Biometric Bit locker

Project Plan

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/

Revised: 10/26/18

List of Figures

Table of Contents

1. Introduction

1.1 Problem Statement

Asymmetric encryption is an encryption that allows us to hide our data and makes sure it's secure. Typically, files would be decoded using a public key and then they would be encoded using a private key. The private key is stored in the TPM (Trusted Platform Module) which is a cryptographic module that enhances computer security and privacy. TPM chips are usually discrete chips soldered into a computer's motherboard which allows them to be separate from the rest of the system. Android phones, however, lack the TPM chip. Therefore, encryption keys must be stored on the device somehow. If the keys are stored on the devices, they can be found and could fall into malicious hands.

1.2 Purpose

Our solution to the problem is to dynamically generate the key using a PUF (Physical Unclonable Function). If we do that, the key will not be stored anywhere, and it can only be generated at runtime. Solving both the security issue and the storage of the key.

1.3 Goals

Our goal is to develop an open source PUF library and then to integrate the library in the android OS. Using the library to authenticate the user by dynamically generating a private key. Which, then encrypts user's data when the user's phone is shut down and decrypts on boot after authentication.

1.4 Intended Users and Uses

The integrated PUF would be used by any person who has a phone with information that they deem worthy of protecting.

1.5 Assumptions and Limitations

Assumptions:

- The PUF library is working at the end of the first semester.
- Android continues to support full disk encryption.

- Android allows developers to integrate in the boot sector.

Limitations:

- Nexus 7 hardware.
- Previous PUF library implementation and research.

2 Project Deliverables and Specifications

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.
    - At least 70% test coverage
- 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.
    - Authentication should happen within 5 seconds.
    - App should work on phones with 1 GB of RAM
    - App size should not exceed 80 MBs.

Dates for deliverables detailed in timeline section.

3 Design

3.1 Previous Work/Literature Review

For our project, we will be working with existing software and integrating it into our project. The given implementation is named PUF. Presently, the PUF creates a unique key from data received from the user. It receives this data by displaying a pattern on the screen and forcing the user trace the design several times. From this data, a function will find a trace range the user should always fall between. When the user attempts to draw the shape of the design again, it will recognize the user. We received our knowledge of PUF through a paper published by our client. [1]

When considering whether similar products are in the market, no comparable implementation has been found. However, there are similar technologies available that offer different functionalities. PUF is not a new way of securing a device. In fact, the most common use of this technology for securing phones has been the fingerprint scanner, which is a property found on most mobile phones. Additionally, the idea of an Android bit locker is not new either. Having a secure way to unlock one's data once it has been encrypted has been a problem that many are trying to solve. Microsoft has a dated bit locker that uses a key pair to encrypt and decrypt data. Their solution utilizes a USB stick, a PIN or both to be able to authenticate the user. [2]

Although popular the finger-print scanner is not always secure. In a study by Michigan State and New York University [3] it was found that there was a way to break a finger print scanner 65% of the time. In a world where we are not authorizing all sorts of payments from our phone, we think our solution would be better. As it is more unique in its generation of a key and covers more reference points than a finger scanner.

We have touched on pure authentication but what about the other side Awe need to be able to encrypt the data, hopefully at a kernel level. Android already has the encryption level that we are hoping to implement, Full-disk encryption. This is where when the phone is turned off all the functionality of the mobile device would be encrypted until the user is authenticated at boot. [4]

