trusted application inside the TEE revealed very complex since a trusted OS needs to be built (or bought) and installed on the device. Then the secure monitor must be implemented and used to communicate with this trusted OS. The use of the available Android Keystore API to take advantage of the ARM TrustZone enabled us to have access to the TEE.

Chapter 5

Evaluation

In order to evaluate our system, we have decided to take into account the state of the art and both the design and implementation details to assess the proposed solution and its provided properties. In this chapter we start by analyzing some of the attributes of the proposed solution comparing it with other solutions existing in the market (Section 5.1), then we present an overview of SmartZone's overall performance regarding each supported operation (Section 5.2). From then on we describe how the requirements defined in Section 1.3 are fulfilled by the proposed solution (Section 5.3). Finally, we discuss some of the security considerations and limitations of our work (Section 5.4).

5.1 Solution Comparison

Due to its nature, it is relevant to assess the proposed system by comparing it with other existing systems. The most significant properties to take into account are the environment where file encryption occurs, file key storage, and management.

Table 5.1 allows for the comparison of the proposed solution with the secure data access control method of cloud storage DFCloud [SKPP12] presented in (Section 2.3.4).

Category	**File Encryption**	**Key Storage**	**Key Management**
DFCloud	Client Side (TEE)	TPM (TEE)	Client Side
SmartZone	*Client Side (TEE)*	*Partially: Storekeeper SDS, Smart card*	*Server Side*

Table 5.1: Comparison of DFCloud with the proposed solution.

The proposed solution presents some similarities and some differences to other solutions, more specifically to DFCloud. When it comes to the file encryption environment both solutions perform the file cipher and decipher operations using ARM TrustZone, hence these operations

are performed in a TEE. This way, an attacker that manages to compromise the local OS will not be able to recover the content of the files.

When it comes to key storage approaches each solution differs. DFCloud uses a TPM emulator inside the secure world to store the file encryption keys, while, SmartZone implements a hybrid key storage solution, where the key KR is securely stored (ciphered with $KU+$) on the Storekeeper SDS and the file key KF never leaves the smart card, providing a high level of security due to the use of the smart card.

Finally, the key management also differs between solutions, since on the Storekeeper design the user is relieved from the responsibility of maintaining file credentials by having them stored on the server-side in a ciphered format. SmartZone applies the same logic, while DFCloud burdens the client with the job of storing and managing all the necessary keys. Also, the keys are generated on the TEE on DFCloud while the main objective of SmartZone was to move the key generation and management to the smart card due to the high level of security and robustness provided by it.

5.2 Solution Performance

Since a reduced performance may affect the usability and utility of a system it is necessary to analyze how the proposed solution performs its most commonly used operations and how said performance scales depending on the size of each file. It is also relevant to analyze this performance when communicating with a smart card.

The evaluation of SmartZone focuses on execution time measurements that were obtained running several benchmarks. We must bear in mind that the SmartZone application was executed on the AVD emulator running on top of a Macbook Pro with an Intel®Core™i7-6700HQ 2,6GHz CPU and 8GB of RAM, running MacOS Catalina (v.10.15.7). Each benchmark measures an operation provided by SmartZone, such operations are the user authentication, new file, and read file operations. The file size parameter was used with different data unit sizes of 10KB, 100KB, and 1MB. For each file size, a category was attributed, being that the 10KB, 100KB, and 1MB files were attributed to the Small (S), Medium (M), and Large (L) categories respectively (See Table 5.2). Due to various factors, such as operating system scheduling and system load, the same operation's time span may vary.

The first benchmark consists of measuring the total time used by the application to perform the user authentication operation. These calls consist of sending user credentials to the smart card, generate a challenge and authenticate the user. Also, the time to generate the user keypair is taken into account in this benchmark. For the entire operation, the execution time

Category	Size (KB)
S	10
M	100
L	1000 (1MB)

Table 5.2: File size categories.

was 1330ms, which is divided between the Challenge Command and Response execution which took 437ms, and the Authentication Command and Response which took 893ms. The higher execution time for the second command can be attributed to the generation of the user key-pair on the smart card since is the Authentication Command that triggers this operation. See Table 5.3.

Operations	Values (ms)
Challenge Command	437
Authentication Command	893

Table 5.3: Execution times in milliseconds for User Authentication operation.

For the benchmarks related to the file operations, we can divide the measurements in two different sub-benchmarks.

