



13/04/2018

Technical Report

Kryptium

National College of Ireland

BSc in Computing

2017/2018



Aaron Healy
X14532757

Table of Contents

1	Introduction	4
1.1	Background	4
1.2	Aims	5
1.3	Technologies	5
2	System	6
2.1	Purpose	6
2.2	Project Scope	6
2.3	Definitions, Acronyms, and Abbreviations	7
2.4	User Requirements Definition	8
2.5	Requirement Specification	11
2.5.1	Functional requirements	11
2.5.2	Use Case Diagram	11
2.5.3	Requirement 1 <Register/Login >	12
2.5.4	Requirement 2 < Add Password >	14
2.5.5	Requirement 3 < Add Text Block >	17
2.5.6	Requirement 4 < Add File >	19
2.5.7	Requirement 5 < Add Image >	22
2.5.8	Requirement 6 < Decrypt/Delete Image >	24
2.5.9	Requirement 7 < Decrypt/Delete File >	27
2.5.10	Requirement 8 < Decrypt/Delete Password>	29
2.5.11	Requirement 9 < Decrypt/Delete Text Block >	32
2.6	Data Requirements	35
2.7	Environmental Requirements	37
2.8	User Requirements	38

2.9	Usability Requirements	38
2.10	Non-Functional Requirements	39
2.10.1	Performance/Response time requirement	39
2.10.2	Availability requirement	39
2.10.3	Recover requirement	39
2.10.4	Security requirement	40
2.10.5	Reliability requirement	41
2.10.6	Maintainability requirement	41
2.10.7	Reusability requirement	41
2.11	Interface Requirements	42
2.11.1	GUI	42
2.11.2	Application Programming Interfaces (API)	53
2.12	System Architecture	55
2.14	Design and Architecture	57
2.13	Logical Architecture	58
2.15	Implementation	61
2.16	Testing	90
3	Further Development	105
4	Conclusion	105
5	References	106
6	Appendix	107
6.1	Project Proposal	107
6.1.1	Objectives	107
6.1.2	Background	108
6.1.3	Technical Approach	109

6.1.4	Special Resources Needed	111
6.1.5	Technical Details	111
6.1.6	Evaluation	113
6.2	Project Plan	113

Executive Summary

From the research conducted at the beginning of the project development I found there was a severe lack of android applications that allowed users to encrypt various file types and store them securely in a database of any type. Most applications on the Google Play Store that allow encryption of some sort are mostly messenger like applications or focus on one type of file to encrypt such as images. I also found applications using unsecure encryption algorithms such as MD5 as well have a basic or unappealing user interface.

The Kryptium android application will allow users to encrypt and safely store passwords, text blocks, images and files, such as word, pdf and excel files. These will be encrypted using currently secure encryption algorithms such as AES 256 and TwoFish as well as steganography techniques. The user interface will be as simple as possible to allow users of all experience levels to use the application with ease.

A web application will also contain all the functionality of the android application and allow users to encrypt or decrypt their data on either the android application or on a laptop, or anything with a browser.

The android applications database is a Google Firebase Realtime Database which allows users to access their data on both platforms as the database is hosted online not physically on either device.

1 Introduction

The aim of this document is to provide all the technical details of the mobile and web application being developed, named Kryptium. This application will provide multiple encryption features described in the document below. This section also gives a scope description and overview of everything included in the requirement specification document. Also, a list of abbreviations and definitions are provided.

1.1 Background

In my research I found there were very few encryption applications on the Google Play Store that could be considered up to par and usable. I found that most apps were old and had UI that didn't have usability in mind, as well as some apps using non-secure encryption algorithms. While there were one or two good apps in terms of features they had outdated UI and were over complicated with too many options and buttons which probably left some users confused and wanting to uninstall the app. So, I wanted to make an app that had usability, good UI design as well as secure encryption features in mind.

As I am in the Cyber Security stream I wanted to double down on this idea and include as many security and encryption algorithms as possible without making it too complex. In the end I settled for 5 different ways of encrypting data, including the login and register. The user will be able to encrypt passwords, text blocks, files and images so each of these sections in the application will be encrypted using different algorithms. My android phone also has a fingerprint scanner, so I wanted to include that somehow, so I decided it would be used as a method of decrypting a file the user had previously encrypted.

Although this seemed like a good idea I didn't feel like the complexity was there, so I decided I would make it cross platform by building a web application that registered users could log into to view their encrypted files that way and decrypt them on their PC or laptop. I would use the Google Firebase databases for this as it would allow data to be accessed by both mobile devices and PC's over the internet.

1.2 Aims

The aims of this project are to develop an application which will allow them to encrypt their personal or sensitive data using safe and effective encryption algorithms. Their data will then be stored in a Firebase Database or in Firebase Storage. The user's data can then be decrypted and viewed or downloaded on either an android device or using the web application. The application, on both the web and android devices, will allow users to create a Kryptium account and login at any time. Strong passwords and sessions will keep the user's activity and data safe, also the system will only allow one account per email address. The logged in user can encrypt and decrypt passwords, text blocks, images and files using the application. Each section of the application will use a different encryption algorithm, but all algorithms will create secure ciphertexts and mitigate any attacks, such as dictionary attacks. Both the android and web application will be security tested before it is ever released.

1.3 Technologies

The mobile application will be built for devices using Java for the backend and XML for the user interface. I will be using Android Studio to develop the application as it allows me to test the app on my phone but also run emulators to test features. The web application will be developed in Visual Studios using HTML, CSS, C# and JavaScript.

All the user's data will be stored in a Google Firebase Realtime Database and will be accessed through the Firebase API's. The user's images and files will be stored in Google Firebase Storage and the account creation and login functionality will be created and handled by the Google Firebase Authentication API.

2 System

2.1 Purpose

The purpose of this document is to give a detailed description of all the requirements for the Kryptium cross-platform application. It will illustrate the purpose and need for the development of the application. It will also explain the interface and interactions with external applications and systems. This document is primarily intended to breakdown and showcase the application.

2.2 Project Scope

Kryptium is an encryption-based android and web application which allows people to easily and securely encrypt and save private information or files, such as passwords, addresses, personal images or private files. The android application should be close to completion while the web application being closer to a prototype application.

Users will have to first create an account using either email and password or the Google sign in API, as only authenticated users can access the application and use it. The authenticated users can then encrypt private information, this information is stored in a Google Firebase Database and Google Firebase Storage.

The application will need an internet connection to be able to upload data to the database but also to fetch and display the data in the application. All system information is maintained in a database, which is a Google Firebase Database, while all images and files will be stored in Google Firebase Storage. The android application will also need permission to use the fingerprint-scanner, if the mobile device has one, this allows users to delete or decrypt data with their fingerprint.

The web application and android application will essentially be copies of each other allowing users to log in and encrypt or decrypt data on both platforms.

2.3 Definitions, Acronyms, and Abbreviations

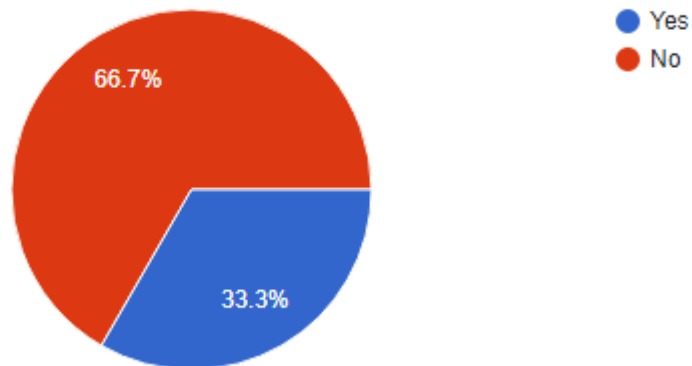
Term	Definition
User	Someone who interacts with the web/mobile application
Firebase Authentication	System used to verify users as well as Login/Register functionality.
Firebase Database	System used to store all the user data (Strings, int)
Firebase Storage	System used to store files (pdf, png, jpg)
AES 256	Advanced Encryption Standard using 256-bit key, used to encrypt the user's passwords.
Twofish	Encryption algorithm to encrypt files
Steganography	Used to hide text in an image which is uploaded to database.
3DES	Encryption algorithm to encrypt images
GUI	Graphical User Interface, the applications front-end.
API	Application Programming Interface.