3.2 Proposed Design/System/Solutions

For our project, some of the groundwork for the solution has already been completed. The current solution is to develop an Android application that uses data generated from pressure readings returned by the screen when the user traces a generated shape like the Android unlock pattern. The data would be generated by a Physical Unclonable Function (PUF), which has already been implemented. As stated earlier, mobile devices lack the Trusted Platform Module (TPM) chip which can be found on most laptop computers. This enables full disk encryption on laptop computers and each TPM has its own unique and secret RSA key that will de-crypt the disk. Since each chip has its own unique RSA key, this makes it an extremely secure method for encryption. With the lack of this chip on mobile devices, there is a lack of such a secure method for encryption. Ideally, the PUF will emulate the security of the TPM chip by dynamically generating the key every time and thus getting rid of the need to save it. The key generation will occur when the user traces the pattern generated by the app. The app provides the user with a given trace and has them trace it a certain amount of times depending on the selected user selected strength (higher the strength, the higher the number of traces) to develop a profile for the user based on the patterns for pressure and speed the user exhibits for that particular pattern. Furthermore, the hardware for no two devices is the same, even among devices of the same model. Therefore, the pressure readings will vary from screen to screen and thus importing a profile to another device should result in failure even if the same person traces the pattern on the new device. Therefore, this implementation will lead to both user authentication and device authentication, thus leading to maximum possible security with the given hardware.

The pattern tracing and user authentication features have also been implemented but need to be updated, reworked and thoroughly tested before we can deem them functional or reliable. The implementation of the pattern tracing and user authentication system currently has the user trace a given pattern X amount of times when a new profile is created. When creating the profile, the user can change a Strength field. The higher the user sets the strength, the more times they will need to trace to create the profile and the more accurate and specific the authentication algorithm will be. From these initial traces, the application will take x,y and pressure measurements at a variable number of points along the user trace. Over the course of the user traces, the application will develop an average user trace which will be used for authentication. Along with this average line, the application will find a line that is 2 deviations above this average and 2 deviations below the average, thus creating a zone of acceptance. When the user tries to trace the pattern later, the trace they generate will be evaluated at 32, 64 or 128 points along the trace depending on the settings and the length of the trace. From these points, a certain percentage need to fall within the zone of acceptance for the trace to pass and for the user to be authenticated. This is illustrated in the Fig. 1 and Fig. 2 from Dr. Akhilesh Tyagi's team's paper [1] Fig. 1 shows a trace that passes as every point fell within the zone of acceptance whereas Fig. 2 shows a trace that failed as 22 out of the 32 points fell out of the zone and failed.

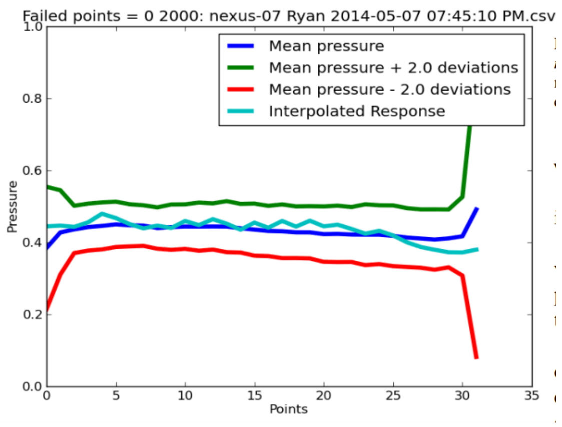


FIGURE 1 STATISTICAL CONCENTRATION/CORRECTION FOR PROFILED USER

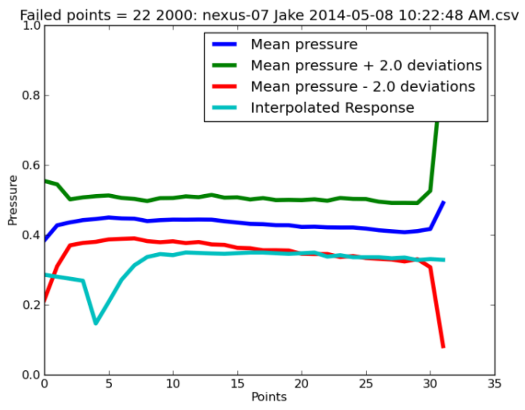


FIGURE 2 STATISTICAL CONCENTRATION/CORRECTION FOR A USER OTHER THAN PROFILED USER

Currently, the goal of the solution is to integrate this application into the Android OS, so we could have application level encryption. This may look like a series of patterns opening that you must trace and pass when you open a certain application. If we can get this functionality in place, we will begin to move the encryption further back into the boot up process. Our end goal is to be encrypting and requiring user authentication at the kernel level before the OS is booted, which is like Bit locker on Windows computers. However, the feasibility of this solution is currently in question, and research is being done to see how if we could implement this feature at all. We have not explored any alternatives to this solution because this project is a research project whose purpose is to examine the feasibility and security of using a PUF in Android phones to emulate the encryption behavior of TPM chips found in laptop computers.