On the first file operation benchmark, the execution time for the new file operation was measured. This operation is performed when a new unprotected file is added to SmartZone, as described on Section 3.4.2. This benchmark is divided in two steps: GetFileKey Command and Response interaction between the SmartZone Application and the smart card; File cipher operation performed inside the ARM TrustZone. On the Figure 5.1 note that the GetFileKey Command and Response interaction maintains a similar execution time of 23ms, 19ms and 29ms for S, M and L files respectively. On the other hand, for an increase on the file size is is possible to verify an increase on the execution time of the file cipher operation. On the cipher operation for the S file an execution time of 75ms was measured, for the M file a 92ms execution time and finally the L file cipher operation resulted on a 308ms execution time.

Now on the second file operation benchmark, the execution time for the read file operation was measured. This operation is performed when a protected file already exists in SmartZone and the user wants to open the respective file, as described in Section 3.4.2. This benchmark is divided into three steps: GetFileKey Command and Response interaction between the SmartZone Application and the smart card; File decipher operation followed by the cipher operation, both executed inside the ARM TrustZone. On the Figure 5.2 note that the GetFileKey Command and Response interaction maintains a similar execution time independently from the file

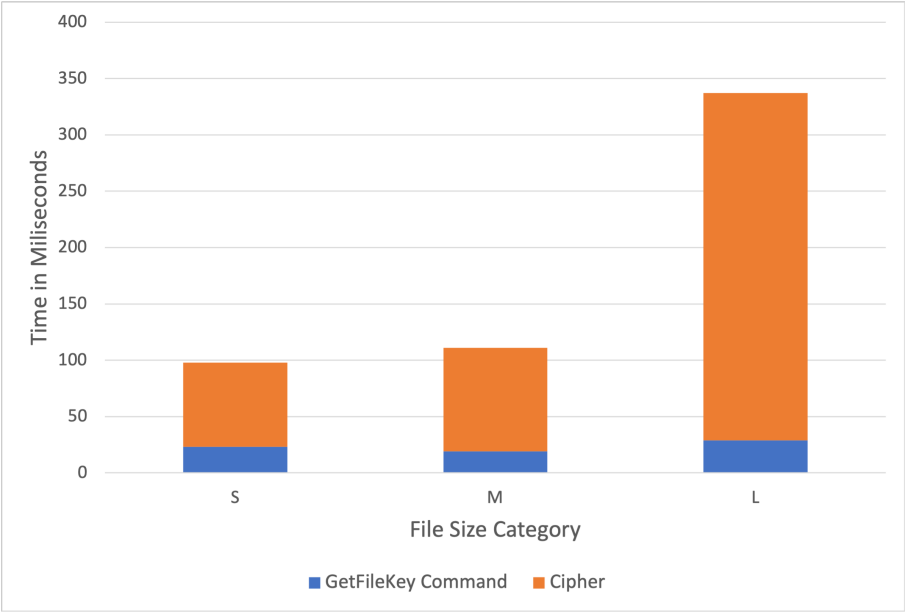


Figure 5.1: Performance of new file operation.

size, as on the previous benchmark. On this benchmark, the execution time measured for these interactions were 33ms, 28ms, and 23ms for the S, M, and L files respectively. Now when it comes to the uncipher operation execution time it is possible to verify that it takes considerably more time than the cipher operation. The uncipher operation execution time for the S file was 67ms, on the M file was 205ms and on the L file it was 1533ms, increasing as the file size increases. The cipher operation execution times on this benchmark are similar to the previous benchmark, being for the S file a 69ms execution time, for the M file a 91ms time and for the L file a 340ms execution time.