2.4 User Requirements Definition

To get a better understanding of what customers need or want in this application I conducted a short survey containing questions I felt were important. I created the survey using Google Forms. Google Forms allowed me to create a survey online and then share the survey with people on Facebook or by email. This platform also generates the pie charts you see below and updates them every time a survey is complete. Below is a breakdown of the results of this survey along with all the questions that were asked.

Have you ever used an encryption application before?

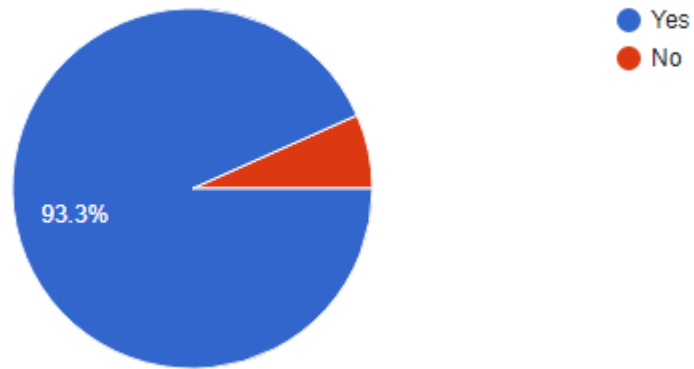
15 responses



I felt this was important because it shows only 33 percent of the survey applicants have used an encryption application or service at some stage.

Would you consider using one to store private or sensitive data?

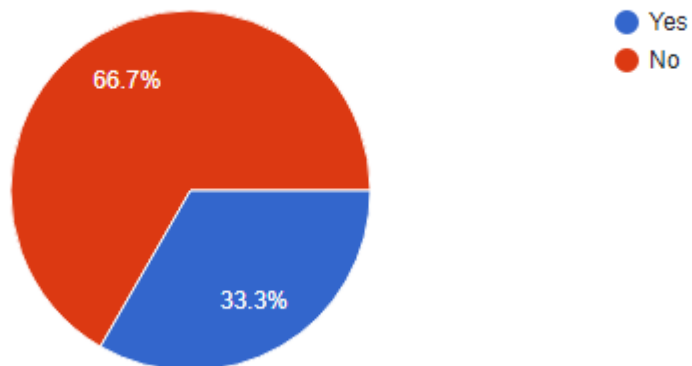
15 responses



From this graph and the one above we can see that most people haven't used an encryption application but would consider using one, whether it be for personal data or company data they hold.

The application will need an internet connection to access your data. Does this affect your decision to use this application?

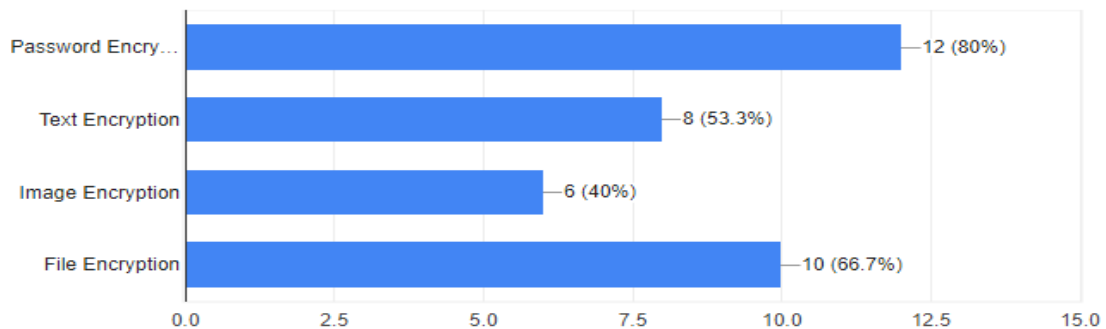
15 responses



Although more people said it doesn't affect their decision to possibly use my application the high number of people that said it would has made me consider some sort of offline capabilities.

What features would you want to be in this application?

15 responses



From this graph we can see that most people are interested in keeping their passwords and files safe and secure.

What would make you use this application over a competitor?

15 responses



The biggest attraction for users is that their data is in fact secure and encrypted properly. But also, users like unique features, the one I mentioned in the survey was the fact that I would be incorporating the fingerprint scanner on their mobile devices.

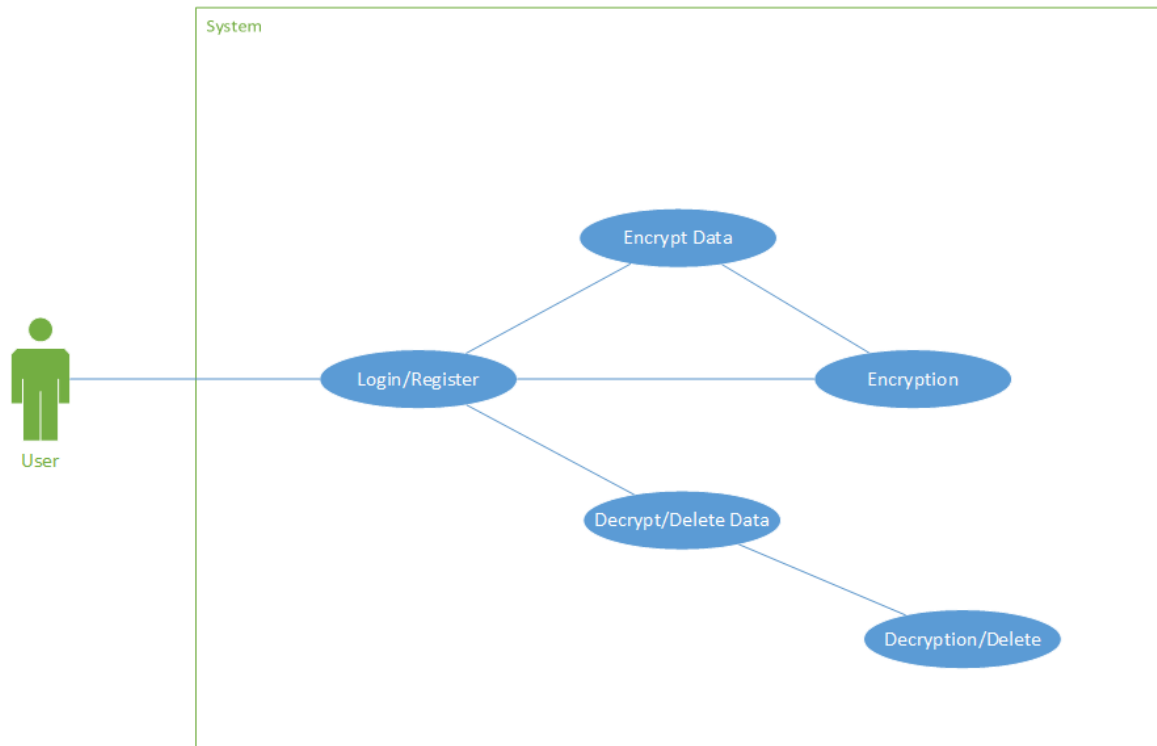
2.5 Requirement Specification

This section contains all the functional requirements of the system. It gives a detailed description of the system and all the features as well as how a user interacts with these features.

2.5.1 Functional requirements

This section lists the functional requirements in ranked order. Functional requirements describe the possible effects of a software system, in other words, *what* the system must accomplish.

