Biometric Bit locker

Design Document

Team: 23

Client/Advisor: Akhilesh Tyagi

Yousef Al-Absi/DevOps, Cole Alward/Scrum Board Master, Morgan Anderson /Scribe, Larisa Andrews/Scrum Master, Ammar Khan/Product Owner, Justin Kuhn/Testing Engineer

sdmay19-23@iastate.edu

http://sdmay19-23.sd.ece.iastate.edu/

Contents

1 Introduction

1.1 Problem Statement

Our problem is the inability to fully encrypt phone data because of the lack of a TPM(Trusted Platform Module) chips on Android phones. We can solve the problem by dynamically generating the key using a PUF(Physical Unclonable Function).

1.2 Project Goals and Deliverables

Goals:

- To develop an open source PUF library.
- To make the PUF work as a lock screen by asking a user to draw shapes to unlock the phone and authenticate then properly.

Deliverables:

- A well tested PUF Java gestures library
    - An open source library that should be released with unit tests and rewritten methods. We need to provide a valid testing framework and an appropriate architecture for the software.
- A PUF based Android app
    - The app should act as a lock screen whenever the phone is closed, it'll authenticate users by asking them to draw a shape and it should only work for the correct user. The app will also automatically start when the phone boots.

2 Design Specifications

- Functional Requirements:
    - Application should appear whenever the phone is locked
    - Application should be able to create multiple profiles
    - Application should be able to authenticate users
    - Application cannot be closed when its locking phone
- Non-Functional Requirements:
    - Performance: Response time for authentication should be less than 4 seconds.
    - Scalability: Application should have more than 2 profiles.
    - Maintainability: The repository should update the application automatically
    - Security: Only the proper user can unlock the application.
    - Data Integrity: Data will be encrypted and decrypted successfully provided the correct key.

2.1 Proposed Design

There is already considerable work done to implement our proposed solution since we are picking up an approximately four-year-old project. The selected solution was to develop an Android application that generates patterns for the user to trace, and then can authenticate and identify the user based on the pressure readings on the screen. Everyone will have slightly different habits regarding how much pressure they apply when tracing certain patterns, but these behaviors are reproducible and distinguishable enough to be used as a reliable means to differentiate users. This system would initially be used to encrypt at the application level, but the goal is to find a way to get this encryption to take place at the kernel level during boot up so that we can achieve full disk encryption more like a traditional bit locker. However, this level of encryption becomes difficult on mobile phones due to the lack of the Trusted Platform Module Chip (TPM) which is used for encryption on laptop computers. We intend to get around this issue by using a Physical Unclonable Function (PUF), which will generate the key that allows us to encrypt and de-crypt data. The PUF has already been developed along with the application that generates the traces and identifies the user. While these parts of the solution have been developed, they still need to be further tested and refined.

Right now, the trace generating application is only able to generate traces and recognize users. We still need to implement the actual encryption of data and we need to integrate the application into the Android operating system. The initial goal is to simply be able to encrypt data at the application level. An example of this would be the user attempting to open a messaging application. The application data should be encrypted and then the user should be prompted to trace some generated patterns. The user would trace the pattern and if they were successfully identified and authenticated, the PUF would generate the key to de-crypt the application data and the user would then be able to access the application. The end goal for our user is to have full disk encryption occur at the kernel level before the operating system is even loaded. However, due to the way Android works, we are not entirely sure if this is possible yet. Our plan is to first implement the encryption at the application level and then progressively move the encryption back until we reach a point where we cannot move encryption any earlier in the boot cycle.