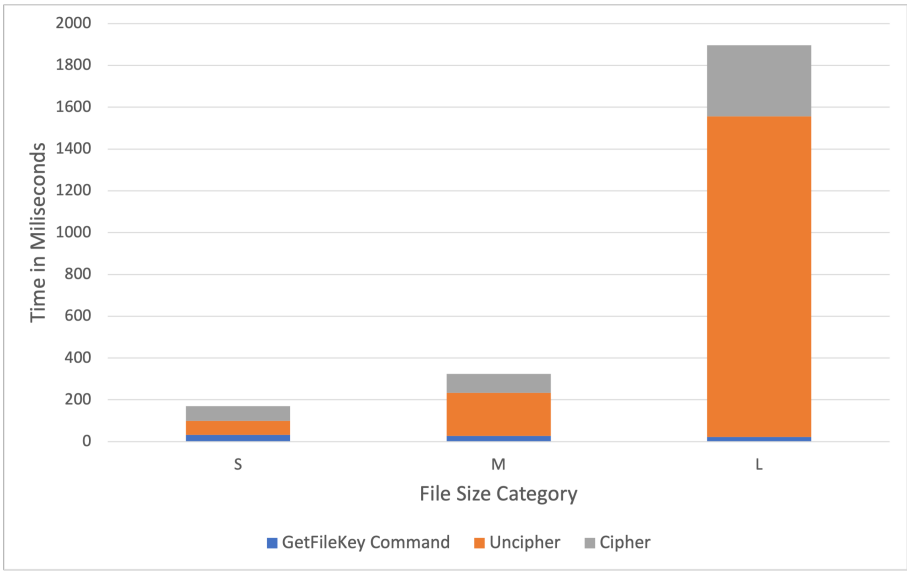


Figure 5.2: Performance of read file operation.

As depicted in Figures 5.1 and 5.2, in both operations, the proposed solution performance has a significant increase of execution time, especially on the file cipher and decipher operations. Such increase was expected, although since the smart card applet was emulated and no benchmark tests were performed on a real smart card hardware it is difficult to predict the impact of the smart card applet execution on the solution.

5.3 Objectives Fulfilment

In Section 1.3 the objectives of the proposed solution are defined and throughout the document both the design and implementation details of the solution have been introduced. This section presents how the properties and behavior of the solution match the objectives outlined initially.

1 - Allow for greater isolation between the operating system and the critical operations performed by Storekeeper

By merging the functionalities of the ARM TrustZone, which provides a much more reliable platform and user interaction than 'normal world' applications, with a smart card embedded on a microSD card it is possible to provide the highly secure cloud storage aggregator service. The critical operations, like file cipher and decipher operations and the key management, are now performed on ARM TrustZone TEE and on the smart card applet, respectively.

2 - The user must be able to use the system independently of his location

By design, the proposed solution is based on a Smart Card, such as those presented in Section 2.2. These smart cards have the property of being easily transported. The proposed solution is easily transported and can be utilized by the user in any location without requiring transportation of its data into any insecure devices or synchronization through other systems.

3 - The performance overhead must be low enough to not critically impact the system usability

There were no specific benchmark tests performed, however, from the results of the execution of the solution on the emulator the performance loss does not affect the user experience.

4 - The overall system and the authentication operations must be "user-friendly" and easy to the user

As can be seen, by the solution UI and operation flow, the user is abstracted from the authentication and file operations. The solution adopts a "file explorer" style UI, with clear icons distinguishing between protected and unprotected files, for easy user comprehension.

5.4 Security Considerations and Limitations

The attack surface of our work is mainly divided by the ARM TrustZone TEE and the smart card. As already detailed in Section 2.3.1 there are a series of considerations when it comes to the ARM TrustZone security.

The main concern is the possibility of abusing the SMC instruction to perform a loop of world switches, resulting in a denial of service by locking the normal world OS. The shared memory region, which allows to read and write data between worlds, also presents a concern since a possible attack is writing wrong information into the secure world through this region, although there are hashing functions in the secure world that can be used to verify the veracity of the data provided by the normal world. Due to the support of secure memory provided by the ARM TrustZone, we can safely say that the memory allocated for the secure world is protected and therefore cannot be accessed by the normal world in any way.

The normal world is not protected, so it can be compromised. Since the normal world OS is vulnerable an attacker can spoof the world switch and return fake information to the user. As such we can not provide confidentiality of the files when the user opens the file on the device to see its contents, since the ability for the secure world to display information to the user is very limited.

Finally, any kind of system on chip attacks to the ARM architecture or on the smart card are out of the scope of this paper and are not considered. We assume that the hardware is correctly implemented and the TrustZone mechanisms cannot be circumvented.

5.5 Conclusion

In this chapter, we start by analyzing some of the attributes of the proposed solution and comparing it with other solutions that already exist. Then we presented the overall performance of the system regarding the main operations offered by SmartZone. We then describe how the requirements defined in Section 1.3 are fulfilled by the proposed solution. Finally, we do a discussion on some security considerations and limitations of the system.

Chapter 6

Conclusions

Current cloud storage aggregator services have limited security since they fail to provide end-to-end privacy. In order for mobile devices to manage identities, several solutions have been proposed and designed. ARM TrustZone provides secure computational environments, and smart cards are known for providing a very strong user authentication and key management functionality. By marrying trusted execution environments with smart card technology it is possible to obtain a very strong user authentication and key management functionalities. This solution aims to mitigate the key management and user authentication problem present on the distributed cloud storage service, Storekeeper. In addition to this, several tests should be performed in order to evaluate the performance and security impact of this work.