2.5.2 Use Case Diagram



2.5.3 Requirement 1 <Register/Login >

2.5.3.1 Description & Priority

This requirement is essential because only authenticated users can login and use the features in the application. Users must first be able to register an account and then login. The user's information must be saved to Firebase Auth and the Firebase Database.

2.5.3.2 Use Case

Identification Code

FR1

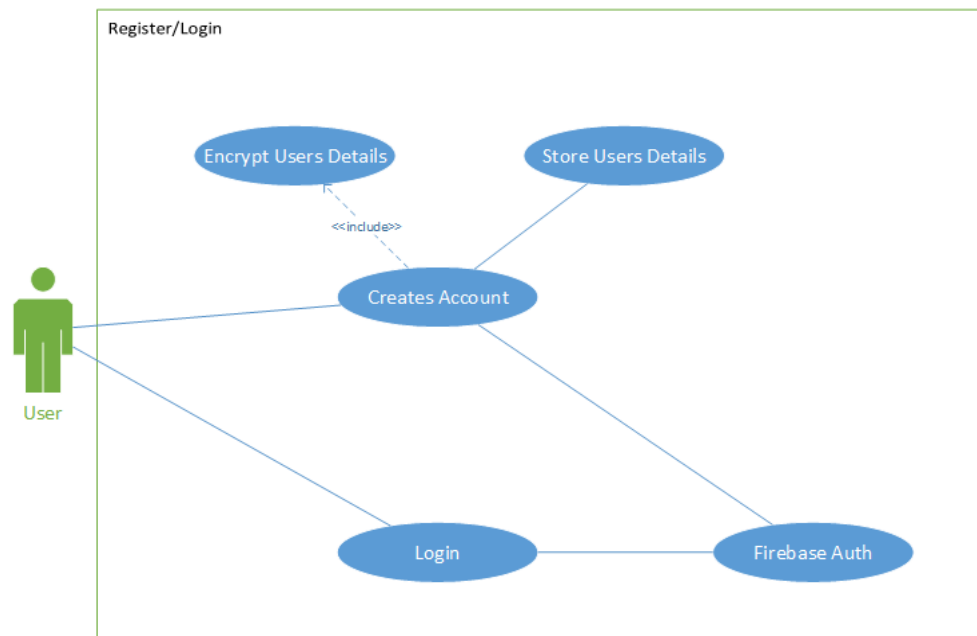
Scope

The scope of this use case is to show the interactions between the user and the application as well as key parts of the application interacting with each other.

Description

This use case describes the user registering an account and then logging in, if the user is already registered they can skip straight the login.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode, user brought to login screen.

Activation

This use case starts when a user selects register and is brought to the register screen.

Main flow

1. The user clicks the register button, the system identifies the register button has been selected.
2. The user fills in the text boxes on the screen, all must be filled.
3. The user clicks the register button.
4. The system encrypts the user's credentials and details using AES.
5. The system connected to Firebase Authentication and Database.
6. The system saves the users credentials.
7. The user is brought to the login screen.
8. The user can enter their email and password and brought to the home screen where they can use the features of the application.

Alternate flow

A1: <Passwords don't match >

1. The system compares password and confirm password textboxes.
2. The system notifies user passwords don't match
3. The use case continues at position 2 of the main flow

A2: <Email check >

1. The system checks the email address isn't already in use.
2. If email already in use, the system notifies user the email is already taken so must either use another email address or recover the account.
3. The use case continues at position 2 of the main flow

A3: <Textbox check >

1. The system checks all textboxes are filled correctly
2. The system notifies if a textbox is empty or not filled in correctly.
3. The use case continues at position 2 of the main flow

A4: <User already has an account >

1. The use case continues at position 6 of the main flow

Termination

If register is successful user is brought to the login page, if the user logs in the login/register process terminates.

Post condition

Success Conditions:

1. User creates an account.
2. User can login to application.
3. User details uploaded to Database and Authentication.

Failure conditions:

1. User cannot create an account.
2. User cannot login to application.
3. User details not uploaded to Database and/or Authentication.

2.5.4 Requirement 2 < Add Password >

2.5.4.1 Description & Priority

This use case describes a user adding a new password to the database, being able to view all passwords in the database and can add passwords. The user's passwords must be saved to the Firebase Database as well as be read from this database.

2.5.4.2 Use Case

Identification Code

FR2

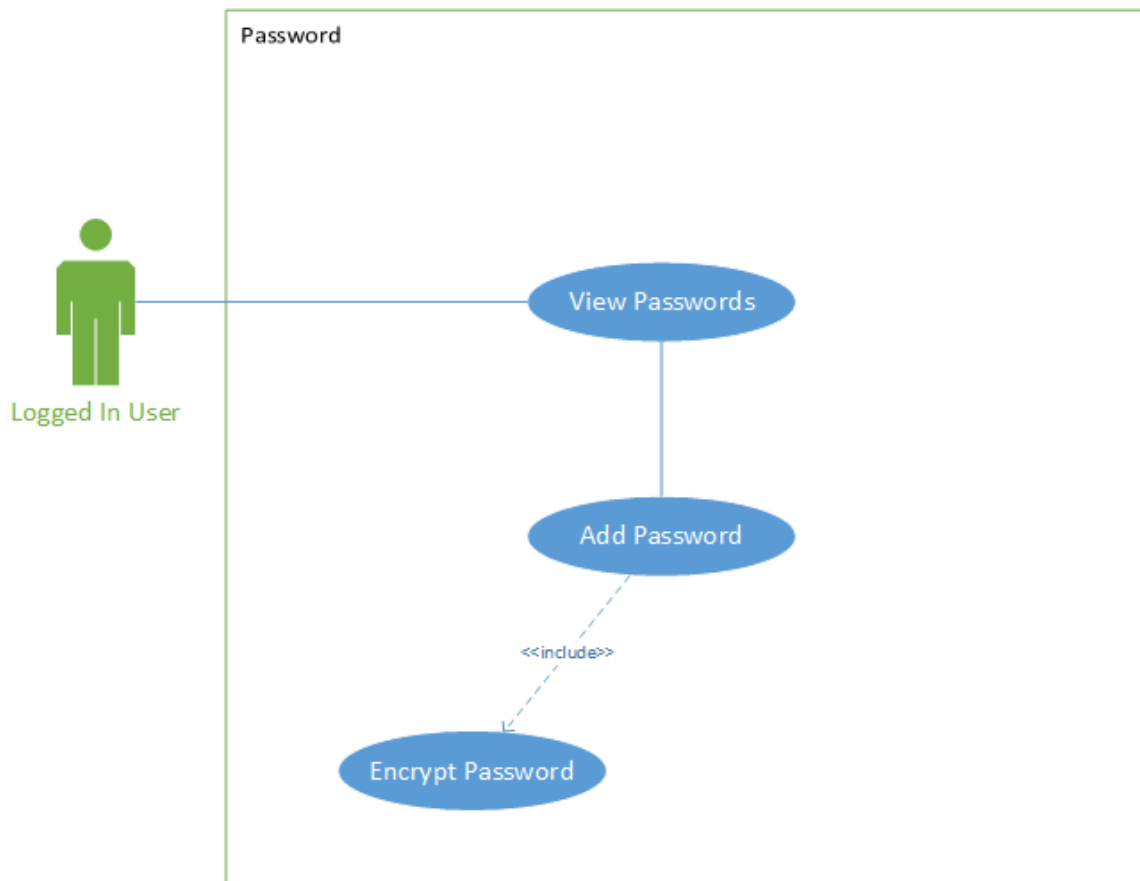
Scope

The scope of this use case is to show how a user views and adds a password to the database

Description

This use case describes how a user views and can add a new password to the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view passwords. The system shows all passwords in the database in a list.

Activation

This use case starts when a user clicks “create new password” button.

Main flow

1. The system identifies the “create new password” button has been clicked by the user.
2. The system brings the user to the page where the user can create a new password
3. The user fills in all textboxes on the page (name, password, pin).