3.3 Assessment of Proposed Solution

The proposed solution has a strength that most other encryption methods contain, and that is PUF implementation. When functional, this allows some of the most effective authentication available, as only the user on their exact phone has access to their application data. Having data saved at an application level allows only specific data to be encrypted, which can be a benefit if there are certain file systems of the device that must stay unencrypted for general functionality to resume but could also be a weakness as the user's operating system wouldn't be given the same protection, making the device susceptible to some attacks still.

An alternative solution is boot and kernel level encryption, which offers even lower-level security for the entire system, however the trade-off with this design is that several usual background applications and general functionalities of the device would need constant authentication from the user to be allowed to operate normally, which isn't very user friendly. As ideal as this solution is from a security standpoint, it may not be feasible to make changes to the android kernel at this level, and it may have to be partly implemented as a surface app that starts as soon as boot sequence ends.

A weakness of tying the authentication system to the silicon level of the hardware device used for advanced security is the limited support for the models of devices supported. The usage of this application requires the user use a specific hardware model and they are tied to that exact device to remain secure. This creates a higher cost for those who don't already own a supported device and costs the project more as well to add support for multiple models as each hardware device requires an individual design of the authentication system.

3.4 Validation and Acceptance Test

Each project requirement requires a test case for validation alongside passing the acceptance testing criteria to be considered sufficiently implemented:

Functional Requirement 1: Only a user with an authentication profile may gain access through to user data

Test Case:

1. User A creates an authentication profile on the test device.
2. At least 2 different participants including User A attempt to access an application which requires authentication to access user data.
3. Each participant traces the authentication pattern.

Acceptance Test:

No person other than User A may unlock or decrypt user data via trace authentication through their authentication profile. Requires at least a 95% acceptance rate while maintaining less than 1% false positive results of unauthorized users gaining access.

Functional Requirement 2: Device cannot be shut down when application is encrypting

Test Case:

1. User A has an existing profile on the test device, or otherwise creates one.
2. User A shuts down and starts up the device.
3. During startup, User A attempts to shut down the device at boot.

Acceptance Test:

User A should be prompted with a message notifying that the device cannot be shut down by normal means until the encryption steps at boot for the application complete

Non-Functional Requirement 1: The application should be able to store multiple user profiles

(Scalability)

Test Case:

1. User A creates a user profile with the application.
2. User B creates a separate user profile with the application.

3. Both users attempt to access user data one at a time, and each trace the authentication pattern when prompted.

Acceptance Test:

Both User A and User B should be properly authenticated through their respective authentication profile and are given access to the user data. At least 95% acceptance is expected while less than 1% of the time providing access to the user via the another's authentication profile.

Non-Functional Requirement 2: Response time of the authentication process should be no

more than 4 seconds (Performance)

Test Case:

1. User A has an existing profile on the test device, or otherwise creates one.
2. User A attempts to access user data and traces the authentication pattern when prompted.

Acceptance Test:

Once User A has completed tracing the given authentication pattern, the application should begin and complete the authentication process as well as notify the user of the result within 4 seconds. Requires 95% acceptance to be deemed successful, as the application or device is occasionally expected to see slowdowns.

All acceptance test cases require manual system testing utilizing the hardware device, however validation testing when completing an issue can be enough with Java Virtual Machine unit tests depending on the natures of the component. Unit testing follows the discretion specified in the test plan.

4 Project Timeline

The schedule for this project is based on the rolling wave planning technique, which enables the team to discuss and make decisions concerning the project as it progresses. Tasks that should be completed soon are discussed in-depth, whereas tasks scheduled for later dates are discussed in a more abstract, high-level manner. The current team is composed of individuals who have the necessary knowledge of the project parameters, hardware and software used in the project to make educated estimates for the projected times. Once the current tasks are completed and it is time to analyze the subsequent tasks, dates may be modified depending on the coming in-depth discussions and knowledge of the team.