6.1 Achievements

This thesis presented the design, implementation, and evaluation of SmartZone, a distributed solution, consisting of an enabled ARM TrustZone mobile application and a smart card applet. The target application is the privacy-preserving cloud aggregation service Storekeeper. SmartZone addresses the problems with dealing with highly sensitive data on a vulnerable mobile device that is being used as a client of a distributed cloud storage system. Using smart card technology for key management and storage provides tamper-proof storage and secure computation capabilities. ARM TrustZone provides clear isolation from a possibly compromised normal world OS and guarantees that sensitive data is only handled in the trusted execution environment, being used for file encryption and decryption operations it provides security advantages when compared to using the device storage and normal world environment.

6.2 Future Work

For future work, there is still some work that can be produced to either augment its functionalities or to enhance its security. We briefly discuss some possible directions:

Custom Trusted Application

The work done was considering two modules, the SmartZone Application running on the normal world and the SmartZone applet running on the smart card. The communication between these two parties is performed by the normal world environment, hence the need for an extra shared key KS in order to provide confidentiality. It may be of interest to develop a third module consisting of a SmartZone Trusted Application that is running on an open-source TEE in order to provide a greater isolation from the normal world. This way the smart card applet interacts solely with the trusted application running inside the ARM TrustZone secure environment.

Advanced Implementation

When it comes to the implementation, it is of interest to implement the SmartZone application on a physical mobile device running Android OS. Also when it comes to the SmartZone smart card applet, the use of a real Java Card enabled smart card on a microSD format raises interesting issues related to the integration between these technologies. A more advanced implementation using real hardware also results in more reliable benchmark results than using emulated devices.

Integration with Storekeeper

Finally integration with the target of this solution the cloud aggregation service Storekeeper using its respective API. Since SmartZone presents itself as an enhance on the already implemented Storekeeper, the integration of the Storekeeper file operations with the client-side operations performed by SmartZone provides an overview of the security and reliability enhancements provided by this latter. Also by integrating both solutions is possible to measure the overall benchmark of this new hybrid solution and more specifically measure the overhead causes by SmartZone on the Storekeeper performance.

Bibliography

[And] Android. Android Secure.

[App] Apple. iOS.

[ARM09] ARM. ARM Security Technology. Building a Secure System using TrustZone Technology. *ARM white paper*, page 108, 2009.

[BGK17] Stefan Brenner, David Goltzsche, and Rüdiger Kapitza. TrApps: Secure compartments in the evil cloud. *Proceedings of the 1st International Workshop on Security and Dependability of Multi-Domain Infrastructures, XDOM0 2017 - Co-located with European Conference on Computer Systems, EuroSys 2017*, 2017.

[Box] Boxcryptor. Boxcryptor.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[CDRP14] Tim Cooijmans, Joeri D. De Ruiter, and Erik Poll. Analysis of secure key storage solutions on android. In *Proceedings of the ACM Conference on Computer and Communications Security*, volume 2014-Novem, pages 11–20. ACM, 2014.

[DW09] Kurt Dietrich and Johannes Winter. Implementation aspects of mobile and embedded trusted computing. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5471 LNCS, pages 29–44. Springer, 2009.

[EKA14] Jan Erik Ekberg, Kari Kostiainen, and N. Asokan. The untapped potential of trusted execution environments on mobile devices. *IEEE Security and Privacy*, 12(4):29–37, 2014.

[FZFF10] Ariel J Feldman, William P Zeller, Michael J Freedman, and Edward W Felten. {SPORC}: Group Collaboration using Untrusted Cloud Resources. In *Proceedings*

of the 9th USENIX Symposium on Operating Systems Design and Implementation, pages 337–350, 10 2010.

[Glo] GlobalPlatform. Card Specification.

[Goo] Google. Android.

[Jam12] Brandon James. Smart Card Based User Authentication. (June 2010), 2012.

[KK14] Max Kellner and Fredrik Kvant. A Development Environment for ARM TrustZone with GlobalPlatform Support. (June):46, 2014.

[Lie17] Ingo Liersch. Id cards and passports. In *Smart Cards, Tokens, Security and Applications*, pages 387–412. Springer, 2017.

[MA17] Konstantinos Markantonakis and Raja Naeem Akram. Multi-application smart card platforms and operating systems. In *Smart Cards, Tokens, Security and Applications*, pages 59–92. Springer, 2017.