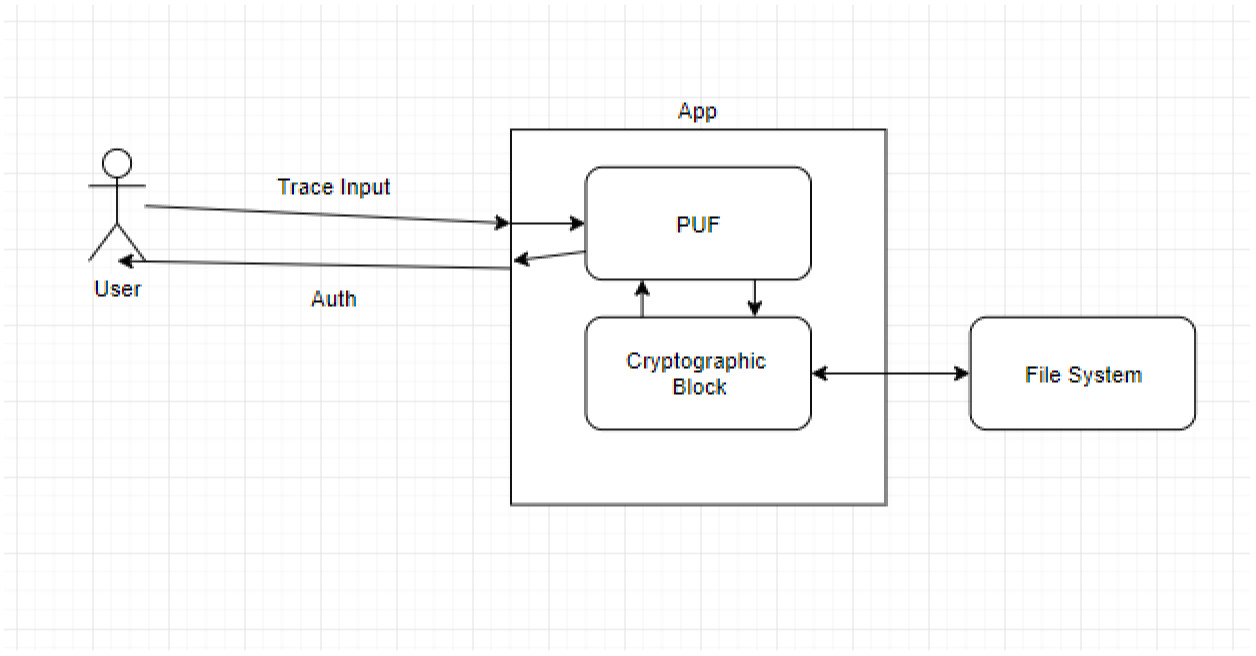

Figure 1 Initial Design Overview

2.2 Design Analysis

The proposed solution has various strengths that will motivate our success. The trajectory of our plan allows us to initially solve what should be an easier problem and use the information to implement a more secure and difficult solution. We also benefit from PUF being so far along in the process; the project was started 4 years ago. Consequently, we have the luxury of focusing primarily on using PUF in conjunction with our solution, mitigating the scope of the problem. Although the PUF concept is relatively new, there are many academic papers analyzing the topic, so we are confident that the part of the solution using PUF is feasible.

However, it is still undetermined if this solution is possible; we may not be able to encrypt at the kernel level. There may be a reason why Android has not shipped products with this level of encryption, such as applications sitting on top of the operating system may not be able to access what they need to. Similarly, very few resources on the subject exist that we can utilize. Although it helps to have a subset of the solution (the PUF library) already functioning to some degree, it could also be a hindrance to update and fix it, as drastic changes may need to be done to it. Although the PUF concept is established, it does not address issues like drastic changes in climate, which would affect authentication.

3 Testing and Implementation

3.1 Interface Specifications

- Interface should be an Android application.
- Interface should have an option for creating a new profile
- Interface should have an option for deleting a profile
- Interface should have accessibility labels
- Interface should have help options for anything in the application.
- Interface should start with the system booting.

3.2 Hardware/Software

Most of our development so far has been Android application development, so we have been doing most of our development and testing with Android Studio. When we eventually begin exploring kernel level encryption, we will begin working at the kernel level and this may require us to use additional software tools but that has yet to be determined as of this stage.

As far as hardware is concerned, all the initial testing that was done with the application before we took up the project was done on Nexus 7 tablets. Currently, we are specifically testing the PUF and the Android trace applications ability to identify a specific user, so we want to minimize hardware variations to really focus on the accuracy of the software. As a result, we will currently continue to test with Nexus 7 tablets however we do plan to increase our range of test hardware later in the project.