4. The user clicks “upload” button.
5. The system connected with the Firebase Database.
6. The data from the textboxes is encrypted and saved to the database.
7. The system goes back to the View Passwords page.

Alternate flow

A1: <Textboxes empty/not filled correctly >

1. The system checks all textboxes are filled correctly.
2. The system notifies the user to fill the textboxes correctly.
3. The use case continues at position 3 of the main flow

A2: <Password name already exists >

1. The system checks password name does not already exist.
2. The system notifies the user create a new password name.
3. The use case continues at position 3 of the main flow

A3: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully saves the password and user is brought back to view passwords page.

Post condition

Success Conditions:

1. Password is encrypted.
2. Password is uploaded to Database.

Failure conditions:

1. Password fails to encrypt.
2. Password fails to upload to Database.

2.5.5 Requirement 3 < Add Text Block >

2.5.5.1 Description & Priority

This use case describes a user adding a new text block to the database, being able to view all new text blocks in the database. The user's new text blocks must be saved to the Firebase Database as well as be read from this database.

2.5.5.2 Use Case

Identification Code

FR3

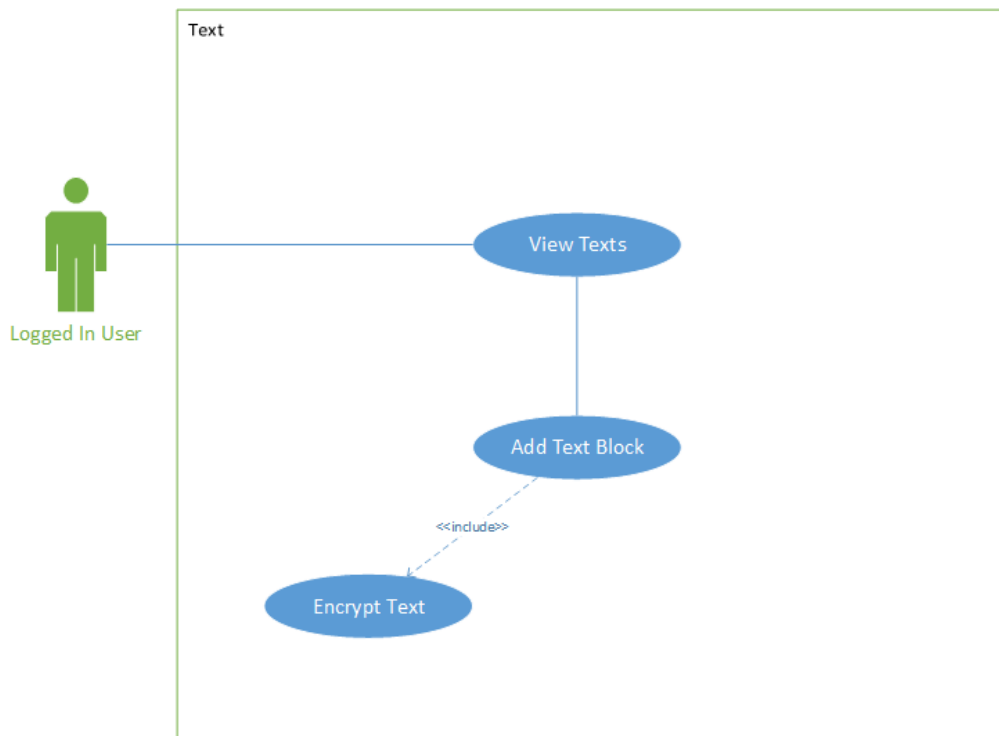
Scope

The scope of this use case is to show how a user views and adds a new text block.

Description

This use case describes how a user views and adds a new text block to the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view new text blocks. The system shows all passwords in the database in a list.

Activation

This use case starts when a user clicks “create new password” button

Main flow

1. The system identifies the “create new text block” button has been clicked by the user.
2. The system brings the user to the page where the user can create a new text block.
3. The user fills in all textboxes on the page (name, text, pin).
4. The user clicks “upload” button.
5. The system connected with the Firebase Database.
6. The data from the textboxes is encrypted and saved to the database.
7. The system goes back to the View Passwords page.

Alternate flow

A1: <Textboxes empty/not filled correctly >

1. The system checks all textboxes are filled correctly.
2. The system notifies the user to fill the textboxes correctly.
3. The use case continues at position 3 of the main flow

A2: < new text block name already exists >

1. The system checks new text block name does not already exists.
2. The system notifies the user create a new password name.
3. The use case continues at position 3 of the main flow

A1: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully saves the text and user is brought back to view passwords page.

Post condition

Success Conditions:

1. Text Block is encrypted.
2. Text Block is uploaded to Database.

Failure conditions:

1. Text Block fails to encrypt.
2. Text Block fails to upload to Database.

2.5.6 Requirement 4 < Add File >

2.5.6.1 Description & Priority

This use case describes a user adding a new file to the database and storage, being able to view all files in the database. The user's files must be saved to the Firebase Database as well as be read from this database.

2.5.6.2 Use Case

Identification Code

FR4

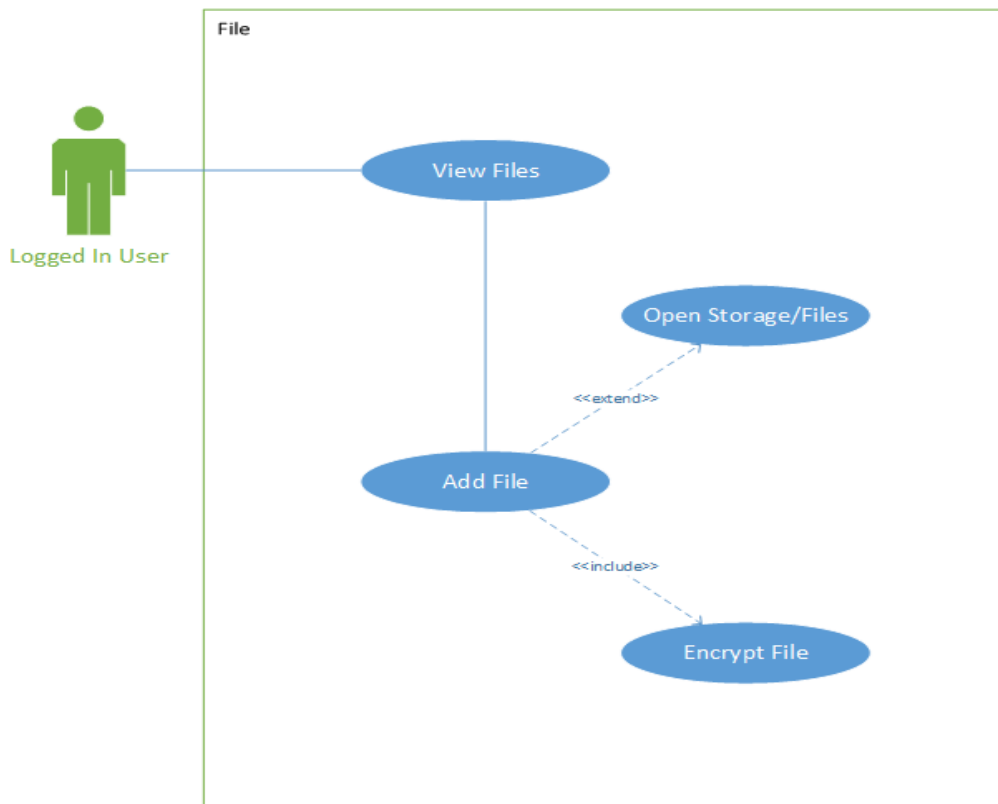
Scope

The scope of this use case is to show how a user views and adds a file.

Description

This use case describes how a user views and adds a file to the database

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view files. The system shows all files in the database in a list.

