sdmay19-23 presents:

Mobile, Biometric Bitlocker

Advisor and Client: Dr. Akhilesh Tyagi and Timothy Dee
http://sdmay19-23.sd.ece.iastate.edu/

Problem Statement

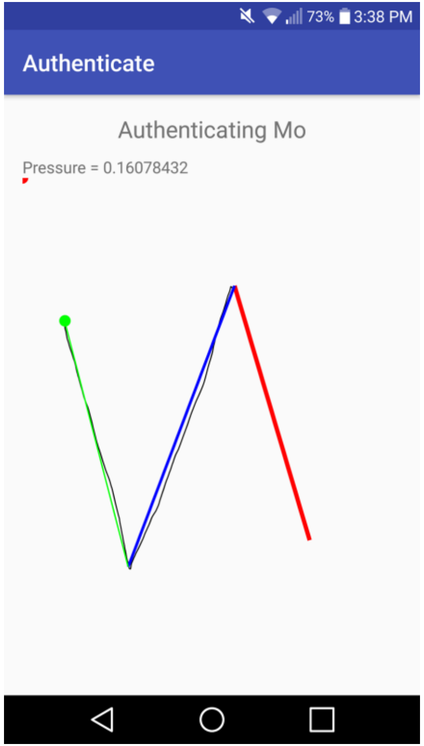
Problem:

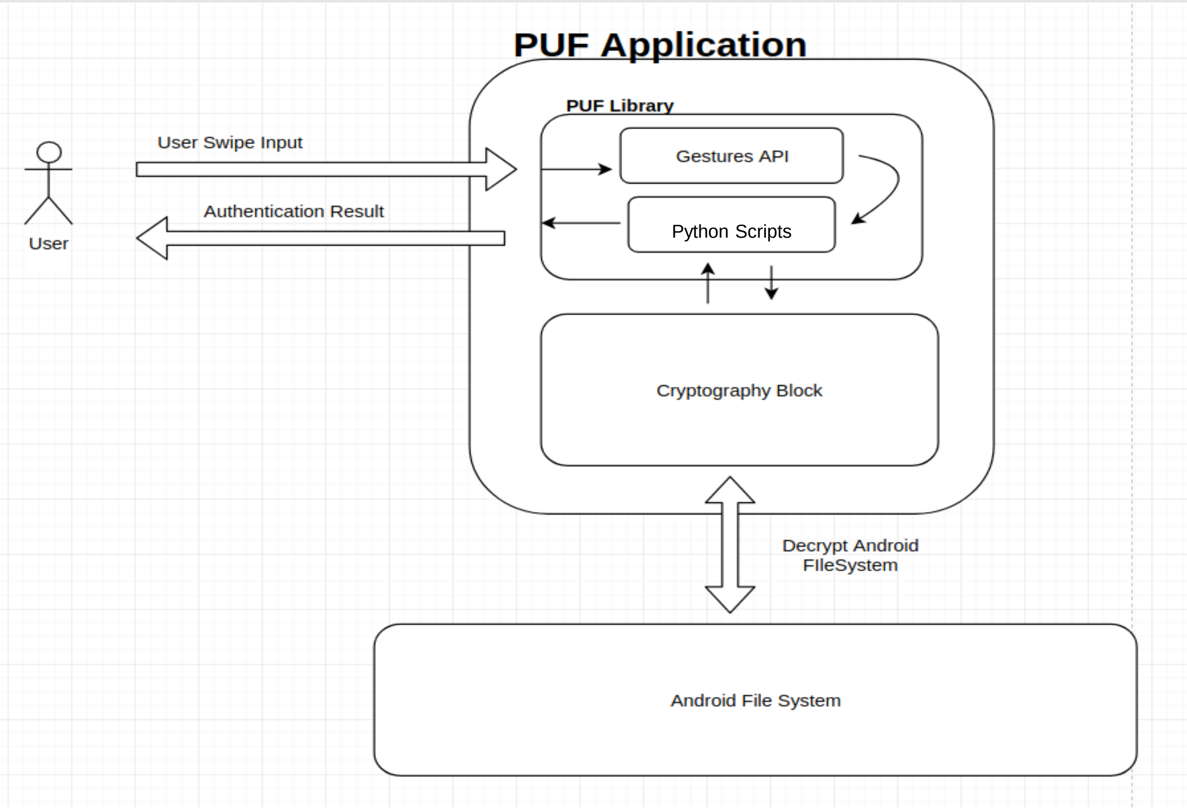
- Android phones lack a Trusted Platform Module (TPM).
- Encryption keys must be stored somewhere somehow.
    - If the keys are stored on the devices, they can be found and could fall into malicious hands.