The following diagrams present the process (**Figure 3**) and projected schedule (**Figure 4** and **Figure 5**) our team will utilize over two semesters, respectively.

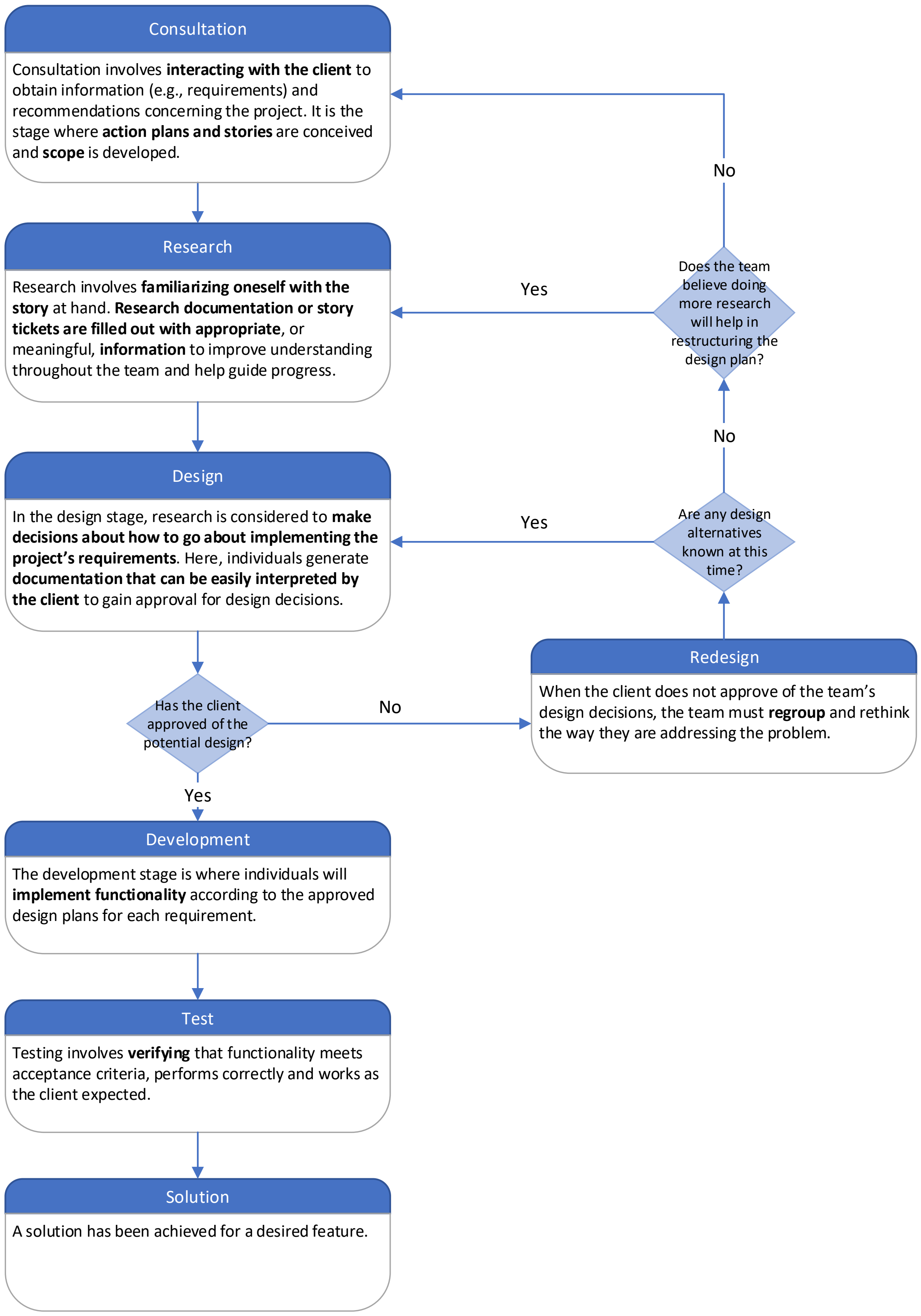
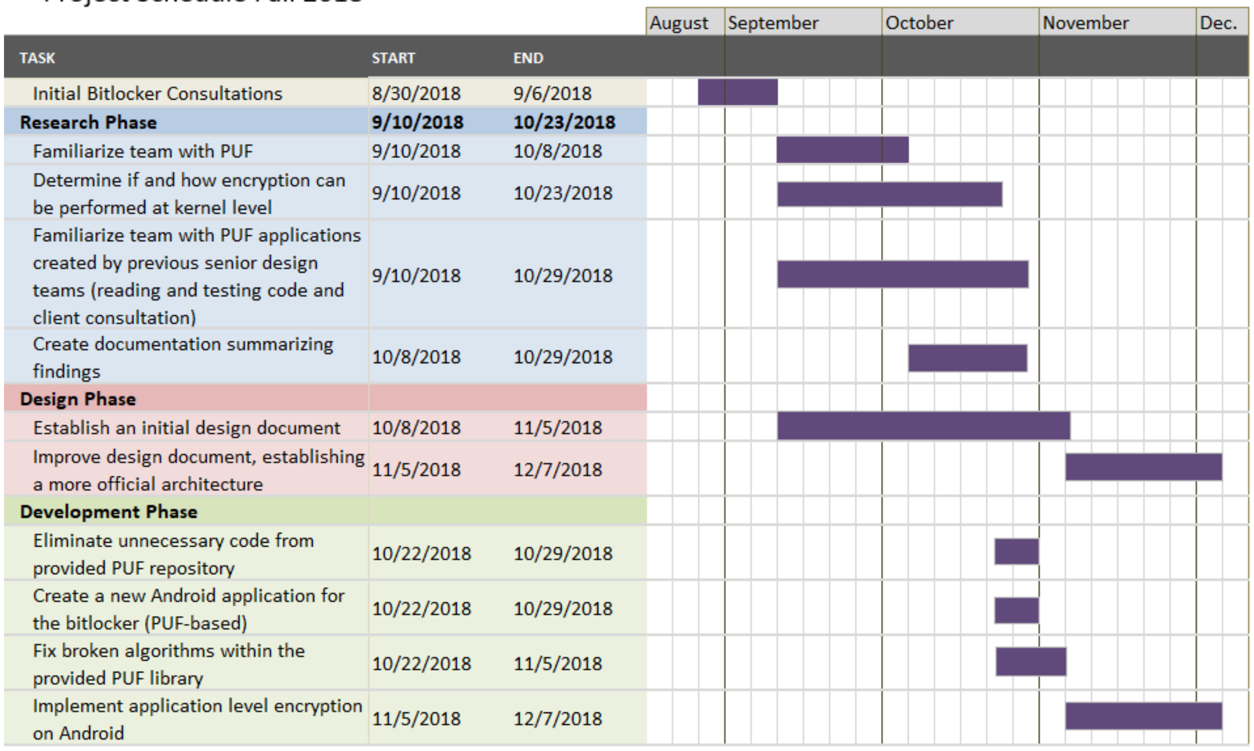

