SmartZone: Enhancing the security of TrustZone with SmartCards

Afonso Maria Viegas Bota Carreira Caetano
Instituto Superior Técnico, Universidade de Lisboa

***Abstract*—Current cloud storage services have limited security and require users to grant full access privileges to individual cloud storage accounts. With recent advances the need for mobile devices that can manage identities and provide convenient access authorisation 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 cards embedded on a microSD card, and applying these functionalities to the already developed cloud storage aggregator service Storekeeper [1]. With this, it will be possible to provide a 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.

I. 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. Cloud aggregator systems provide users 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 authorisation 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 [2].

Storekeeper [1] 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.

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 [3]. The target application for these functionalities is the distributed cloud storage service Storekeeper that has already been developed [1]. 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.

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.

II. 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.

A. 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 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 file size, operations regarding cipher and decipher of files must be performed on the Trusted Execution Environment (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 a 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 it contents to the user. After that, the file must be protected again.

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

B. 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 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 on 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 a 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 proper implemented firewall between applications.
- Only the authenticated user has access to he 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.

C. Architecture

The proposed architecture is divided among two environments, the device side and the smart card side. As can be seen on Figure 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 on the ”normal world” we have the SmartZone Application component. This

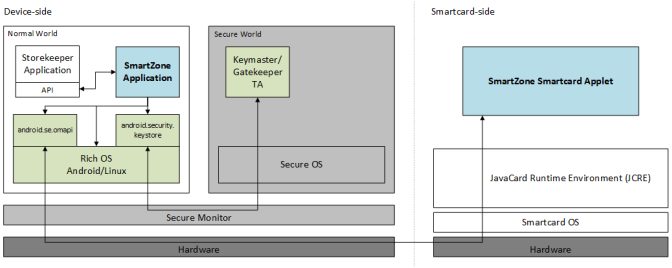

Fig. 1: SmartZone architecture.

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

Fig. 2: Keys Exposure.

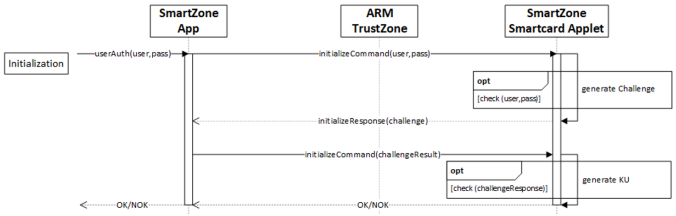

Fig. 3: SmartZone user authentication.

component interacts with *AndroidKeystore* via the Android Framework 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 behaviours 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 on Section III-B.

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 to import 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 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 2, respectively:

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

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

D. 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.

E. 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.

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.

F. 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

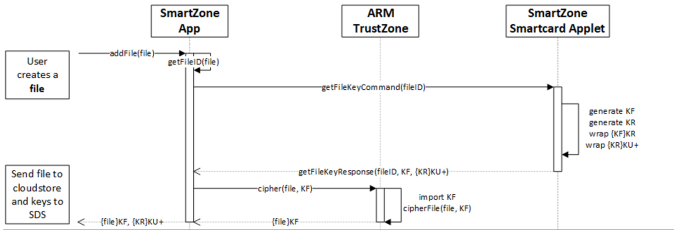

Fig. 4: SmartZone new file operation.

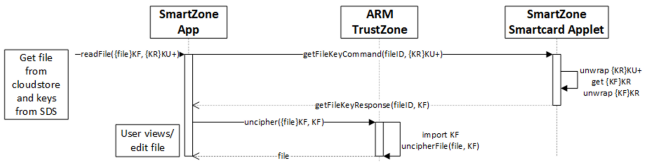

Fig. 5: SmartZone view file operation.

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 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 it 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 operations takes account the Storekeeper API in order for full integration between SmartZone solution and Storekeeper cloud storage service.

G. 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 Application Protocol Data Unit (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 a *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 behaviour on the smart card applet. A third command for getting the smart card status regarding authenticated users and existing file keys also exist and is designated *GetStatusCommand*.

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

III. Implementation

The proposed architecture is composed by several components which cooperate to provide the desired features and properties. The implementation, the components and their communication requires 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 quality properties of the system.

A. 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 on 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). Described below is the Android Virtual Device (AVD) configuration user for implementation:

- **AVD Name** : Pixel 3a XL;
- **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 make balances between noticeable encryption and acceptable performance. This library also promotes the use of the Android Keystore.

Since the access to ARM TrustZone on Android is very limited, one can use security-oriented API's 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 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.