Solution:

- Dynamically generate the key using a Physical Unclonable Function (PUF).

Initial Conceptual Sketch





Requirements Specification

Functional Requirements

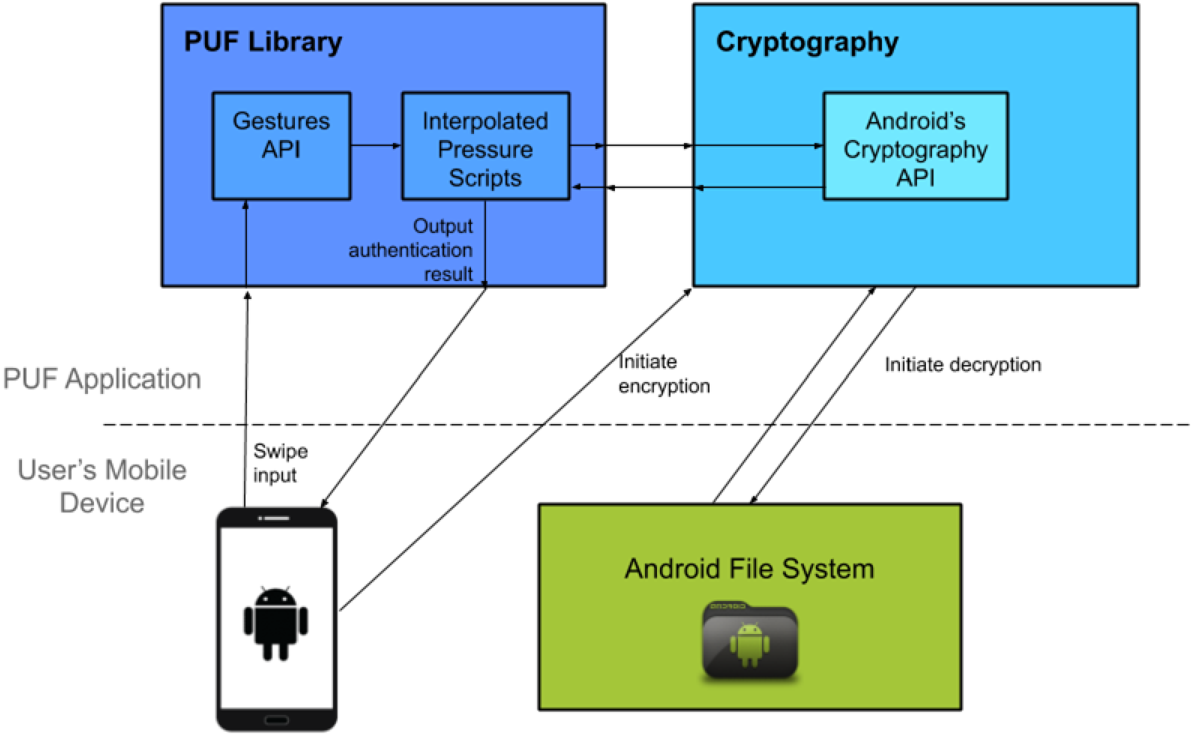
- Application should encrypt and decrypt data without corruption.
- Only an authenticated user may access data.
- Encryption continues when phone is locked.
- Authentication required when an application is opened.