Activation

This use case starts when a user clicks “create new file” button

Main flow

1. The system identifies the “create new file” button has been clicked by the user.
2. The system brings the user to the page where the user can create a new file upload.
3. The user fills in all textboxes on the page (name, file description, pin).
4. The user clicks the “select file” button.
5. The user clicks “upload” button.

6. The system connected with the Firebase Database.
7. The data from the textboxes and the file is encrypted and saved to the database.
8. The system goes back to the View files page.

Alternate flow

A1: <Textboxes empty/not filled correctly >

1. The system checks all textboxes are filled correctly.
2. The system notifies the user to fill the textboxes correctly.
3. The use case continues at position 3 of the main flow

A2: < file name already exists >

1. The system checks file name already exists.
2. The system notifies the user create a new file name.
3. The use case continues at position 3 of the main flow

A3: <No file selected >

1. The user hasn't selected a file.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

A4: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully saves the file and user is brought back to view files page.

Post condition

Success Conditions:

1. File is encrypted.
2. File is upload to Storage.
3. File data is uploaded to Database.

Failure conditions:

4. File fails to encrypt.
5. File fails to upload to Storage.
6. File data fails to upload to Database.

2.5.7 Requirement 5 < Add Image >

2.5.7.1 Description & Priority

This use case describes a user adding a new image to the database and storage, being able to view all images in the database. The user's images must be saved to the Firebase Database as well as be read from this database.

2.5.7.2 Use Case

Identification Code

FR5

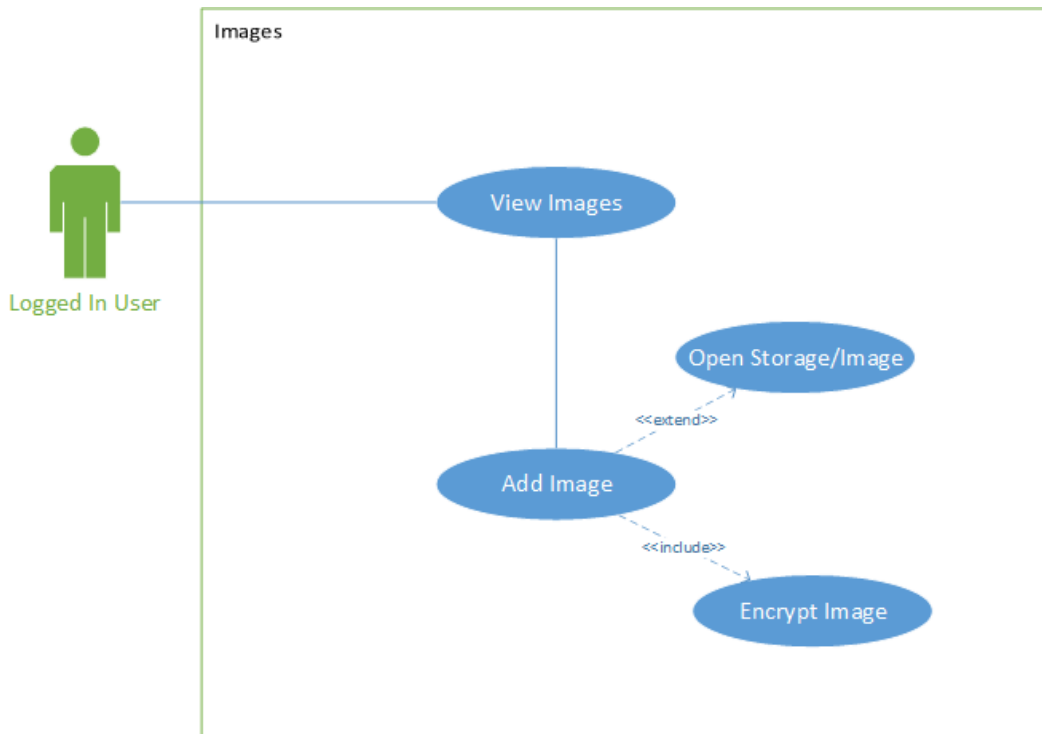
Scope

The scope of this use case is to show how a user views and adds an image.

Description

This use case describes how a user views and adds an image to the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view images. The system shows all images in the database in a list.

Activation

This use case starts when a user clicks “create new image” button

Main flow

1. The system identifies the “create new image” button has been clicked by the user.
2. The system brings the user to the page where the user can create a new image upload.
3. The user fills in all textboxes on the page (name, image description, pin).
4. The user clicks the “select image” button.
5. The user clicks “upload” button.
6. The system connected with the Firebase Database.
7. The data from the textboxes and the image is encrypted and saved to the database.
8. The system goes back to the View Images page.

Alternate flow

A1: <Textboxes empty/not filled correctly >

1. The system checks all textboxes are filled correctly.
2. The system notifies the user to fill the textboxes correctly.
3. The use case continues at position 3 of the main flow

A2: < Image name already exists >

1. The system checks file name already exists.
2. The system notifies the user create a new file name.
3. The use case continues at position 3 of the main flow

A3: <No Image selected >

1. The user hasn't selected an image.
2. The system notifies the user with error message.
3. The use case continues at position 4 of the main flow.

A4: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully saves the image and user is brought back to view images page.

Post condition

Success Conditions:

1. Image is encrypted.
2. Image is upload to Storage.
3. Image data is uploaded to Database.

Failure conditions:

1. Image fails to encrypt.
2. Image fails to upload to Storage.
3. Image data fails to upload to Database.

2.5.8 Requirement 6 < Decrypt/Delete Image >

2.5.8.1 Description & Priority

This use case describes a user decrypting or deleting an image from the database and storage. The user's images must be read or deleted from the Firebase Database and Storage as well as be read from this database.

2.5.8.2 Use Case

Identification Code

FR6

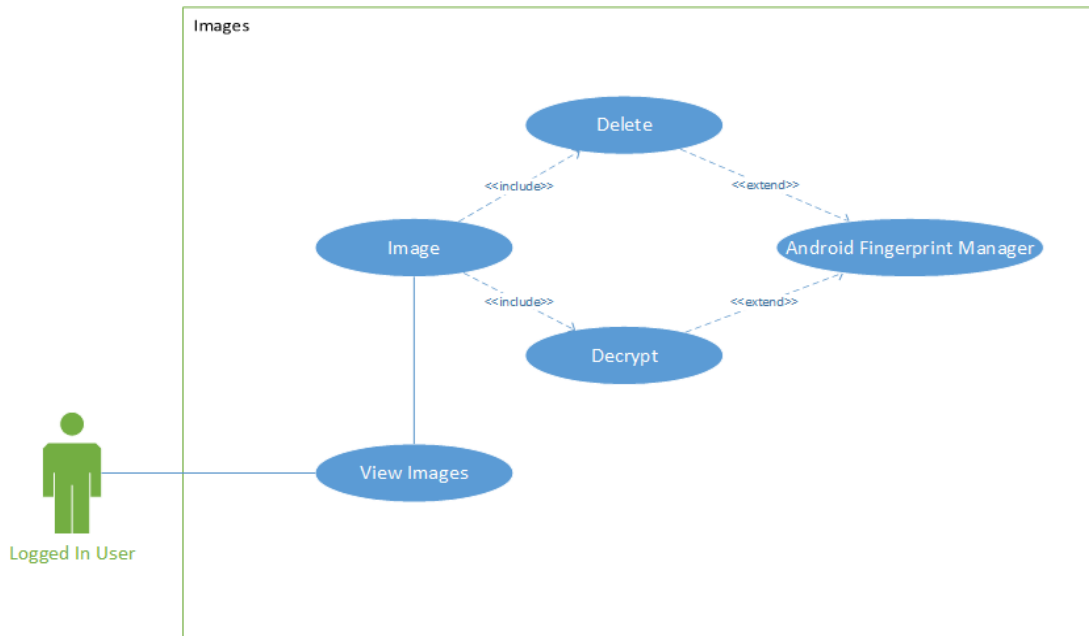
Scope

The scope of this use case is to show how a user views and either decrypts or deletes images.