3.3 Functional Testing

Every new Java class created should have an accompanying Junit Test class that shows correct and incorrect behavior as well as demonstrates working results. This testing class should coincide with the Java class in question in file hierarchy within the testing folder for ease of access and identification. Unit tests should be created as seen necessary to keep testing requirements unrestrictive to development time, however each new method should see some amount of coverage.

Integration testing should begin once component dependencies start to show in new features, and integration tests should be designed for expected results with the intent of real data being returned from the depending component. Integration testing should be designed with only between required components in a new functionality, with the remaining existing components mocked or stubbed to keep the test functional requirements isolated and reliably tested.

System testing of the proposed solution and its requirements will be initiated as multiple components of the solution approach full functionality. System tests will primarily be created by the test engineer of the team, however other members of the team most familiar with specific components are invited to aid in creating parts of end-to-end testing as well. System tests are end-to-end and will allow us to demonstrate that all functional requirements are enough in a live environment on the Nexus 7 tablet.

Acceptance testing will be down in two stages:

1. Tickets are reviewed at the closing of a ticket where Unit and any Integration tests are validated.
2. Components are verified at the end of their completion to ensure all desired functional requirements are met and pass all system tests.

3.4 Non-functional Testing

There are several non-functional requirements that must be met in acceptance testing to verify the product.

- Performance: Testing will require the actual use of the Nexus 7 tablet for real world scenarios. Given the requirement that full authentication should take no longer than 4 seconds to complete, acceptance testing for this feature will begin as soon as the user traces the onscreen pattern. Authentication systems are heavily reliant on the speed of the device alongside the optimization of the authentication process. In the case of this requirement not being passed, further optimization of the code may be necessary.
- Scalability: Given that more than 2 profiles are necessary for convenient testing, an acceptance test will consist of over 2 user profiles being created on the device while ensuring that all profiles are saved and accessible by each user. Each user with a profile will be able to unlock their application data on the same device.
- Maintainability: Acceptance testing requires that the project repository can update the version of the application automatically and without human intervention. Testing consists of pushing an update to the repository to prompt the repository to upload the updated application. Note that this test must be used sparingly to avoid spamming the distribution platform.
- Security: Testing is comprised of a user prompting the application for the authentication process and tracing the pattern on a device in which they do not have a profile created for them. A passing test consists of the application denying the user access and failing authentication gracefully.
- Data Integrity: Data must be correctly encrypted and decrypted without corruption for the application to operate successfully. Several system tests will be composed of encrypting and decrypting different types of files and

application information with comparisons of the data before and after this process to ensure that the integrity of the data is maintained for the user.

3.5 Modeling and Simulation

To assist in testing the application, our team uses Android Studio to model the layouts of activities throughout the program. Doing so helps ensure the accuracy of the client's desired application appearance. Additionally, Android Studio provides access to an emulator which simulates a user utilizing the application and allows developers to easily test functional requirements. By using this form of simulation, program behavior and a user's experience can be quickly evaluated.

3.6 Implementation Issues and Challenges

There are several implementation issues and challenges related to development of this project. By design, once authentication profiles for each user are created, they are tied to the device they are created on. Therefore, our testing environment is not very mobile, which makes this very difficult as each set of profiles must be tested on an exact device. In this same way, we also cannot develop the application for any other devices in the project's current state as authentication is designed for use with the nexus 7's hardware. Another issue is the level of encryption we can implement within the android SDK. It is unclear yet whether we can implement out application as the operating system is booting for an ideally lower level of protection. It's also unclear if implementing full disk encryption is a feasible solution for this application, or whether it would create an unfriendly user experience requiring constant user authentication from the user by the operating system.

4 Closing

4.1 Conclusion

Currently the team is working on designing and implementing test procedures and documenting the provided library. Once the library is fully tested and documented the team can start work on integrating the library into an android application. Once the application can encrypt and decrypt in a way which satisfies the previous functional and non-functional requirements the feasibility of encrypting at a kernel level can be decided upon. The goal for the end of the project is to deliver to the client a functioning, well-tested and documented application that can encrypt/decrypt using the PUF at the lowest android level feasible.