[May17] Keith Mayes. An introduction to smart cards. In *Smart Cards, Tokens, Security and Applications*, pages 1–29. Springer, 2017.

[MBG19] Muhammad Asim Mukhtar, Muhammad Khurram Bhatti, and Guy Gogniat. Architectures for Security: A comparative analysis of hardware security features in Intel SGX and ARM TrustZone. *2019 2nd International Conference on Communication, Computing and Digital Systems, C-CODE 2019*, pages 299–304, 2019.

[MC12] Luis A. Maia and Manuel E. Correia. Java JCA/JCE programming in Android with SD smart cards. *Iberian Conference on Information Systems and Technologies, CISTI*, pages 1–6, 2012.

[ME17] Keith Mayes and Tim Evans. Smart cards for mobile communications. In *Smart Cards, Tokens, Security and Applications*, pages 85–113. Springer, 2017.

[Mic] Microsoft. Windows 10.

[MM17] Konstantinos Markantonakis and David Main. Smart cards for Banking and Finance. In *Smart Cards, Tokens, Security and Applications*, pages 129–153. Springer, 2017.

[Mul] Multos. The Multos Technology.

[Odr] Odrive. Odrive.

[Oraa] Oracle. Java Card Platform Architecture.

[Orab] Oracle. Java Card Technology.

[Ora19] Oracle. Java Card: The Open Application Platform for Secure Elements, 2019.

[PASC16] Sancha Pereira, Andre Alves, Nuno Santos, and Ricardo Chaves. Storekeeper: A Security-Enhanced Cloud Storage Aggregation Service. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, (May):111–120, 2016.

[PS19a] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys*, 51(6), 2019.

[PS19b] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys*, 51(6), 2019.

[RE10] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook Fourth Edition*. John Wiley & Sons, Ltd, fourth edi edition, 2010.

[SAB15] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015*, 1:57–64, 2015.

[SD 14] SD Association. Activating New Mobile Services and Business Models with smartSD Memory cards. 2014.

[SD 19] SD Association. SD Express and microSD Express Memory Cards : The Best Choice for Your Future Product Designs. 2019.

[SKPP12] Jaebok Shin, Yungu Kim, Wooram Park, and Chanik Park. DFCloud: A TPM-based secure data access control method of cloud storage in mobile devices. *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, pages 551–556, 2012.

[SRSW14] N Santos, H Raj, S Saroiu, and A Wolman. Using {ARM TrustZone} to build a trusted language runtime for mobile applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 67–80, 2014.

[Tar09] Talha Tariq. Extending Secure Execution Environments Beyond the TPM (An Architecture for TPM & SmartCard Co-operative Model). Technical report, Department of Mathematics, Royal Holloway, University of London, 2 2009.

[Tun17] Michael Tunstall. Smart card security. In *Smart Cards, Tokens, Security and Applications*, pages 217–251. Springer, 2017.

