2016 11th International Conference on Availability, Reliability and Security

Ensuring the Authenticity and Fidelity of Captured Photos Using Trusted Execution and Mobile Application Licensing Capabilities

Kostantinos Papadamou, Riginos Samaras, Michael Sirivianos
Cyprus University of Technology
Limassol, Cyprus
{*ck.papadamou, ri.samaras*}*@edu.cut.ac.cy, michael.sirivianos@cut.ac.cy*

***Abstract*—Mobile devices, which users habitually carry along, have become the main data gateway for the majority of the online services. Any device is able to collect at any time various types of data through its sensors. At the same time, modern identification techniques ask users to send photos of their ID documentation in order to be verified by an online service. Those photos are captured by the device's camera and are considered extremely sensitive. They must be secured and establish that they will not be modified. This paper describes a security framework that preserves the authenticity of a captured photo and ensures that it remains intact while transferred to a remote server. The key insight is to use a background service that is tied to the photo-capturing application and uses secure key storing and cryptographic computation capabilities offered by the Trusted Execution Environment (TEE) of commodity Android devices. At the same time, we leverage Playstore's Licencing Verification Library (LVL) to remotely attest the authenticity of the photo-capturing application at registration time. We have implemented our framework as an Android application on a Nexus 5X, which is powered by a Qualcomm processor with ARM TrustZone Technology. The evaluation of our prototype implementation demonstrates the efficacy of the proposed framework in terms of performance overhead and usability.**

***Keywords*-Cryptography; Privacy; Trusted Computing; Authenticity; Fidelity;**

I. Introduction

In the last decade, Android devices have become the most popular gadget used in the daily life. According to YAHOO! Tech in 2014, 52% of U.S. smartphones' owners used a handset that runs an Android operating system [2]. Except from an entertainment product, an Android device is also a channel and a storage of sensitive information. At the same time, mobile devices are equipped with a variety of sensors and have become the eyes and ears of a lot of applications, including Internet of Things (IoT) applications, by providing their sensing information. Therefore, the authenticity and fidelity of such information must be ensured while staying in memory or when sent to a remote client or server.

Data authenticity and fidelity is crucial for user identification techniques that employ acquisition of physical documentation over the Web. Such techniques request from users to capture a photo of their identity documentation through the camera of their mobile device. The authenticity and fidelity of such photos must be preserved and the service performing the identity verification must be able to determine whether the received photo is authentic. This is because a malicious user may want to upload a modified identity documentation photo pretending to be someone else or use it for various other purposes.

Any digital data can be assumed to be authentic if we can prove that it has not been corrupted or modified after their creation. Especially in the area of user identification, where a strict sense of data authenticity is applied, any processing means corruption. The data is considered authentic only if is the outcome of the acquisition process of a real-world document.

In this paper we focus on ensuring the authenticity and fidelity of captured photos for user identification techniques and we introduce a security framework for solving the aforementioned problems. We present the design and a prototype implementation of our framework that exploits the capabilities of the Trusted Execution Environment (TEE) offered in commodity mobile devices. We use the TEE to store cryptographic keys and to perform cryptographic operations in the trusted environment of the mobile device. Our primary goal is to empower the remote service with the ability to determine about the authenticity of the received photo and to decide whether to accept it or not. The key insight is to use a background service that is tied to the application and performs all the sensitive operations.

Performing cryptographic operations, such as RSA signature in our case, means that you have the private key and you share a public key with the verifier of the signature. At registration time our framework creates an RSA key-pair using the Android KeyStore system and sends the public key to the remote server. Subsequently when a photo has been captured, our background service intervenes and reads the source photo. Once the background service has done so, it computes and signs the hash of the photo with the private key that resides in the TEE. When the user tries to submit the photo, the application allows the background service to handle the submission. The background service takes the photo that the user tries to submit, computes a hash of it and compares it with the hash computed from before. If the two hashes match then the background service submits it to the server along with the signature. The server uses the public key for the corresponding user to verify the signature and to decide whether to store or discard the photo.

In order to prevent unauthorised use of our application and ensure that the server has received the correct public key, for each user, we use the Licensing Verification Library (LVL) offered by Google [5]. When a user tries

978-1-5090-0990-9/16 $31.00 © 2016 IEEE
DOI 10.1109/ARES.2016.83

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

to register a new account, we perform the License Verification. To this end, we have modified the LVL in a way that the response from the Google Play Licensing server is forwarded to the server so we can perform server-side license validation for the application. Thus, the server can ensure that the public key received from the client application is the correct one. In our approach, an application is considered licensed only if it has been downloaded from the Google Play Store and it is a compliant implementation of our framework. In our approach we assume that the user does not have root privileges on her device and no other application that runs on the same device can access the internal storage of our application or use the private keys stored by our application in the TEE. The application may also run securely on rooted devices under some conditions that we will describe in the discussion section.