B. 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 smart phone in a microSD. This solution 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 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.

C. Applet Emulation

To implement the desired smart card behaviour, and guarantee the sucessfull 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-respose message format. The applet behaviour was emulated in order for the operations to run in a consecutively order. When a command is received from the client application, the smart card applet performs the required operation and return the response to the client application.

An extra module was added designated *APDUParser*, which is responsible for parsing of 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.

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*

Fig. 6: Comparison of DFCloud with proposed solution.

IV. Evalutation

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.

A. 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 6 allows for the the comparison of the proposed solution with the secure data access control method of cloud storage DFCloud [4].

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 differ. DFCloud uses a TPM emulator inside the secure world to storage the file encryption keys, while, SmartZone implements a hybrid key storage solution, where the key *KR* is securely stores (ciphered with *KU+*) on the Storekeeper SDS and the 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 relived from the

Operations	Values (ms)
Challenge Command	437
Authentication Command	893

Fig. 7: Execution times in milliseconds for User Authentication operation.

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.

B. Solution Performance

Since a reduced performance may affect the usability and utility of a system it is necessary to analyse 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 analyse 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. 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 timed 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 key-pair is taken into account in this benchmark. For the entire operation, the execution time 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 7.

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 II-F. 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 8 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.

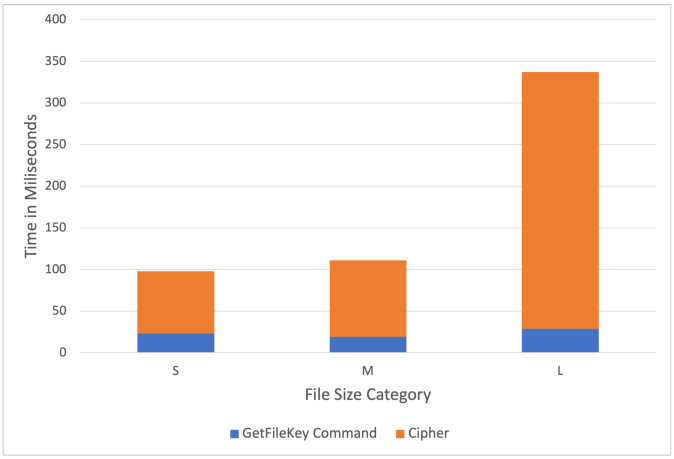

Fig. 8: Performance of new file operation.

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 on Section II-F. This benchmark is divided in 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 9 note that the GetFileKey Command and Response interaction maintains a similar execution time independently from the file 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.

As depicted in Figures 8 and 9, in both operations the

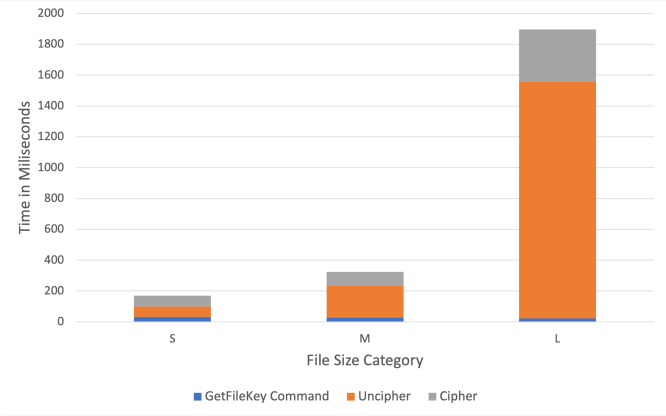

Fig. 9: Performance of read file operation.

proposed solution performance has a significant increase of execution time, specially 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.

C. Objectives Fulfilment

This subsection presents how the properties and behaviour of the solution match the objectives outlined initially.

1 - Allow for a 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.

D. 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 there are a series of considerations when it comes 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 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.

V. Related Work

While cloud storage has a lot of potential, unless the issues of confidentiality and integrity are addressed many users will be reluctant 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.

A. Cloud Storage

Cloud providers put a lot of effort in 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.

Storekeeper is a distributed system compromised 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 which are provided by the user and stored on the SDS, by having the files stored on these stores it results on the files not being directly stored on the SDS.

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. 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.

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 consider cloud providers to be potentially malicious with respect to violation of data confidentiality, and honest but curious regarding to passively listening to the 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-channels 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 the file encryption keys of a file or the access token, violating the end-to-end confidentiality and authentication premises. Integrity of data and meta-data is also not guaranteed by Storekeeper when intentional modifications are performed by the cloud aggregator or cloud providers [1].