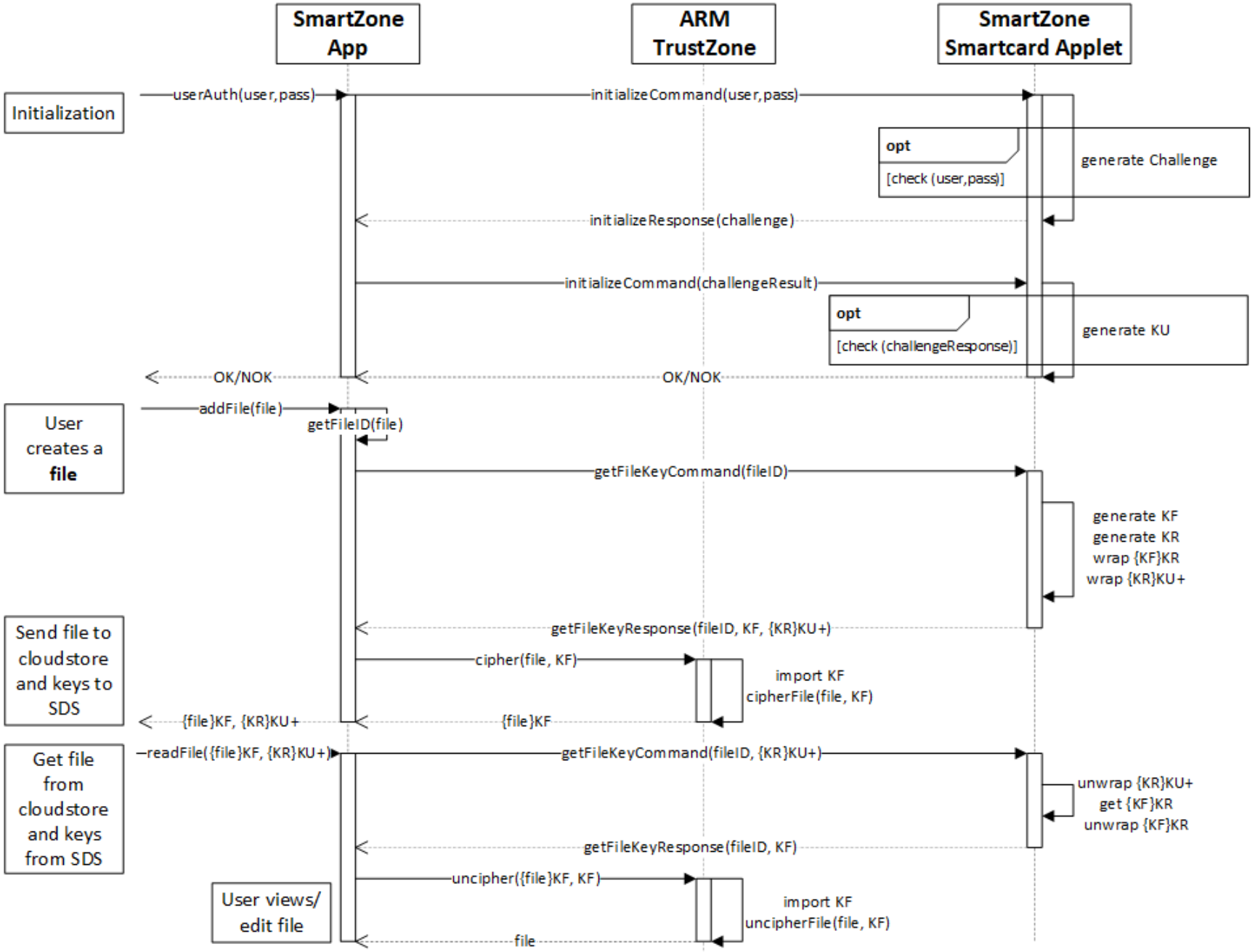
[VSV12] Michael Vrable, Stefan Savage, and Geoffrey M Voelker. BlueSky: a cloud-backed file system for the enterprise. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012.

[VV00] Jean-Jacques Vandewalle and Eric Vétillard. Developing Smart Card-Based Applications Using Java Card. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications*, pages 105–124, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Yal18] Sileshi Demesie Yalew. *Mobile Device Security with ARM TrustZone*. PhD thesis, KTH Royal Institute of Technology, 2018.

Appendix A

SmartZone Sequence Diagram


SmartZone App
ARM TrustZone
SmartZone Smartcard Applet
Initialization
userAuth(user,pass)
initializeCommand(user,pass)
opt
[check (user,pass)]
generate Challenge
initializeResponse(challenge)
initializeCommand(challengeResult)
opt
[check (challengeResponse)]
generate KU
OK/NOK
OK/NOK
User creates a file
addFile(file)
getFileID(file)
getFileKeyCommand(fileID)
generate KF
generate KR
wrap {KF}KR
wrap {KR}KU+
getFileKeyResponse(fileID, KF, {KR}KU+)
Send file to cloudstore and keys to SDS
cipher(file, KF)
import KF
cipherFile(file, KF)
{file}KF, {KR}KU+
{file}KF
Get file from cloudstore and keys from SDS
readFile({file}KF, {KR}KU+)
getFileKeyCommand(fileID, {KR}KU+)
unwrap {KR}KU+
get {KF}KR
unwrap {KF}KR
getFileKeyResponse(fileID, KF)
uncipher({file}KF, KF)
User views/ edit file
import KF
uncipherFile(file, KF)
file

Appendix B

Gradle Build configuration

The Gradle Build configuration used for the implementation, with details regarding compilation and minimum SDK version and also all the dependencies used:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

    defaultConfig {
        applicationId "pt.ulisboa.tecnico.smartzone"
        minSdkVersion 28
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
            'proguard-rules.pro'
```

```
        }
    }
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'androidx.recyclerview:recyclerview:1.2.0'
    implementation 'androidx.security:security-crypto:1.1.0-alpha03'
    implementation 'androidx.swiperefreshlayout:swiperefreshlayout:1.1.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

}
```