Non-Functional Requirements

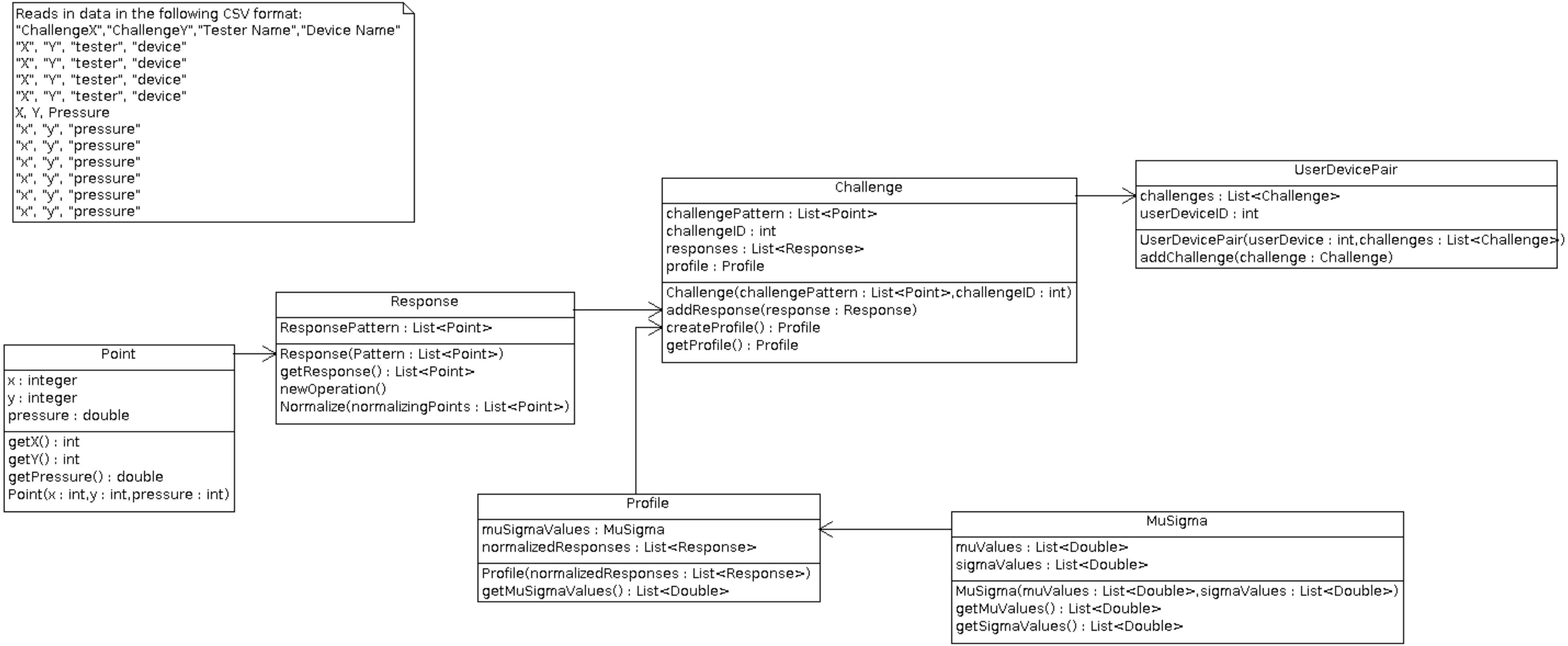
- Application can store multiple user profiles.
- Response time for authentication takes no longer than 5 seconds.
- Gitlab repository should generate the application APK automatically.
- Only the creator can unlock their profile.
- Authentication accuracy of at least 80%.

System Design and Development

System Design



PUF UML



Reads in data in the following CSV format:
"ChallengeX","ChallengeY","Tester Name","Device Name"
"X", "Y", "tester", "device"
"X", "Y", "tester", "device"
"X", "Y", "tester", "device"
"X", "Y", "tester", "device"
X, Y, Pressure
"x", "y", "pressure"
"x", "y", "pressure"
"x", "y", "pressure"
"x", "y", "pressure"
"x", "y", "pressure"
"x", "y", "pressure"

Challenge

challengePattern : List<Point>
challengeID : int
responses : List<Response>
profile : Profile

Challenge(challengePattern : List<Point>,challengeID : int)
addResponse(response : Response)
createProfile() : Profile
getProfile() : Profile

UserDevicePair

challenges : List<Challenge>
userDeviceID : int

UserDevicePair(userDevice : int,challenges : List<Challenge>)
addChallenge(challenge : Challenge)

Response

ResponsePattern : List<Point>

Response(Pattern : List<Point>)
getResponse() : List<Point>
newOperation()
Normalize(normalizingPoints : List<Point>)

Point

x : integer
y : integer
pressure : double

getX() : int
getY() : int
getPressure() : double
Point(x : int,y : int,pressure : int)

Profile

muSigmaValues : MuSigma
normalizedResponses : List<Response>

Profile(normalizedResponses : List<Response>)
getMuSigmaValues() : List<Double>

MuSigma

muValues : List<Double>
sigmaValues : List<Double>

MuSigma(muValues : List<Double>,sigmaValues : List<Double>)
getMuValues() : List<Double>
getSigmaValues() : List<Double>

PUF Library Decomposition

- Gestures API
  - Reads data from user-device gesture interaction
  - Creates user device pair
    - Provides challenges
    - User completes challenges
    - Challenges create profile
    - User-device pair created from list of challenges and their responses