We have implemented a prototype of our framework for Android on a Nexus 5X, which is powered by a Qualcomm processor with the ARM TrustZone Technology. Our custom registration and photo capture and submission processes have a reasonable execution time without affecting the usability of the application. A short evaluation shows that our approach is feasible and can ensure authenticity and fidelity of captured photos with an acceptable performance overhead.

The rest of the paper is organised as follows: Section II describes background information on the technologies we are using. Section III describes the related work in the area of data fidelity and authenticity. Section IV provides an overview of our proposed framework while Section V we define our trust and threat model. A detailed description of the design and the prototype implementation is provided in Sections VI and VII accordingly. The evaluation of our framework is provided in Section VIII. In Section IX we discuss our future work and offer recommendations to devices manufacturers that can make our framework more secure. Last, Section X summarizes our conclusions.

II. Background

A. Trusted Execution Environment

The Trusted Execution Environment (TEE) is a secure area on the main processor of a smart device [10]. The TEE offers isolated secure execution of authorized security software and can ensure that sensitive data is stored, processed and protected in a trusted environment isolated from the rich OS environment. Furthermore, the TEE is embedded in the processor of the device during manufacture and executes trusted applications built in by device vendors as well as trusted applications installed by users.

B. ARM TrustZone Technology

ARM TrustZone Technology is a System On Chip (SoC) approach to secure the whole system through CPU [1]. The SoC approach can build a root of trust on mobile devices. A software that is executed in CPU is either in a "trusted world" or a "non-secure world". To switch between those two worlds, TrustZone either uses microcontrollers or application processors. Every application that resides in a "trusted world" has its own secure memory, transactions on a bus, and interrupts living in the SoC.

1) Cortex-A processor: This ARM processor is creating a Trusted Execution Environment by securing the boot and the OS procedures of the device. This TEE is used for authentication, cryptographic operations and secure memory access.

2) Cortex-M processor: This ARM processor has the same functionality as the Cortex-A but it can switch between the two worlds by using a hardware based approach, which is faster and with less power cost.

As described in GlobalPlatform [10] and explained by Brian McGillion et al. [9], a device's OS can be separated in two distinct environments:

a) The Rich Execution Environment (REE), where all regular applications are executed. Those application are called client applications and do not require secure execution.

b) The Trusted Execution Environment (TEE), as described above is a trusted environment in the processor where all the critical operations of the device is executed. Such critical operations is Cryptographic key generation and RSA encryption and decryption.

C. Android KeyStore System

Cryptographic operations that reside in the TEE can be utilized by using the Android KeyStore System [3]. Android KeyStore lets you store and protect your cryptographic keys in a secure trusted environment. Once the keys are in the KeyStore, they can be used but the key material remains non-exportable. Those keys can be used for any cryptographic operation like signing and verifying data and can be retrieved using an alias that is set up during the creation of a key. The generated alias are tied to the application and cannot be used by any other application running on the device neither can see them. This allows the applications to take an authorization on the key that cannot be changed once those keys have been created. The cryptographic keys are available in plain text only inside the TEE where the operations are taking place. This ensures the security of a cryptographic operation.

D. Google Play Licensing Service

Google Play Licensing is a network-based service offered by Google to Android Developers [5]. It allows any application that has implemented a License Verification, to query the trusted Google Play licensing server in order to determine whether the application is licensed to the current device and user. A user is considered to be licensed if she has purchased the application or has downloaded the application from the official Google Play Store. License Verification is most common to be used for paid applications but it can also be used by free applications that wants to prevent the unauthorized use of these application. We refer to authorized use of an application only if that

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

application has been downloaded from the Google Play Store.

III. Related Work

Even thought quite some research has been done on ensuring data authenticity and fidelity of sensor data in mobile devices, not all of them are using the TEE. In addition to that, from those that make use of the TEE none of them is emphasizing in the preservation of the authenticity of photos without accepting any modifications. Anyone can claim that requiring the users to upload unmodified data is not practical; however, in the area of user identification this is crucial.

Dua et al., proposed a solution for ensuring the integrity of sensor data. This solution is to equip devices with a Trusted Platform Module (TPM) that allows them to sign their sensor readings and attest their integrity [6]. Despite the fact that this approach can solve our problem, requiring modifications or additions in the existing hardware architecture is not straight-forward and may not be accepted by device manufacturers for cost and portability reasons. Instead of requiring hardware modifications, we propose the use of the existing hardware architecture taking advantage of the TEE that is offered with the same success rate in attesting data integrity.