Consultation
Consultation involves **interacting with the client** to obtain information (e.g., requirements) and recommendations concerning the project. It is the stage where **action plans and stories** are conceived and **scope** is developed.
Research
Research involves **familiarizing oneself with the story** at hand. **Research documentation or story tickets are filled out with appropriate**, or meaningful, **information** to improve understanding throughout the team and help guide progress.
Design
In the design stage, research is considered to **make decisions about how to go about implementing the project's requirements**. Here, individuals generate **documentation that can be easily interpreted by the client** to gain approval for design decisions.
Has the client approved of the potential design?
No
Yes
Redesign
When the client does not approve of the team's design decisions, the team must **regroup** and rethink the way they are addressing the problem.
Are any design alternatives known at this time?
Yes
No
Does the team believe doing more research will help in restructuring the design plan?
Yes
No
Development
The development stage is where individuals will **implement functionality** according to the approved design plans for each requirement.
Test
Testing involves **verifying** that functionality meets acceptance criteria, performs correctly and works as the client expected.
Solution
A solution has been achieved for a desired feature.

Figure 1 Process Diagram

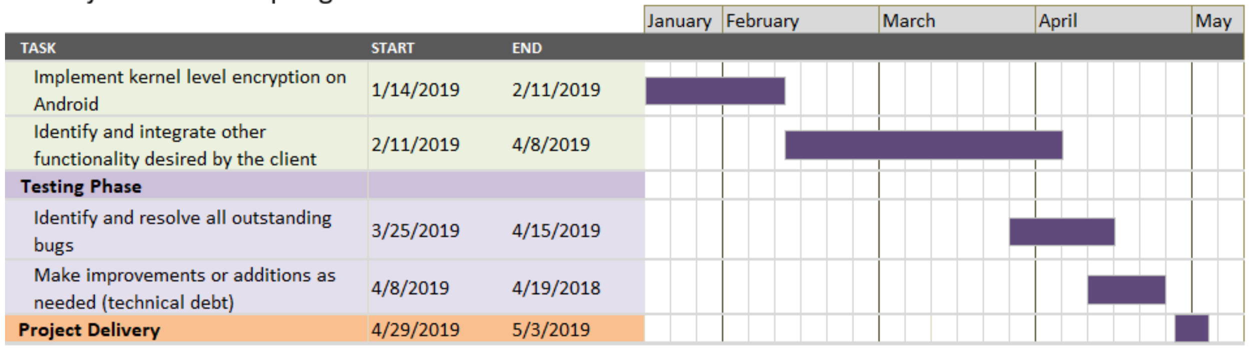
Project Schedule Fall 2018



TASK	START	END
Initial Bitlocker Consultations	8/30/2018	9/6/2018
Research Phase	**9/10/2018**	**10/23/2018**
Familiarize team with PUF	9/10/2018	10/8/2018
Determine if and how encryption can be performed at kernel level	9/10/2018	10/23/2018
Familiarize team with PUF applications created by previous senior design teams (reading and testing code and client consultation)	9/10/2018	10/29/2018
Create documentation summarizing findings	10/8/2018	10/29/2018
Design Phase		
Establish an initial design document	10/8/2018	11/5/2018
Improve design document, establishing a more official architecture	11/5/2018	12/7/2018
Development Phase		
Eliminate unnecessary code from provided PUF repository	10/22/2018	10/29/2018
Create a new Android application for the bitlocker (PUF-based)	10/22/2018	10/29/2018
Fix broken algorithms within the provided PUF library	10/22/2018	11/5/2018
Implement application level encryption on Android	11/5/2018	12/7/2018

FIGURE 4 PROJECTED FALL 2018 SCHEDULE

Project Schedule Spring 2019



TASK	START	END
Implement kernel level encryption on Android	1/14/2019	2/11/2019
Identify and integrate other functionality desired by the client	2/11/2019	4/8/2019
Testing Phase		
Identify and resolve all outstanding bugs	3/25/2019	4/15/2019
Make improvements or additions as needed (technical debt)	4/8/2019	4/19/2018
Project Delivery	4/29/2019	5/3/2019

FIGURE 5 PROJECTED SPRING 2019 SCHEDULE

5 Challenges

The feasibility of our project is undetermined, because it is unknown whether encryption can be performed at the kernel level. Android does not use encryption at the kernel level in its own releases, so it may be the case that it is infeasible for them to implement it. However, it is not known whether this is the case or not. It may be possible to perform this encryption, but functional reasons prevent this from being implemented, such as repetitive authentication.

There are risks involved with integrating the PUF library into a brand-new solution. No member on the team has had experience with the library or the concept physically unclonable functions in general. We have witnessed issues with an existing application that uses the PUF library, so it has already been demonstrated to be susceptible to failure.