- Interpolated pressure scripts
  - Performs various statistical analyses and operations
    - Normalized trace: represent each trace as a set of pressure values at certain points
    - Authenticate based on normalized trace

Cryptography Block

- Android cryptography API is used for encryption
    - Have researched dm-crypt and fscrypt for kernel level

- Attempting to mimic Trusted Platform Module (TPM) on computers

- Client's end goal is to encrypt at the kernel level like Windows Bitlocker (full-disk encryption)
    - Started with application-level encryption
    - Progressively researched and experimented with encrypting at lower levels

- Kernel-level encryption deemed infeasible

System Requirements

1. Software
   a. Android Operating System
   b. Minimum Version 5.1 (Lollipop)

1. Hardware
   a. Should be able to be run on any hardware that is running required Android Version
   b. Should have a touch screen

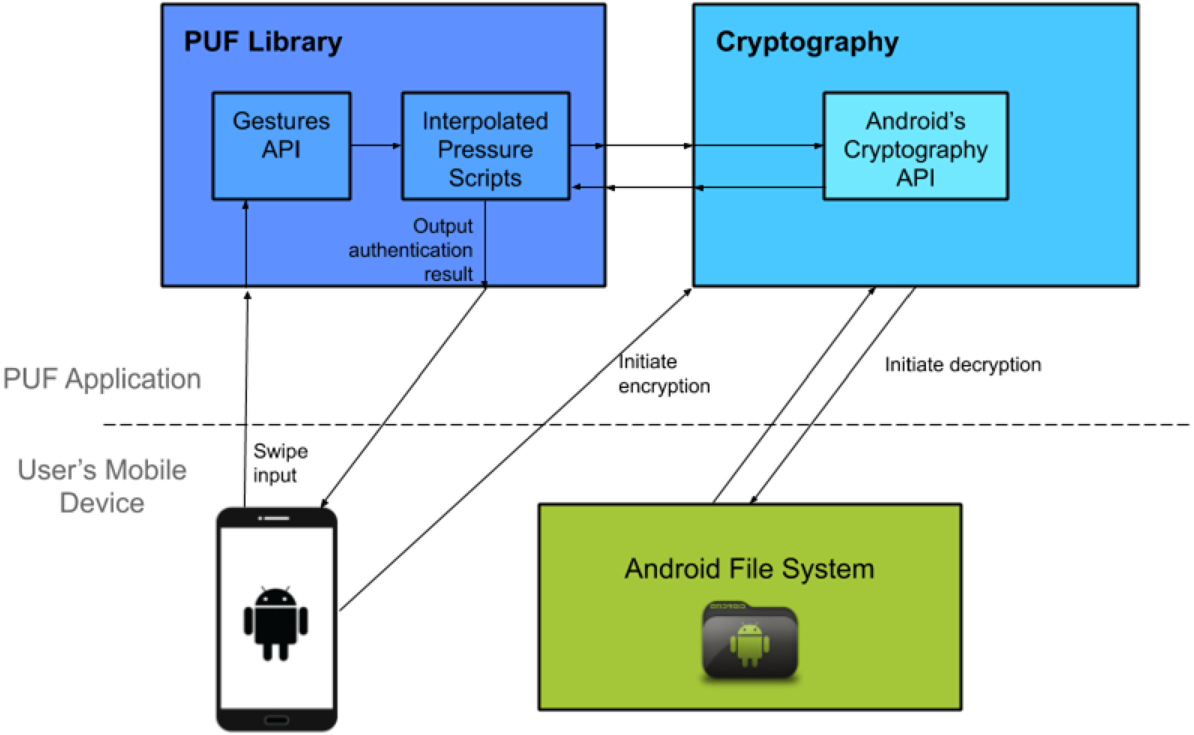
1. Operating Environment
   a. Nexus 7

Implementation

Demo

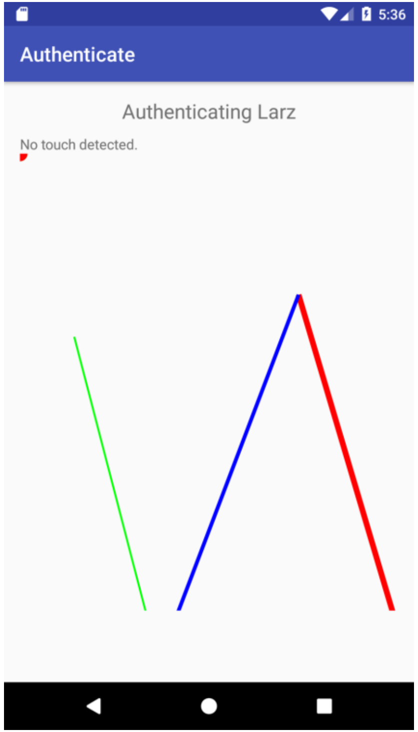
10:16
94%
46°
Ames
Updated 4/30 6:50 AM
11
days left
Graduation Day
Say "Ok Google"
DU RECORDER