YouProve, introduced by Gilbert et al., allows untrusted applications to control the fidelity of data they upload and services to verify that the meaning of source data is preserved [8]. They analyze the derived data and generate statements comparing the content of a derived data item to its source. Instead, in the problem we are trying to solve there is a more strict view of data authenticity, where no modification of any type is allowed on the source data. We are also take advantage of the trusted environment of the device, as YouProve does in a different way, and we empower remote services to determine whether the received data matches its source. In addition, we do not require the client to perform any complex analysis on the data and this minimizes our power costs. Our analysis of a photo is almost instant in contrast to YouProve's approach.

In 2010, Gilbert et al. introduced an architecture for trustworthy sensing on mobile devices that utilizes Virtual Machines (VMs) and a Trusted Platform Module (TPM) [7]. In the proposed approach only applications that are considered as trusted, for modifying sensor data, are allowed and are encapsulated within a VM. Any application that is not considered as trusted to modify source data it cannot be used by users. In our framework our scope is also to verify the authenticity of the source data utilizing trusted software and hardware but without allowing any fidelity reduced actions.

Saroiou et al. described two architectures for making sensor readings trustworthy by signing them in the capture device [11]. The main difference between the two architectures is that the one does not require any additional hardware except a TPM. The first one embeds signing hardware in the sensors making them able to sign their readings so that their is no need for verifying services to make any trust assumptions about a device's software. The second one exploits the TPM and VMs in order to minimize the Trusted Computing Base (TCB) by separating sensor drivers from all the other functionality. This approach can be adapted in our problem but instead of requiring hardware modifications, we propose to sign sensor readings utilizing the TEE.

IV. Approach Overview

Our goal is to design a framework that can ensure to a remote service that the provided photo has not been modified and is the same photo captured from the device's camera and at the same time the remote service can verify the authenticity of the received photo. In this section we briefly describe our proposed framework from the time where an application that implements our framework is downloaded from the official Google Play Store and installed in the mobile device to the time where a captured photo is received and verified by a remote server.

At first we assume that the client side is not compromised and the user has no root privileges in her device. The first thing a user has to do is to download the application from the Google Play Store. In order to use the application and capture photos, she has to create an account to the verifying service. At the time of registration we use the Android KeyStore trusted service to create an RSA key-pair in the TEE of the device, which will be used by the client to sign the source photo and by the service to verify the received photo. For this purpose, we also take advantage of the License Verification Library (LVL) offered by Android OS to verify that the application has been downloaded from Google Play Store and that the public key of the RSA key-pair will be successfully submitted to the remote server. We have modified the standard license verification process in a way that the response from the Google License verification server is forwarded to the server along with our public key and the necessary information for the registration.

The server receives this information and before storing them, it verifies the license verification response the same way as it is typically happen in the device. If the verification is successful then it accepts the registration and stores the public key for the corresponding user.

The key insight of our approach is a background service that is tied to the application and undertakes all the critical operations. When a user captures a photo we store the photo in the internal storage of the application. Then the background service reads the photo and computes and signs the hash of that photo using the private key that is store in the TEE. The sign operation is secure and takes place in the TEE where the key resides.

When the user tries to submit the photo, the background service intervenes to verify the authenticity of the photo and to submit it. First, the service computes the hash of the photo that user tries to submit and compares this hash with hash computed before. If the two hashes match then the service submits the signature and the photo to the server.

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

One of the most important contributions of our framework is the ability of the remote service to verify the authenticity of the received photo. When the data has been received, the server computes the hash of the photo with the same hash function as the client side and using the public key for the corresponding user verifies that the signed hash matches the hash of the received photo. If this is successful, then the photo is stored otherwise it is discarded and the client is informed to submit another unmodified photo.

V. Trust and Threat Model

In order to evaluate the security guarantees that our proposed framework is intended to offer, we define a threat model with two types of attackers. In the area of user identification techniques it is more likely that the attacker is the device owner himself who wants to fool the verifying service by providing a modified photo pretending someone else. It is acceptable that an average user is not a technical expert. Additionally, the minority of the Android users have root privileges on their mobile device.

An attacker's goal though is to compromise an application and modify its content using a malicious tool in order to operate the way she prefers. We assume that this is possible and that a modified version of our application has been created and is accessible by anyone who wants to act maliciously. In this section we discuss the different types of attackers we have to defend against and the capabilities of each one and how we prevent each type of malicious action. The assumptions we have made regarding a device's hardware and software configuration are also discussed.

A. Root Attacker

A malicious user with root privileges on his device who is able to run applications under root permissions and modify the file system. A user with technical experience on Android devices.