B. 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 [5]. A smart card is a portable, tamper-resistant platform that provide secure storage of confidential data and can protected against unauthorized access and manipulation. It is used in many different industries that come in different sizes and forms such as SIM cards, identification card, USB sticks, etc [6].

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. 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.

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.

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 is 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.

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. Also the introduction of random delays at different points in the algorithm execution, although does not prevent an attack, but can be used to increase the time required to attack. The use of 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 [7].

SmartSD cards [3] 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 [8] 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 [9]. 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 stable increase in SD cards read and write speeds and storage capacity throughout the years.

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 [10], Multos [11] and GlobalPlatform [12] are the most widely utilised smart card platforms [13].

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 on more independence between objects and their interaction using well defined interfaces [14]. 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 be ran into a Java Card virtual machine in the developer's computer. 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 [15].

C. 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.

TrustZone 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"* [16].

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 in terms of memory, to this environment, it gravitates to be small in size and only contain the bare essentials. Trusted applications can come pre-installed from the factory or they can be installed during the lifetime of the device. The normal world runs the traditional OS, the Rich OS, which the user interacts with. Common examples include Windows 10 [17], iOS [18] and Android [19]. 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.

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. This critical links can be broadly categorized in three categories [20]: **Type 1**, where the attacker makes link to security critical application through privileged software. **Type 2**, attacker links to security critical application through micro-architectural events of hardware, and finally **Type 3** where the attacker makes link to security critical application through directly probing hardware.

Android OS [19] 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. 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 [9] [21].

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 applications 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.

Nowadays anyone has access to wireless networks very easily and have access do 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 have arisen. Cloud storage services is 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) [4]. 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 do prevent credential or data leakage.

VI. 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 environment 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.

This documented presented the design, implementation, and evaluation of SmartZone, a distributed solution, consisting on 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.

References

[1] S. Pereira, A. Alves, N. Santos, and R. Chaves, "Storekeeper: A Security-Enhanced Cloud Storage Aggregation Service," *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, no. May, pp. 111–120, 2016.

[2] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM Computing Surveys*, vol. 51, no. 6, 2019.

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

[4] J. Shin, Y. Kim, W. Park, and C. 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*, pp. 551–556, 2012.

[5] K. Mayes, "An introduction to smart cards," in *Smart Cards, Tokens, Security and Applications*. Springer, 2017, pp. 1–29.

[6] T. Tariq, "Extending Secure Execution Environments Beyond the TPM (An Architecture for TPM & SmartCard Co-operative Model)," Department of Mathematics, Royal Holloway, University of London, Tech. Rep., 2 2009.

[7] W. Rankl and W. Effing, *Smart Card Handbook Fourth Edition*, fourth edi ed. John Wiley & Sons, Ltd, 2010.

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

[9] L. A. Maia and M. E. Correia, "Java JCA/JCE programming in Android with SD smart cards," *Iberian Conference on Information Systems and Technologies, CISTI*, pp. 1–6, 2012.

[10] Oracle, "Java Card Technology." [Online]. Available: https://www.oracle.com/java/technologies/java-card-tech.html

[11] Multos, "The Multos Technology." [Online]. Available: https://www.multos.com

[12] GlobalPlatform, "Card Specification." [Online]. Available: https://globalplatform.org

[13] K. Markantonakis and R. N. Akram, "Multi-application smart card platforms and operating systems," in *Smart Cards, Tokens, Security and Applications*. Springer, 2017, pp. 59–92.

[14] J.-J. Vandewalle and E. Vétillard, "Developing Smart Card-Based Applications Using Java Card," in *Smart Card Research and Applications*, J.-J. Quisquater and B. Schneier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 105–124.

[15] Oracle, "Java Card: The Open Application Platform for Secure Elements," pp. 1–5, 2019. [Online]. Available: https://www.oracle.com/technetwork/java/javacard/overview/java-card-data-sheet-19-01-07-5250140.pdf

[16] 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*, 2014, pp. 67–80.

[17] Microsoft, "Windows 10." [Online]. Available: https://www.microsoft.com/en-us/windows

[18] Apple, "iOS." [Online]. Available: https://www.apple.com/ios/ios-13/

[19] Google, "Android." [Online]. Available: https://www.android.com/

[20] M. A. Mukhtar, M. K. Bhatti, and G. 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*, pp. 299–304, 2019.

[21] Android, "Android Secure." [Online]. Available: https://source.android.com/security