Final Implementation



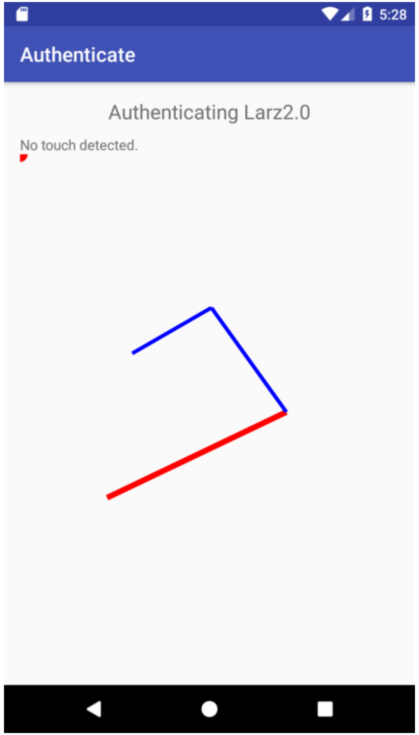
PUF Library

- Fixed implementation
- Fixed trace pattern



Authenticating Larz

No touch detected.



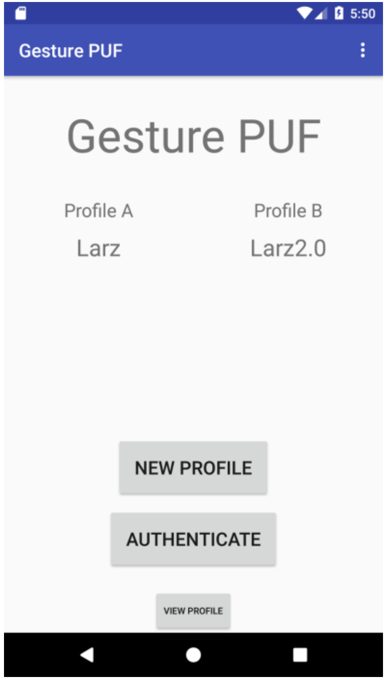
Authenticating Larz2.0

No touch detected.

Encryption and Mobile Application

- Added ability to encrypt and decrypt using a dynamically generated key

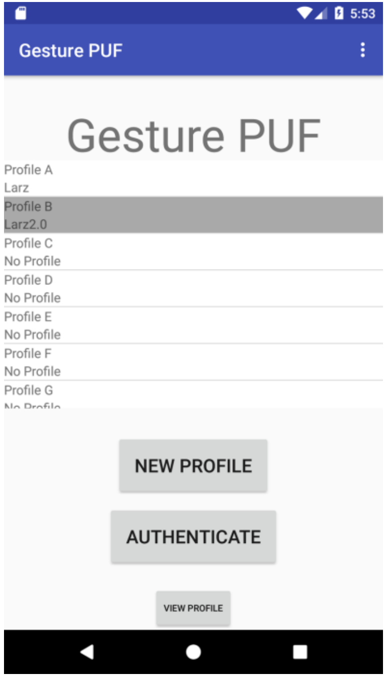
- Added multiple profiles



Gesture PUF

Profile A
Larz

Profile B
Larz2.0



Gesture PUF

Profile A
Larz
Profile B
Larz2.0
Profile C
No Profile
Profile D
No Profile
Profile E
No Profile
Profile F
No Profile
Profile G
No Profile

Rationale

- PUF design decisions
  - The only decisions we were apart of was rewriting internal scripts.
  - We wrote internal Python scripts in Java.
  - To fix them and for easier use in kernel.

- Stopping at application level
  - We decided to stop at application level due to PUF library issues.
  - We were spending more time updating and fixing PUF than on implementing other features.
  - No straightforward way to integrate at kernel level.