1) Accessing the content of the Legitimate Application: A root attacker can access the contents and the data of the Legitimate Application by executing super user (*su*) commands using the Android Debug Bridge. This can also be done using a malicious application that has root permissions and executes shell commands. In such incidences there is not a lot you can do to defend against as soon as a user with root permissions can access and modify almost everything in her device. However, we made some propositions for rooted devices that can secure our framework such circumstances. Those propositions are discussed in Section IX.

2) Using the Key Aliases: A root attacker can see the aliases created by the legitimate application by adopting the User ID (UID) of the legitimate application to her malicious application. Then she can execute *su* commands to list all the key aliases. The malicious application will then be able to use the key to sign its preferred data and send it to the server. This is the main problem that our framework faces and can be solved by having the background service, that takes the original photo and secures it, to run in the TEE as a trusted service. This is further explained in the discussion section of this paper.

3) Is the RSA private key visible and accessible to the attacker?: As soon as the attacker has root permission, he can use the key as described above. However, according to the official Android KeyStore documentation [3], the key material can be used but it cannot be extracted from the TEE neither can be viewed. Due to the fact that you have to know the alias of the key in order to use it, this problem can be solved with the same way as the previous one.

Even if the cryptographic keys can be used by an attacker in a rooted device, we chose to store those keys in the Trusted Execution Environment to reduce the attack surface in case of side channel attacks in the normal zone of the device.

B. Non-Root Attacker

A malicious user who wants to exploit the vulnerabilities of the legitimate application using another malicious application created by a technical expert. This application imitates the behavior of the Legitimate Application and creates a communication channel with the server to submit modified data.

1) Manual install of a malicious application: A malicious user can manually install a malicious application that imitates the legitimate application. This application is trying to communicate with our server and submit modified data on behalf of the legitimate application. All the aspects of this threat has been considered. First, by using the License Verification Library (LVL) we can ensure that the server will accept and store public keys only by the legitimate application. Next, by signing the captured photo in the client, the server is able to verify the authenticity of the received photos and discard any photos received by the malicious application.

2) Download a malicious application application from Google Play Store: A malicious user downloads a malicious application that imitates the legitimate application from the Google Play Store. Such an application can easily be uploaded to the Google Play Store since it is considered as malicious only by us. Anyone can upload an application to the store with almost no restrictions. However, this threat can be easily mitigated. When such a malicious application tries to communicate with our server we will detect it and request its removal from the Google Play Store.

3) Using a malicious application that imitates the Legitimate application: A malicious user may try to register to the remote server using a malicious application that imitates the legitimate application. Such an application has been installed manually to the deice. By registering to the remote server with his preferred public key, the malicious user can then submit modified photos to the server that will be considered as valid. This threat is the reason that forced us to perform server-side license response validation during the registration process.

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

C. Trust Assumptions

As stated in Section IV the root of trust for each client application is the TEE where the keys are created and maintained. If the TEE is compromised then an attacker can use the keys and sign modified photos. Nevertheless, this can be achieved only if the attacker has root privileges on his device.

In this subsection we describe the trust assumptions that we have made regarding a device's hardware and software configuration and are listed below:

- The most important assumption that we have made during the design of our framework is that the mobile device where the legitimate application runs is not rooted.
- According to the Android Developers documentation the internal storage of our application is private and can only be accessed by our application and the background service and no other service or application that runs on the normal world can access this storage [4]. This assumption presupposes the first assumption that the device is not rooted. The same applies for the TEE and the RSA key-pairs created by our application.
- According to the Terms and Policies of Google for License Verification, no other applications can forge and use the Licencing of the Legitimate Application in order to fool the server and submit a public key of his choice during the registration process.
- Regarding License Verification process, we assume that the Google Play Licensing server is trusted and will always respond with the appropriate way in a license verification request.
- In the case that a malicious application identical to the legitimate application is uploaded on the Google Play Store and tries to communicate with our remote server we assume that we will detect it and ask its removal from the Google Play Store.
- Finally we assume that the communication channel between the legitimate application and the remote server is based on the TLS protocol and is trusted and no one can intervene. Additionally, the remote server will not allow two TLS connections from the same device at the same time. When this is noticed the server will cancel both and mark the device as untrusted.

VI. DESIGN

In this section we analyze the design of the proposed framework that allows mobile devices and remote services to ensure the authenticity and fidelity of captured photos. The design of such framework can be divided in three parts that are considered equally critical by us and need to be designed carefully. The first part is to ensure the secure installation of the application that implements our framework. The second part is to enable the remote server to decide whether to trust or not the client application and this is achieved using the License Verification Library during the registration of a user to the remote service. Trust means that the server and the application will underpin a secure communication with each other and will sustain it in the future. The third one, which is the main part of our framework, is to ensure that the captured photo has not been modified until is submitted to the remote service. As mentioned before, the key insight in our design is a background service that is tied to the application and performs all the sensitive operations.