Description

This use case describes how a user views and delete or decrypt an image to the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view images. The system shows all images in the database in a list.

Activation

This use case starts when a user clicks on one of the items in the list.

Main flow

1. The system identifies an item in the list has been clicked
2. The system brings the user to the page where the user can decrypt or delete an image from the database.
3. The user can select to use a pin code or their fingerprint.
4. Pin Code Alternate Flow<A1>.
5. Fingerprint Alternate Flow<A2>.
6. The user selects the delete or decrypt button.
7. The system connected with the Firebase Database.
8. The image is deleted or downloaded after being decrypted.
9. The system goes back to the View Images page.

Alternate flow

A1: <User Selects to enter a pin code >

1. The pin code textbox appears.
2. The user enters a pin code.
3. The use case continues at position 6 of the main flow

A2: < User Selects to use their fingerprint>

1. Fingerprint dialog appears.
2. User places finger on scanner.
3. The use case continues at position 6 of the main flow

A3: <Pin code is incorrect >

1. The user enters a pin code.
2. The system notifies the user with error message as pin code is not correct.
3. The use case continues at position 4 of the main flow.

A4: <Fingerprint doesn't match stored fingerprint >

1. The user places their finger on the scanner.
2. The system notifies the user with error message as fingerprint is not the same as stored fingerprint.
3. The use case continues at position 5 of the main flow.

A5: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully deletes or decrypts the image and user is brought back to view images page.

Post condition

Success Conditions:

1. Image is deleted from the database.
2. Image is decrypted.
3. Image is downloaded.

Failure conditions:

1. Image fails to delete.
2. Image fails to decrypt.
3. Image fails to download.

2.5.9 Requirement 7 < Decrypt/Delete File >

2.5.9.1 Description & Priority

This use case describes a user decrypting or deleting a file from the database and storage. The user's files must be decrypted or deleted from the Firebase Database and Storage as well as be read from this database.

2.5.9.2 Use Case

Identification Code

FR7

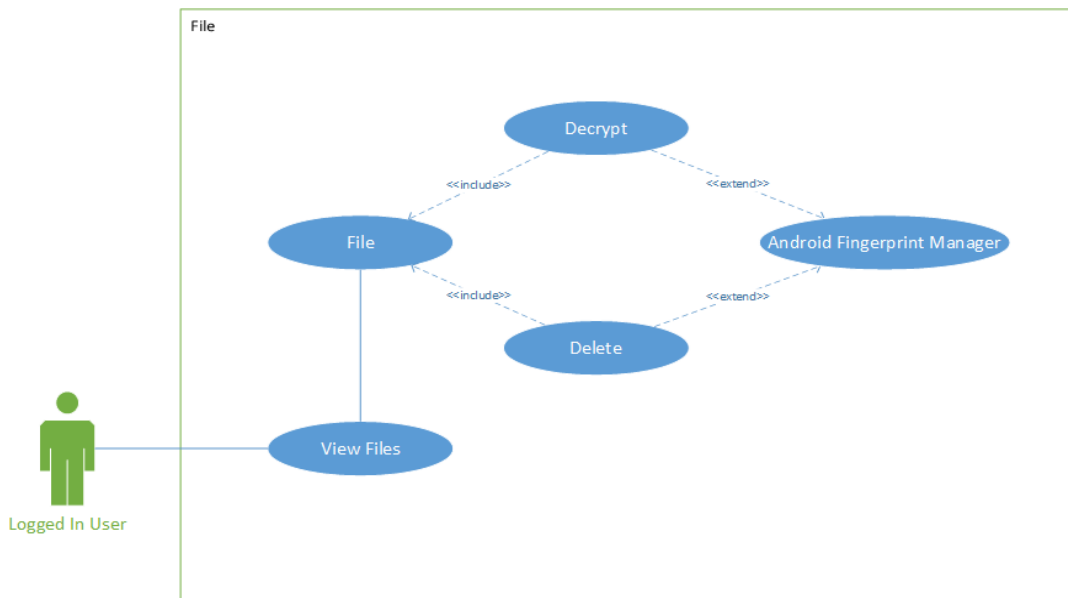
Scope

The scope of this use case is to show how a user views and either decrypts or deletes files.

Description

This use case describes how a user views and delete or decrypt a file to the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view files. The system shows all files in the database in a list.

Activation

This use case starts when a user clicks on one of the items in the list.

Main flow

1. The system identifies an item in the list has been clicked
2. The system brings the user to the page where the user can decrypt or delete a file from the database.
3. The user can select to use a pin code or their fingerprint.
4. Pin Code Alternate Flow<A1>.
5. Fingerprint Alternate Flow<A2>.
6. The user selects the delete or decrypt button.
7. The system connected with the Firebase Database.
8. The file is deleted or downloaded after being decrypted.
9. The system goes back to the View Files page.

Alternate flow

A1: <User Selects to enter a pin code >

1. The pin code textbox appears.
2. The user enters a pin code.
3. The use case continues at position 6 of the main flow

A2: < User Selects to use their fingerprint>

1. Fingerprint dialog appears.
2. User places finger on scanner.
3. The use case continues at position 6 of the main flow

A3: <Pin code is incorrect >

1. The user enters a pin code.
2. The system notifies the user with error message as pin code is not correct.
3. The use case continues at position 4 of the main flow.

A4: <Fingerprint doesn't match stored fingerprint >

1. The user places their finger on the scanner.
2. The system notifies the user with error message as fingerprint is not the same as stored fingerprint.

3. The use case continues at position 5 of the main flow.
A5: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully deletes or decrypts the file and user is brought back to view files page.

Post condition

Success Conditions:

1. File is deleted from the database.
2. File is decrypted.
3. File is downloaded.

Failure conditions:

1. File fails to delete.
2. File fails to decrypt.
3. File fails to download.

2.5.10 Requirement 8 < Decrypt/Delete Password>

2.5.10.1 Description & Priority

This use case describes a user decrypting or deleting a password from the database. The user's passwords must be decrypted or deleted from the Firebase Database as well as be read from this database.

2.5.10.2 Use Case

Identification Code

FR8

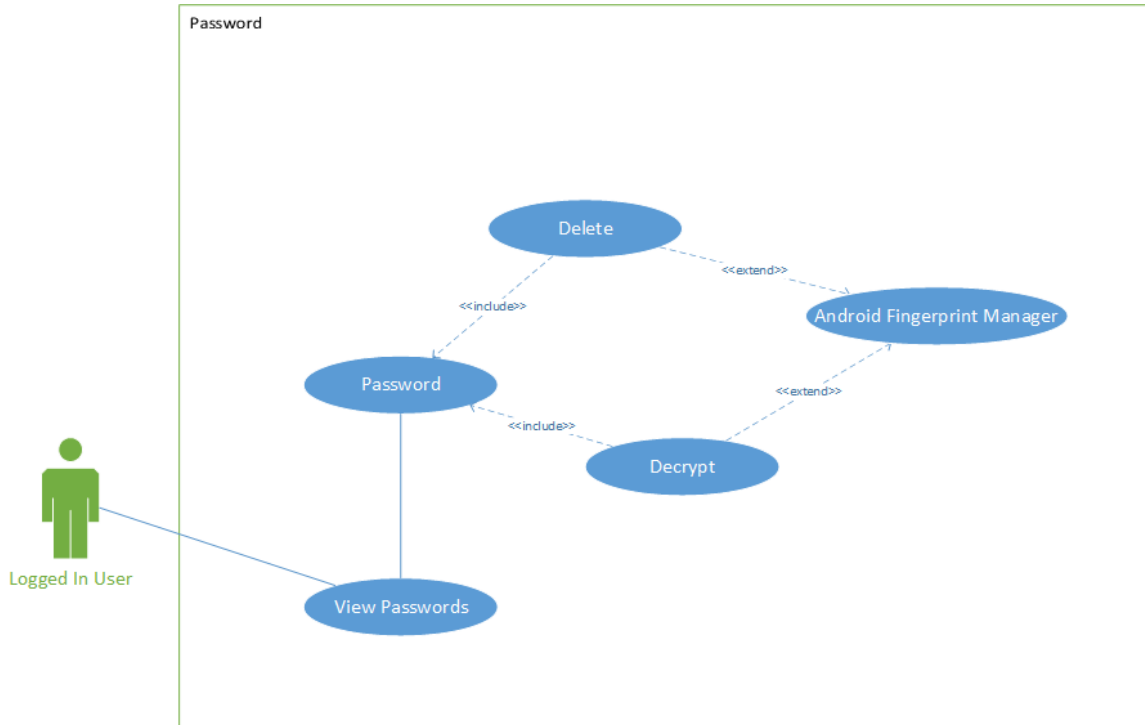
Scope

The scope of this use case is to show how a user views and either decrypts or deletes passwords.