Testing, Validation and Evaluation

Test Plan

- Unit Testing
  - Designed simultaneously during development
  - Tests core functionality of a component

- Integration Testing
  - Created when combining between multiple components
  - Designed to test specific interactions between two components

- System Testing
  - Full operational behavior of the application with actual data
  - Largely performed towards the end of project

Sample Test Cases

- Only authenticated users may access user application data
    - User A has authentication profile
    - User A and B complete authentication trace
    - Only User A with an authentication profile is allowed

- Response time of authentication should take less than 5 seconds
    - User A completes authentication trace
    - Authentication process notifies user A results in under 5 seconds

DevOps

Tests:

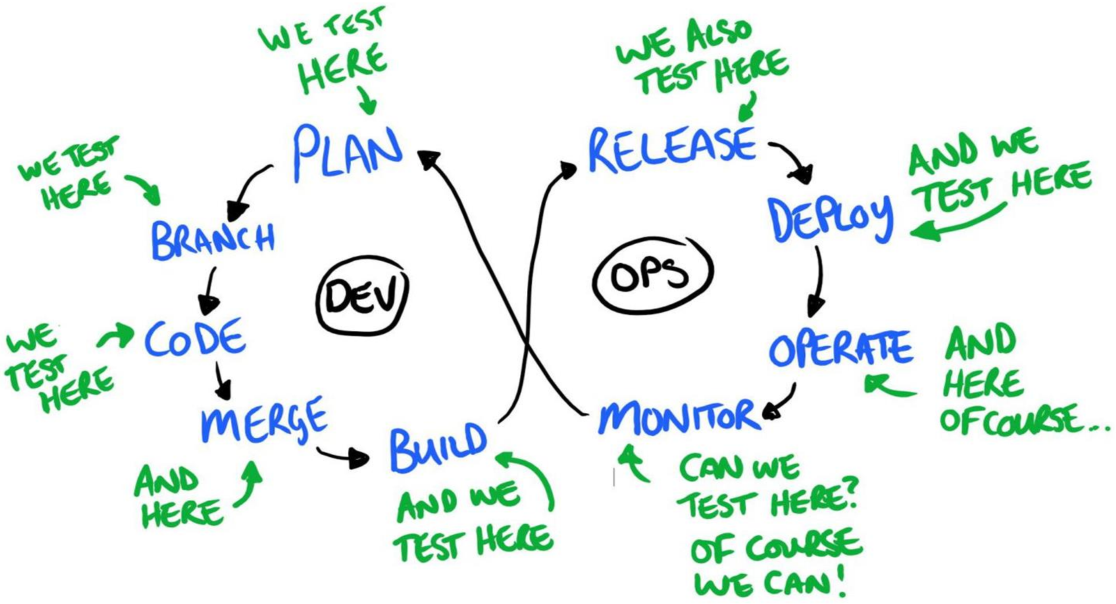
- Pull Requests
- Merges to master

Builds:

- JAR (PUF)
- APK (App)

Deployments:

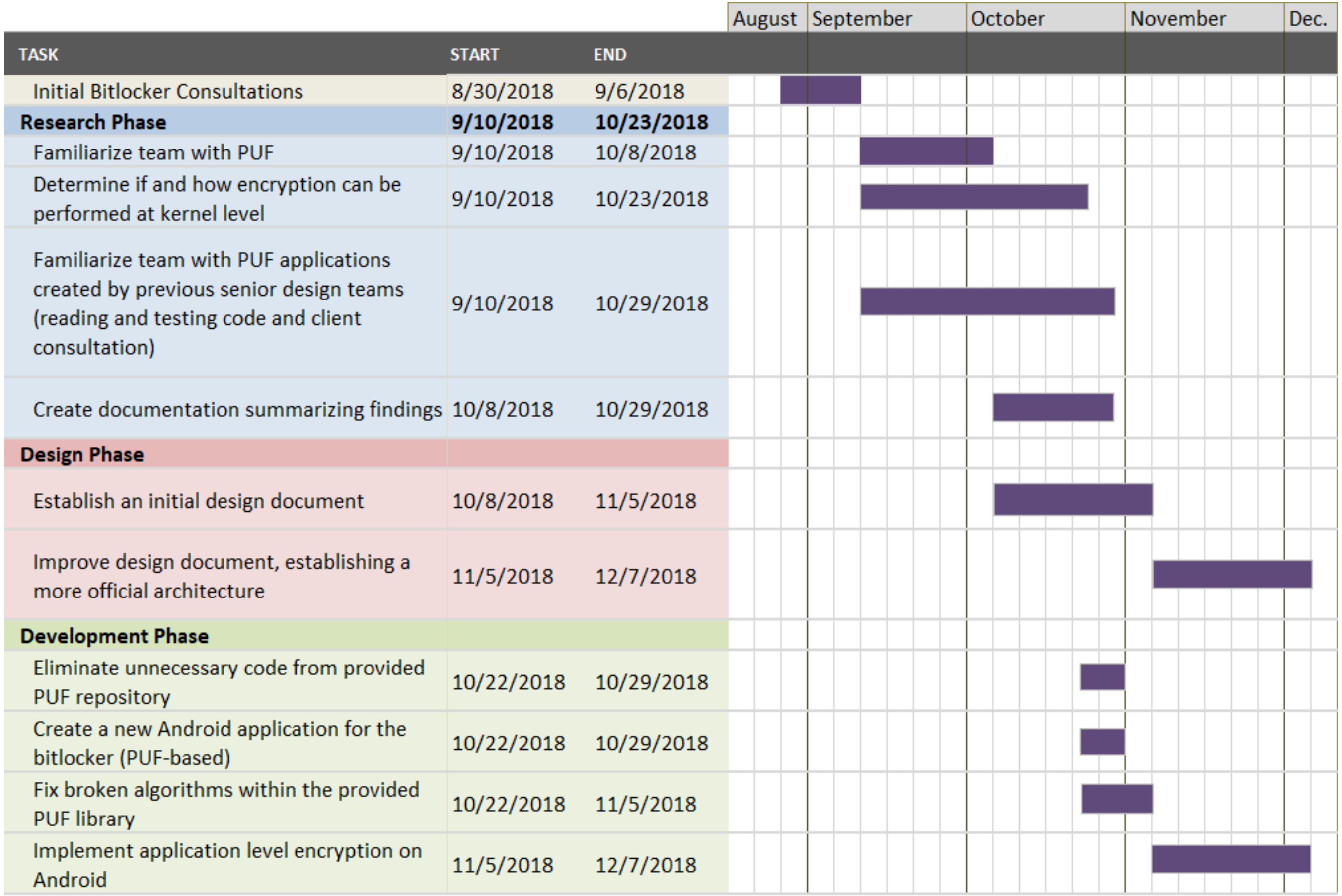
- Google Play Store



Source: https://www.testingexcellence.com/testing-in-devops/

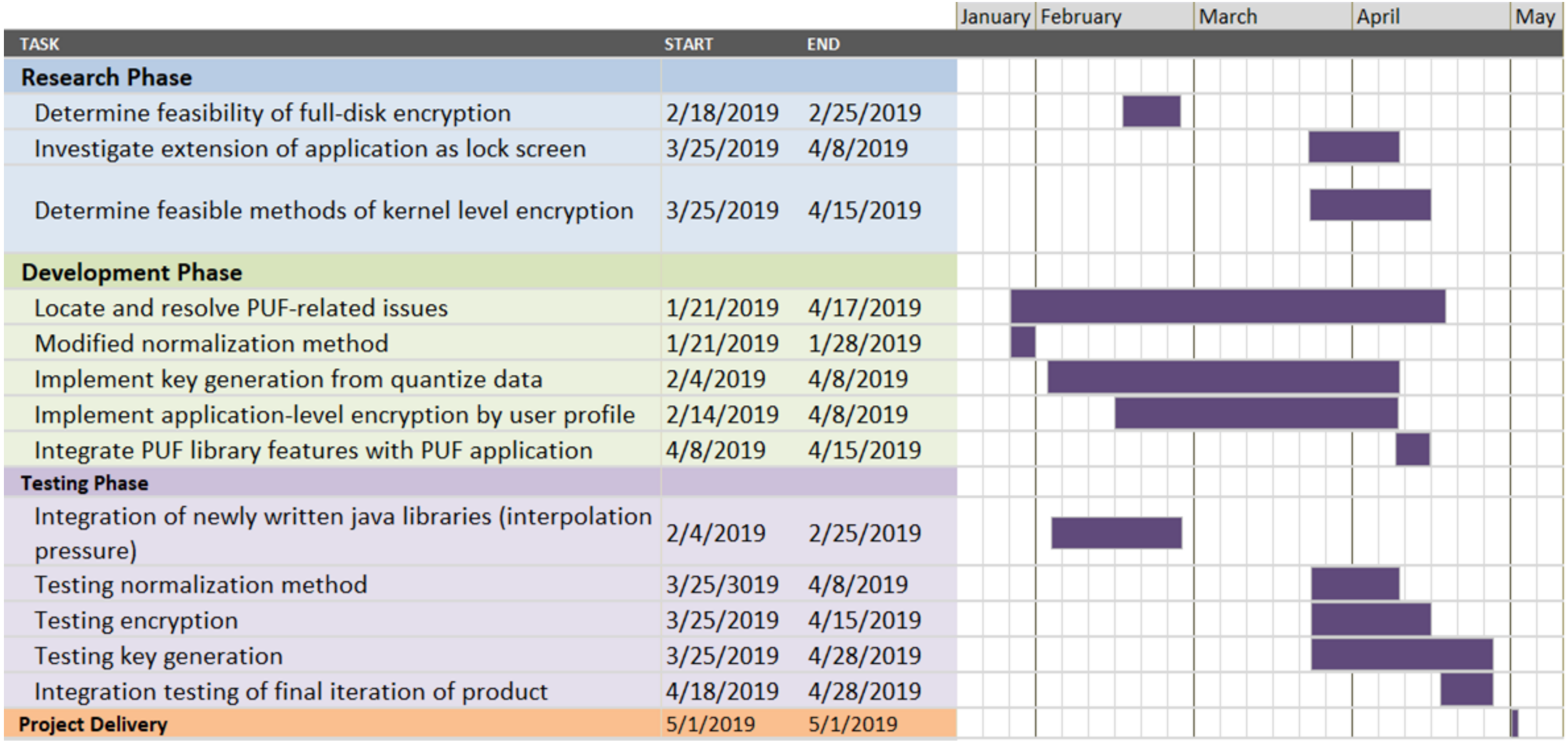
Project and Risk Management

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

Project Schedule - Spring 2019



TASK	START	END
Research Phase		
Determine feasibility of full-disk encryption	2/18/2019	2/25/2019
Investigate extension of application as lock screen	3/25/2019	4/8/2019
Determine feasible methods of kernel level encryption	3/25/2019	4/15/2019
Development Phase		
Locate and resolve PUF-related issues	1/21/2019	4/17/2019
Modified normalization method	1/21/2019	1/28/2019
Implement key generation from quantize data	2/4/2019	4/8/2019
Implement application-level encryption by user profile	2/14/2019	4/8/2019
Integrate PUF library features with PUF application	4/8/2019	4/15/2019
Testing Phase		
Integration of newly written java libraries (interpolation pressure)	2/4/2019	2/25/2019
Testing normalization method	3/25/3019	4/8/2019
Testing encryption	3/25/2019	4/15/2019
Testing key generation	3/25/2019	4/28/2019
Integration testing of final iteration of product	4/18/2019	4/28/2019
Project Delivery	5/1/2019	5/1/2019

Risks and Mitigation

- Integrating preexisting PUF library
  - Library is heavily hard-coded
  - Allocate time

- Inaccurate authentication
  - PUF must be at least 80% accurate
  - The library has multiple methods of normalization, authentication, etc.

- Implementing full-disk encryption
  - Android switched to file-based encryption
  - Workaround is possible through previous versions of Android