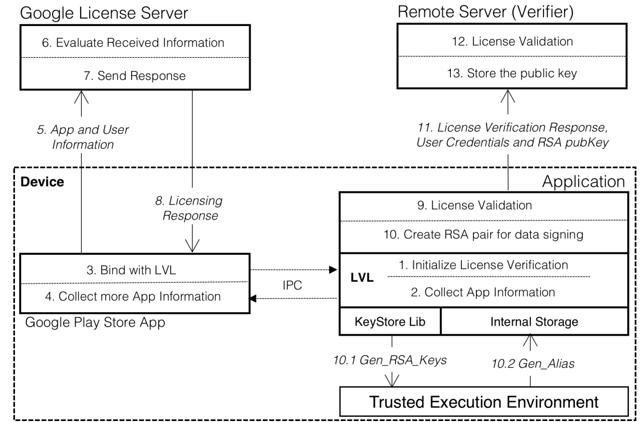

Figure 1. App License Verification and User Registration

Those three parts can ensure that the remote server will receive the authentic photo captured by the camera and in the case that the photo has been modified, the remote service will be able to verify the authenticity of the received photo.

A. Attest the authenticity of the photo-capturing Application using LVL

A developer creates an application and uses her Google Play Developer Console to upload the Android Application Package(APK) to the Google Play Store. As soon as the application is using the LVL [5], Google will provide the developer a public key that she will inject into her Application. As mentioned in Section II, a free application can use the LVL to enhance the security of the application against an unauthorized one.

When requesting License Verification, our application must collect its code's hash, its UID and its package name. Then our application creates a secure Inter Process Communication(IPC) channel with the Google Play Store app and passes the collected information. The Google Play Store application collects all the other necessary information including the device ID and and sends the license verification request to the Google Play Licensing Server. The Licensing server then processes the request and responds back with the licensing status of the application. The Play Store app receives the response and passes it through the IPC to the application. The licensing response, which is signed with the private key of the Google Licensing Server, can now be verified using the injected public key in the application.

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

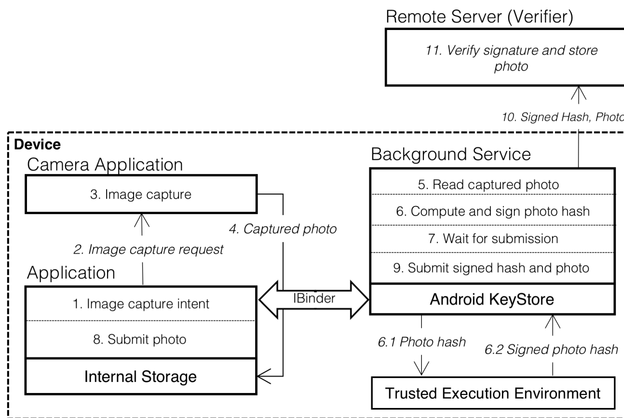

Figure 2. Securing Image Capture using our Background Service

B. Registration and License Verification

After receiving a licensed response, from Google Play Licensing Server, the application can proceed to the registration of the user. The user will provide all the required information, like username and password. The application will then generate an RSA key-pair with a unique alias, *PK* and *SK*. The generation of the keys is executed by a background service using the Android KeyStore system and the keys are stored in the TEE. The private key material, *SK*, is never exposed outside the TEE but can be utilized using the Android KeyStore System [3].

Then, all the user's information along with the license verification response and the created public key are send to the remote service in a registration request. Once the remote service has received the request, it performs server-side license response validation as it is typically happen in the application. If the license response is valid then the remote service will register the user with the provided information and will store the public key. If the license response validation returns a *false* status, then the server realizes that the registration request was send from an unauthorized application and the registration will be canceled. A complete view of the registration process and the use of the LVL is shown in Figure 1.

C. Secure Photo Capturing

This is the main part of our framework and includes the most critical operations. This procedure is a co-operation of the application and the remote service with the common goal to ensure and verify the authenticity and fidelity of a captured photo. The application is responsible to ensure that the captured photo will not be modified before it is submitted to the server. In our approach, even the slightest modification is not tolerated. A complete view of the photo capture process is shown in Figure 2.

As in the registration process, a background service tied to the application, executes all the sensitive operations. We have extended the *Binder* class to provide our application the exclusive use of this background service. This extension prevents any other applications or separate processes from using the background service. Our background service oversees the whole process and takes actions when needed. Below we describe the whole procedure step by step.