Description

This use case describes how a user views and delete or decrypt a password from the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view passwords. The system shows all passwords in the database in a list.

Activation

This use case starts when a user clicks on one of the items in the list.

Main flow

1. The system identifies an item in the list has been clicked
2. The system brings the user to the page where the user can decrypt or delete a password from the database.
3. The user can select to use a pin code or their fingerprint.
4. Pin Code Alternate Flow<A1>.
5. Fingerprint Alternate Flow<A2>.
6. The user selects the delete or decrypt button.
7. The system connected with the Firebase Database.
8. The password is deleted or shown after being decrypted.

9. The system goes back to the View Password page.

Alternate flow

A1: <User Selects to enter a pin code >

1. The pin code textbox appears.
2. The user enters a pin code.
3. The use case continues at position 6 of the main flow

A2: < User Selects to use their fingerprint>

1. Fingerprint dialog appears.
2. User places finger on scanner.
3. The use case continues at position 6 of the main flow

A3: <Pin code is incorrect >

1. The user enters a pin code.
2. The system notifies the user with error message as pin code is not correct.
3. The use case continues at position 4 of the main flow.

A4: <Fingerprint doesn't match stored fingerprint >

1. The user places their finger on the scanner.
2. The system notifies the user with error message as fingerprint is not the same as stored fingerprint.
3. The use case continues at position 5 of the main flow.

A5: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully deletes or decrypts the password and user is brought back to view passwords page.

Post condition

Success Conditions:

1. Password is deleted from the database.
2. Password is decrypted.
3. Password is displayed in application.

Failure conditions:

1. Password fails to delete.
2. Password fails to decrypt.
3. Password is displayed in application.

2.5.11 Requirement 9 < Decrypt/Delete Text Block >

2.5.11.1 Description & Priority

This use case describes a user decrypting or deleting a text block from the database. The user's text blocks must be decrypted or deleted from the Firebase Database as well as be read from this database.

2.5.11.2 Use Case

Identification Code

FR9

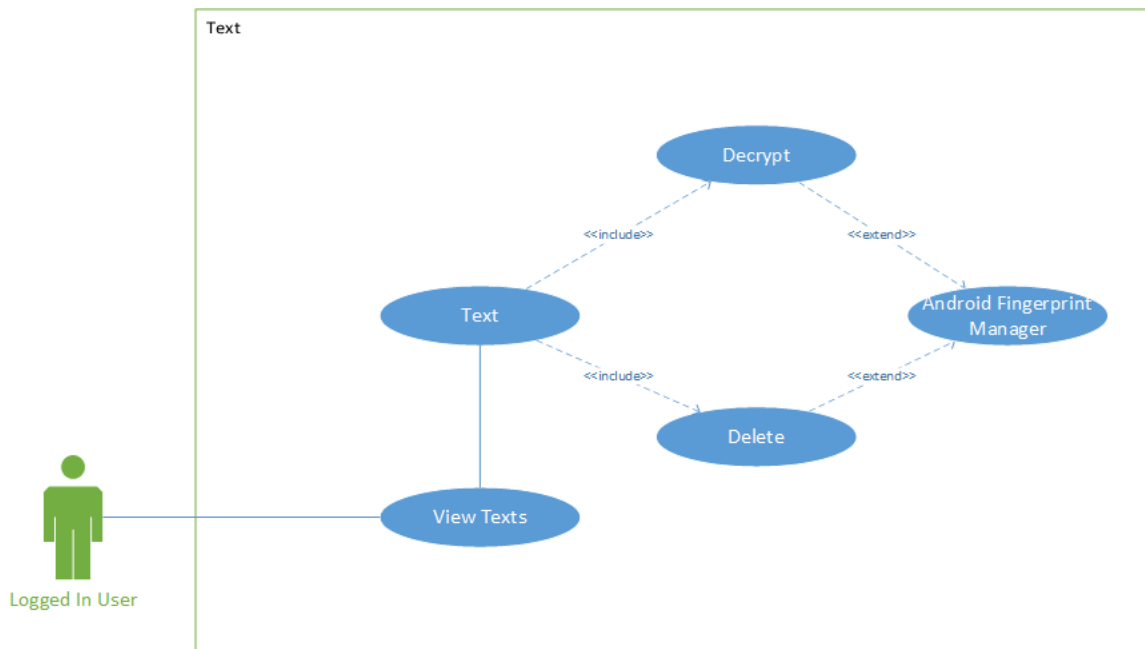
Scope

The scope of this use case is to show how a user views and either decrypts or deletes text blocks.

Description

This use case describes how a user views and delete or decrypt a text block from the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialization mode when the user clicks the button on the applications home screen to view text blocks. The system shows all text blocks in the database in a list.

Activation

This use case starts when a user clicks on one of the items in the list.

Main flow

1. The system identifies an item in the list has been clicked
2. The system brings the user to the page where the user can decrypt or delete a text block from the database.
3. The user can select to use a pin code or their fingerprint.
4. Pin Code Alternate Flow<A1>.
5. Fingerprint Alternate Flow<A2>.
6. The user selects the delete or decrypt button.
7. The system connected with the Firebase Database.
8. The text block is deleted or shown after being decrypted.
9. The system goes back to the View text block page.

Alternate flow

A1: <User Selects to enter a pin code >

1. The pin code textbox appears.
2. The user enters a pin code.
3. The use case continues at position 6 of the main flow

A2: < User Selects to use their fingerprint>

1. Fingerprint dialog appears.
2. User places finger on scanner.
3. The use case continues at position 6 of the main flow

A3: <Pin code is incorrect >

1. The user enters a pin code.
2. The system notifies the user with error message as pin code is not correct.
3. The use case continues at position 4 of the main flow.

A4: <Fingerprint doesn't match stored fingerprint >

1. The user places their finger on the scanner.

2. The system notifies the user with error message as fingerprint is not the same as stored fingerprint.
3. The use case continues at position 5 of the main flow.

A5: <Failed to connect to Database >

1. The system can't create a connection to the database.
2. The system notifies the user with error message.
3. The use case continues at position 3 of the main flow.

Termination

The system successfully deletes or decrypts the text block and user is brought back to view text block page.

Post condition

Success Conditions:

1. Text Block is deleted from the database.
2. Text Block is decrypted.
3. Text Block is displayed in application.

Failure conditions:

1. Text Block fails to delete.
2. Text Block fails to decrypt.
3. Text Block is displayed in application.

2.6 Data Requirements

In this section I will explain and describe the data requirements of the application which are essential to the applications features mentioned above.