Setbacks and Mitigation

- Inaccurate authentication
  - Consulted Technical Advisor, Timothy Dee
  - Remove need for Python interpreter
  - Rewrite Python scripts

- Full-disk encryption infeasible
  - Linux Kernel library “fscrypt”
  - Encryption at application level

Lessons Learned

- Importance of facilitating communication early
- Exploring a provided product in the initial stages of planning before moving forward in the project lifecycle

Conclusion

What did we do?

- Yousef Al-Absi
  - Understanding Gradle
  - Assisted with PUF issues
  - Implemented DevOps
- Cole Alward
  - Implemented encryption
  - Assisted in application integration
  - Organized ticket flow
- Morgan Anderson
  - Implemented key generation
  - Assisted in application integration
  - Aided in technical writing and schedule organization
- Ammar Khan
  - Interacted with client
  - Assisted in rewriting interpolated pressure scripts
  - Aided others with PUF issues
- Justin Kuhn
  - Developing test plan
  - Conducted test integration
  - Assisted in rewriting interpolated pressure scripts
- Larisa Thys
  - Led semi-weekly meetings
  - Assisted in reworking authentication
  - Aided others with PUF issues

Current Project Status

Completed Milestones:

- Completed PUF research
- Completed initial design of project
- Integrated PUF into an application design
- Created application that encrypts and decrypts
- Maintained and updated PUF repository for future use

Going Forward

- Refine PUF library
- Extend Android lock screen API to integrate PUF library
- Implement kernel-level encryption to secure device at boot

Questions?