At first the user is requested to capture a photo. Using an *IMAGE_CAPTURE* intent we enable the use of the device's camera. When the photo is captured the result photo is returned to the application through the intent. At this moment the background service captures the photo, computes a hash from the photo data and signs this hash using the private key that was stored in the TEE during registration. All the cryptographic operations are executed in the TEE. By executing the sensitive operations in the TEE we protect our framework from software attacks.

The captured photo resides inside the internal storage of the application until it is captured from the background service. The internal storage gives the application private access to its content and in a non-rooted device no other applications can access it. However, it is important to mention that a root Attacker can easily modify those contents.

At the end, when the user tries to submit the photo to the remote service, the background service intervenes and undertakes the submission of the photo. Before submitting though, the background service takes the photo that the user wants to submit and computes its hash. This hash is then compare=0d with the signed photo hash stored from before. If the two hashes match then the photo along with the signed hash are submitted to the remote server by the background service. Otherwise if they do not match then the background service cancel the submission process and informs the user. The only way for a root Attacker to be able to submit such modified data is completely to change the way that our background service works. This vulnerability can be mitigated in a way that is described later in the Section IX.

VII. Implementation

In this section, we describe our prototype implementation on a Nexus 5X with Android 6.0 installed. We have implemented our security framework as an android application using classes introduced in Android 4.3(API level 18) contained in the *Android Keystore System*. Nexus 5X has a Qualcomm Snapdragon 808 processor. Snapdragon 808 processor is based on a combination of ARM Cortex-A53 and Cortex-A57 processors. This processor uses ARM TrustZone Technology [1], a hardware based security designed as a System on Chip (SoC). ARM TrustZone technology *Android Keystore Sytem* has its cryptographic generated keys stored inside a Trusted Execution Environment. In such environment the private keys can be used but the key material is never exposed outside the trusted environment. The cryptographic operations like signing and verification of signatures are also executed in a tamper resistant environment.

Our prototype implementation can be divided in two main parts. The first part is the one that we consider as trusted. That is our background service, which encapsu-

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

lates only the critical functionality of our framework such as cryptographic key generation and signing of the hash of the photo. The background service is implemented in Java with less than 200 lines of valuable code. The second part contains the majority of the application's functionality, which is not considered as critical. Such functionality is the Graphical User Interface (GUI), the camera capture request, the registration and log-in services. Regarding the registration process, all of its sensitive operations (e.g., generation of the RSA key-pair) are implemented in the background service. This part of the applications is implemented in less than 1000 lines of Java code.

Our remote server, which represents the remote service has a LAMP stack installed with Ubuntu 14.04 LTS operating system, Apache v2.0, MySQL server 5.5.49 and PHP 5.5.9. The photo authenticity verification, public key storage, registration and the Google Licensing response validation services are all implemented in almost 600 lines of PHP code.

VIII. Evaluation

In the context of mobile devices, performance overhead and power costs are the most important aspects that need to be considered when proposing changes to the software stacks or when changing a typical procedure like our framework does with the photo capturing process.

In this section we evaluate whether our security framework can successfully ensure the authenticity and the fidelity of a captured photo with the minimal extra overhead. At first, we examine if a malicious application can be detected by the remote service with the server side license response validation. We also verify if the client application and the remote service can successfully determine whether a photo has been captured from a device's camera without being modified. In the end we also evaluate the performance overhead of our framework.

A. Feasibility Evaluation

For this purpose we have implemented two Android applications. The first one has been implemented with the specifications described in Sections VI and VII and we will refer to this application as the legitimate. The second one is an application that imitates the legitimate application and tries to perform license verification in order to register with its preferred public key and submit modified photos to the remote service. We refer to the second application as the malicious application and we assume that in a real world scenario such applications will be installed manually by malicious users. We verify registration and photo capturing processes separately.

1) Registration and License Verification: As described in Section IV, our registration process is one of the two most critical procedures in our framework. This is because during registration we also create a cryptographic key-pair that will be then used to sign the captured photos. Furthermore, we transfer and store the public key to the verifying service. That key will be used by the service to verify the authenticity of the received photos. The goal of this evaluation is to ensure that the registration process is successful and that the correct public key is transferred to the remote service. Additionally, it is necessary to evaluate that our framework can prevent a malicious application from registering and submitting its own public key to the remote service.

At first we tried to register with the legitimate application. Since it is the legitimate application the license verification on the device was successful. As soon as the license verification is successful on the client side and the cryptographic keys has been created, the application submits to the remote service all the necessary information. The license response validation on the server side was also successful and the public key was stored. We repeated the registration process for 10 times and the registration was always successful.