However, given our previous work with each other and progress already completed, we do not believe the project to be outright infeasible. Most of our project will be working with Android, which we all have experience working together on. We have created a user application, which the initial goal of this project calls for. The project also involves operating system and kernel levels of programming, and most of us have taken or are taking "Introduction to Operating Systems" or "Linux Operating Essentials," which provide a good foundation for the programming skills and knowledge required. The project is an extension of a project that has been in progress for a few years now, so there is already lot of research and development to use as resources, and the problem space has been thoroughly explored. There are several academic articles written about PUF, justifying its practicality.

There also exist risks pertaining to authentication. Although the PUF library generates the keys used for authentication, the application that uses it must accurately authenticate a user by deciding whether a user sufficiently replicated a key. Deciding what constitutes as enough carries risk into the equation. To make the application user friendly, it cannot produce false negatives when authenticating a user. To make the application secure, it cannot produce false positives, which will jeopardize data that is supposed to be secure. This will require thorough testing as well as statistical development to ensure accurate authentication.

5.1 Cost Considerations

Software

The project primarily utilizes open source software that incur no cost. We are using Gitlab as version control, but there are free alternatives.

Hardware

To accurately test the application, the project requires physical devices that can run Android. For this reason, we have acquired two Nexus 7's from the Electronics Technology Group. The cost for a Nexus 7 is roughly $199, making the total cost of the project $398.

	Device	Cost
1	Nexus 7	$199
2	Nexus 7	$199
		Total: $398

FIGURE 6 COST TABLE

6 Process Details

Our process will consist of establishing functionality at the lowest level of security and later implementing the solution at the highest level. Our initial version will establish success at the user application level, at which point we will advance to explore a solution utilizing higher security. This may or not be the kernel level. We will use the information learned from completing the initial version to assess what the next step should be. The goal is to encrypt a key at the kernel level.

6.1 Standards

We are not working in a traditional lab environment for this project and most of our work is completed outside of a collaborative environment. Most of the work is going to be software development and we have decided to follow the agile model for development, so all our standards and protocols, such as sprints, sprint planning and meeting practices are adapted from the agile model. As a result, our standards and practices should all be approved by IEEE and none of them should considered unethical by any organizations.

6.2 Test Plan

As code is created, developers must create unit tests simultaneously to test their components and ensure the expected behavior is operational and the number of bugs introduced to the repository is minimal. The longer a bug or malfunction is present, the more difficult the bug is to fix since code may build on top of it. Unit tests will be in a parallel directory with similar path and test file names as the actual code. While no minimum or maximum amount of unit tests are required, all core functionality and common edge cases should be unit tests.

As core components are combined, integration tests should be designed to show the implementation of other targeted components delivers results as expected. These tests may be created by collaborating developers or by the test engineer. These tests should focus on core functionality and interactions the connecting components share. They are often still separate from fully operational actions and focus on specific functionalities.

System tests should demonstrate fully operational tests between multiple components using actual data. These should demonstrate intended functionality, and, for our purposes, some may even be end-to-end tests. These tests will be developed by either collaborating developers in need of data from a live environment or by the test engineer to show operational system behaviors.

7 Conclusion

Our goal for this project to integrate existing software that details a PUF into an android based OS application. We will do this by research, design, development, testing and conferring with our client. We think this is a worthwhile project because the use of a pressure PUF as a bit locker for a phone has not been done and will add needed security. We hope to complete the whole application in entirety by the end of spring 2019.

8 Table of References

[1] A. T. Ryan A. Scheel, "Characterizing Composite User-Device Touchscreen Physical Unclonable Functions (PUFs) for Mobile Device Authentication," in *Proceedings of the 5th International Workshop on Trustworthy Embedded Devices*, Denver, Colorado, USA, 2015.

[2] B. Lich, L. Poggemeyer, J. Groce and J. Hall, "BitLocker," 15 October 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview. [Accessed 26 October 2018].

[3] J. Titcomb, "Why your smartphone's fingerprint scanner isn't as secure as you might think," Telegraph Media Group Limited, 2017.

[4] "Android Source," Android Open Source Project , [Online]. Available: https://source.android.com/security/encryption/full-disk#how_android_encryption_works. [Accessed 26 October 2018].