Subsequently, we tried to register with the malicious application. As soon as the application is not published at the Google Play store then the Google Play Licensing server will not respond to any license verification request from this application and the license verification cannot be performed. We tried to submit the registration information along with the preferred public key but without the license verification response but the remote server denied to perform the registration because of the missing license response.

We have not tested our framework with a malicious application that is published in the Google Play market and tries to communicate with our remote service because such applications can be detected and removed from Google Play as described in Section V.

From the evaluation result we can derive that the server-side license response validation can detect and prevent malicious applications from registering to our remote service.

2) Photo capture and submission: Despite the fact that a secure photo capture process is based on a secure registration, it is the most crucial process in our framework. We have evaluated this process with both the legitimate and the malicious application. We used a non-rooted device to perform the evaluation since we already know that in a rooted device a malicious application can use a private key created by the legitimate application to sign and submit a modified photo.

In our legitimate application, all sensitive functionality is performed by the background service. At first we evaluated the photo capturing process with the legitimate application. We captured a photo using the camera of the device and then we tried to submit that photo to the remote service. At the submission time, the background service sent the captured photo along with the signed data to the remote service. Then the service verified the received photo and signature using the public key for the corresponding user. We have also repeated this procedure for 10 times and it was always successful.

In addition to that, we have added an extra method to

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

the legitimate applications that modifies the photo before submission. We have used this method to evaluate whether our application can detect the modified photo and prevent its submission to the remote service. This is detected from the background service when comparing the hash of the captured photo with the hash of the photo that the user tries to submit.

In the end we tried to evaluate whether the remote service can detect modified photos and discard them using the malicious application. First we used a malicious application that tries access the private key created by a legitimate application in order to sign and submit a modified photo to the server. This attempt ended before submitting the photo because the malicious application failed to access and use the private key that was created by another legitimate application. Besides that, we modified the malicious application in a way that it creates its own cryptographic key-pair to sign and submit a photo not captured by the camera. For evaluation purposes we assumed that the user has already registered with the remote service using a legitimate application. When we tried to submit the photo, the remote service was able to detect and discard the modified photo because the public key used by the service to verify the authenticity of the received photo did not match the private key used by the malicious application to sign the photo.

B. Performance Evaluation

Since our proposed framework is intended to run on mobile devices, it was important to evaluate its performance overhead. To do that we deployed our implementation on a Nexus 5X and a Samsung Galaxy S6, which both has a processor with ARM TrustZone Technology and a Trusted Execution Environment. All performance tests were run 20 times while the device was at normal state with only our application running. We have measured the execution time while executing the two most critical operations in our framework: a) registration; and b) photo capturing.

1) Registration process: When evaluating the registration process we have measured the time needed for the license verification request sending and response receive separately from the other registration processes. This is because between those two is the time that the user needs to add her personal details requested for the registration. The average time for the license verification request is 654 ms. The other registration processes including key generation and registration with the remote service takes an average time of 3000 ms to be completed. Thus, taking into account the network latency, we can say that the performance overhead that our framework adds to a typical registration process is negligible.

2) Photo capture and submission processes: At the end, we measured the execution time of the photo capture process separately from the submission process because the time between them must not be considered. This is because the time between those operations depends on the time the user wishes to submit the photo. For the photo capture process we have measured the extra overhead that our framework adds to the typical photo capture process. This extra overhead is the time needed for our background process to take and sign the captured photo and this takes an average time of 284 ms. Again, this extra overhead is acceptable and it does not affect the usability of the application.

Next, we measured the execution time for the photo submission process. This process includes the submission and store of the photo to the remote service and the authenticity and fidelity validation on both the mobile device and the remote service. The average time for this is 780 ms.

Summarizing, we believe that the minimal extra overhead that our framework adds over the typical procedures of photo capturing and submission are negligible and well worth the added authenticity and fidelity guarantees.

IX. Discussion

In this paper we have presented a framework for ensuring authenticity and fidelity of captured photos in mobile devices. Our approach can be considered as a demonstration of how the Trusted Execution Environment is typically used on mobile devices. We use the Android KeyStore service to create and store cryptographic keys in the trusted environment of the device and to perform cryptographic operations. In contrast to other applications, in our framework the attacker is the device owner itself. Thus, having the trusted environment executing secure cryptographic operations is not sufficient for our scope.

As mentioned in Section V, our framework can successfully ensure the authenticity and fidelity of captured photos if the user does not have root privileges in his mobile device. Additionally, even malicious applications that run on non-rooted devices trying to imitate our application can also be detected with the server side license response verification. However, this procedure can be even more secure if the Google Play licensing server sends the response, for a license verification request, directly to our trusted remote server that performs the validation of the response.

Unfortunately, with the current structure of Android OS it is not possible to prevent a user from getting root privileges on his mobile device. However, in online identity verification it is important that such a framework can always ensure the authenticity of a captured photo even if the device is rooted.

Despite the fact that rooted devices are the minority, in our research we have also take them into account and there are some propositions that we have made so that our framework can successfully run in rooted devices. The first one has to do with the way that our background service reads the captured photo. Storing and reading the captured photo from the internal storage of the application is only secure if the user does not have root privileges. Thus, a better way that consists our framework secure, regardless of the state of the device, requires modifications in the existing camera service implementation. We think that modifying the camera service so that it also reports the

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.

source data directly to our background service can prevent a root attacker from modifying the captured photo before our background service reads it.

The background service is the one responsible to create and use the RSA key-pair and to perform the cryptographic operations. In a non-rooted device, those keys are protected by the TEE and are accessible only by our background service, but in a rooted mobile device a malicious application can access and use those keys. In order to prevent unauthorized use of those keys we propose to include our background service in the TEE of the mobile device as a Trusted App [1]. In this way the remote server will be able to directly talk to our Trusted Application and know that the compared signed hashes of our captured photos were not modified just before they were being signed. Furthermore all the cryptographic key alias related to our application will only be accessible from our Trusted Application. This transition is computationally sustainable because the cryptographic operations are already being done in the TEE, however they are exposed as soon as they are bound with a client application, as discussed in *Threat-A.2*.

The Trusted Application will be able to provide authentication to a remote service, not only for the captured photos, but also to various sensitive data captured from the sensors of a device. This can be achieved by slightly modifying our current implementation.

The aforementioned propositions are considered by us as future work but the inclusion of our background service in the trusted environment of the mobile device it also requires an approval from processor vendors.

X. Conclusions

This paper has presented the design and the implementation of a security framework that exploits the capabilities of the TEE offered in commodity mobile devices, in order to ensure the authenticity and fidelity of sensed data. Such framework enables a remote service to determine whether the received data is the real outcome of an acquisition process and has not been modified. The key to our approach is a background service that is tied to the application. This background service reads the captured photo just after its creation and uses cryptographic operations in the TEE of the device to preserve the authenticity of the photo until its submission to a remote server. Based on our evaluation, our proposed framework appears to be feasible and its performance overhead is negligible.

Our framework can also be extended to support any type of sensed data like video and audio. Our future work will be based on extending our support to other types of data and making the proposed framework able to ensure data authenticity and fidelity in rooted devices as described in the discussion section.

XI. Acknowledgment

This research has been fully funded by the European Commission as part of the ReCRED project (Horizon H2020 Framework Programme of the European Union under GA number 653417).

References

[1] ARM. Trustzone technology. http://www.arm.com/products/processors/technologies/trustzone.

[2] Vera H-C Chan. By the numbers: iphone vs android. https://www.yahoo.com/tech/by-the-numbers-iphone-vs-android-97842025474.html, 2014.

[3] Android Developers. Android keystore system. http://developer.android.com/training/articles/keystore.html.

[4] Android Developers. Android storage options - internal storage. http://developer.android.com/guide/topics/data/data-storage.html#filesInternal.

[5] Android Developers. Google play licensing overview. http://developer.android.com/google/play/licensing/overview.html.

[6] Akshay Dua, Nirupama Bulusu, Wu-Chang Feng, and Wen Hu. Towards trustworthy participatory sensing. In *Proceedings of the 4th USENIX Conference on Hot Topics in Security*, HotSec'09, pages 8–8, Berkeley, CA, USA, 2009. USENIX Association.

[7] Peter Gilbert, Landon P. Cox, Jaeyeon Jung, and David Wetherall. Toward trustworthy mobile sensing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10, pages 31–36, New York, NY, USA, 2010. ACM.

[8] Peter Gilbert, Jaeyeon Jung, Kyungmin Lee, Henry Qin, Daniel Sharkey, Anmol Sheth, and Landon P. Cox. Youprove: Authenticity and fidelity in mobile sensing. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 176–189, New York, NY, USA, 2011. ACM.

[9] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N. Asokan. Open-tee - an open virtual trusted execution environment. *CoRR*, abs/1506.07367, 2015.

[10] Global Platform. Trusted execution environment guide. http://www.globalplatform.org/mediaguidetee.asp.

[11] Stefan Saroiu and Alec Wolman. I am a sensor, and i approve this message. In *Proceedings of the Eleventh Workshop on Mobile Computing systems and Applications*, HotMobile '10, pages 37–42, New York, NY, USA, 2010. ACM.

Authorized licensed use limited to: Cyprus University of Technology. Downloaded on July 02,2023 at 07:27:30 UTC from IEEE Xplore. Restrictions apply.