Firestore Realtime Database

The applications database is a Firestore Realtime Database which is a cloud-hosted NoSQL database. It stores and syncs data between the database and the user's device. All the user's details will be stored here as well as all the users encrypted passwords, text blocks, files and images. The database contains five tables Users, Passwords, Text Blocks, Files and Images. The user must be able to add data to the database and the application must be able to sync with the database to show all the users data in the application, as well as allow for the decryption or the deletion of the user's data. The structure of the database along with the rules of the database allow the user to only see their own data and no one else's. Part of the database and its structure can be seen below in figure1.



Figure 1 - Firestore Database

Firebase Authentication

Firebase Authentication is used to allow the user to create an account and allow the user to then login. The Auth API handles the creation of the account and the login functionality. Firebase Auth creates the accounts User UID which is a unique identification code for each account. Auth is also important and must always work because the database rules allow only authenticated accounts to read and write to the database.

Identifier	Providers	Created	Signed In	User UID ↑
464healy@gmail.com		28 Nov 2017	28 Nov 2017	HldMxDM8O4YG8vrZkloim7bdU5i1
test@gmail.com		27 Nov 2017	28 Nov 2017	hFICmgNMFxftGyt5ZGybf1M3p83

Figure 2 - User Credentials

Firebase Authentications handles the user's password by hashing and salting it using scrypt as seen in figure 3.

Password hash parameters ×

These parameters are sensitive information for user account security. Be sure to keep them private.

The information below can be used to migrate password users.

```
hash_config {
  algorithm: SCRYPT,
  base64_signer_key: os3YnH3IeIaIJu8hkBuyX/81NRn1xNauFk1+TWNbk7aUoVZH95tS/BdUz60GEYCnftA1krU21ER3NUVL0KyYTg==,
  base64_salt_separator: Bw==,
  rounds: 8,
  mem_cost: 14,
}
```

[COPY](#)

Figure 3 - Firebase Password Hashing

Firestore Storage

Firestore Storage is used to store the actual images and files encrypted and uploaded by the user. A link, the image or files downloadURL, is stored along with the image or files corresponding data in the database. The three folders used to store the different files can be seen in figure 4.

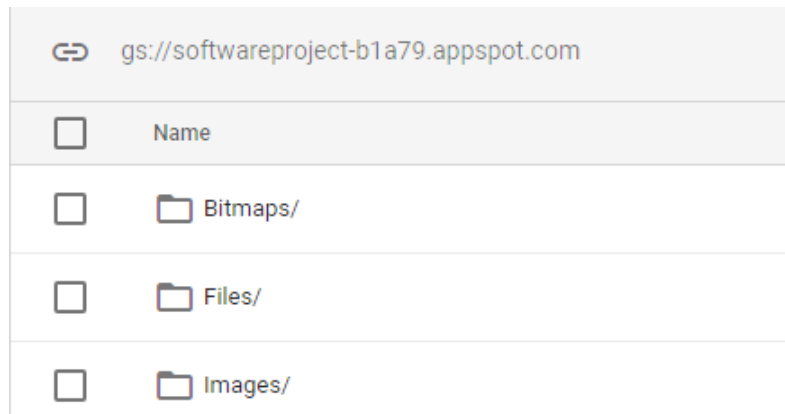


Figure 4 - Firestore Storage

2.7 Environmental Requirements

In this section I will explain the environmental requirements that are essential when developing the application.

- **Laptop/PC**

A laptop and/or PC with an internet browser, like Chrome or Firefox.

- **Android Mobile Device**

An android device is needed to run the android application and perform testing.

- **Android Studio**

Used to develop the android application.

- **Visual Studio**

Used to develop the web application.

- **Internet Access:**

Needed to use the android and web application as the Firestore Database, Storage and Authentication can only be accessed through an internet connection.

2.8 User Requirements

In this section I will explain the user requirements. If a user wants to use this application, he must meet the following requirements:

- **Android Device**

The android application will only run on devices running on Android OS.

- **Android 8.0 - Oreo**

The user should preferably have Android 8.0 installed, but the application is not limited to devices with Android 8.0. The application will also run on Android 7.0 Nougat and Android 6.0 Marshmallow.

- **Laptop/PC**

Needed to access the web application, the laptop or PC must have a browser which allows them to search for the web application as well as use it.

- **Internet Access:**

Needed to use the android and web application as the Firebase Database, Storage and Authentication can only be accessed through an internet connection.

2.9 Usability Requirements

To use the android and/or web application the user must have an account set up. The user must also be able to complete forms on either or both the web application and android application. The user's device must also be connected to the internet either by mobile data (3G, 4G, etc.) or WIFI.

2.10 Non-Functional Requirements

2.10.1 Performance/Response time requirement

Performance will be a major aspect in the overall application. The application is connected to a Firebase Database which requires an internet connection, so all features in the application will require the device be connected to the internet. The speed of the application uploading and downloading the user's data will be quick in some parts of the application, such as password and text blocks, as the size of the data will be small. The image and file sections will be slower but only marginally, depending on the size of the image or file. The speed of the internet the device is connected will also affect performance with slower connections making the features of the application function slower, however the users will be shown an upload or download progress bar. Unhandled network errors could result in uploads being stopped. Downloads however will continue when the device reconnects to an internet source.

Overall, the features of the application will function normally with only slow internet connections affecting the speed of uploads or downloads.

2.10.2 Availability requirement

The application must be available constantly as the user will use the application to store private or personal information and may need it at any moment. Once a user has installed the application on their android device or used the web application the application should be fully functional and accessible to the user. The application should have as little downtime as possible, if at all. Any bugs or errors will be recorded in Firebase Crashlytics and Analytics, so they can be repaired or removed.

2.10.3 Recover requirement

If a user forgets their password to their account the user should have an option to reset the password to a new one, this new password will allow them to login again.

2.10.4 Security requirement

The security of both applications will be to make sure that unwanted people will not be able to access the application without authentication and not be able to access any of the user's data.

The android and web application will follow best coding practices as well as follow the following security guidelines on how applications are attacked and exploited and trying to mitigate against them:

- Android Testing Cheat Sheet
 - https://www.owasp.org/index.php/Android_Testing_Cheat_Sheet
- Web Application Security Testing Cheat Sheet
 - https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet
- 12 Best Practices for user account, Auth and password management
 - <https://cloudplatform.googleblog.com/2018/01/12-best-practices-for-user-account.html>
- Firebase Realtime Database Rules
 - <https://firebase.google.com/docs/database/security/>
- Firebase – Secure Your Data
 - <https://firebase.google.com/docs/database/security/>

When creating an account, the password section will only accept a strong password (At least 8 characters, contain a number and a special character). This password is then encrypted using salted and password hashing, which is provided by Firebase Authentication. Only authenticated accounts can log into the application. The user's data that has been uploaded to the database is also encrypted but can only be decrypted using a pin code or by the user's fingerprint. The application will also not allow the "remember me" functionality so the application will not be constantly logged in to. The application will also be security tested to discover any threats to both the android and web application. The user's personal information which is required when creating an account is encrypted

using AES 256. Only secure encryption algorithms will be used for encrypting the user's passwords, images or files.

2.10.5 Reliability requirement

The reliability of the application has been taken seriously during its development as if the system keeps crashing or the code contains numerous bugs then these will affect the overall usability of the application. Using Firebase Crashlytics and Analytics I can see what are causing crashes and begin to fix or remove the cause of the crashes. The application should be able to handle smaller minor errors and function normally still. Testing will be carried out to find and root any problems that may lead to serious issues or inevitably lead to crashes.

2.10.6 Maintainability requirement

I plan on writing code that is easy to read, as well as commenting all my code, but also code that is easily extendable so that any changes big or small are easy to implement. Each section of the application will be separate from the others allowing me to maintain or change code without affecting other classes or features in the application. Also, the Firebase Database is a NoSQL database which will allow me to make changes to the databases schema easily without affecting data that is already there, this also allows me to change my code used to upload or download data with ease.

2.10.7 Reusability requirement

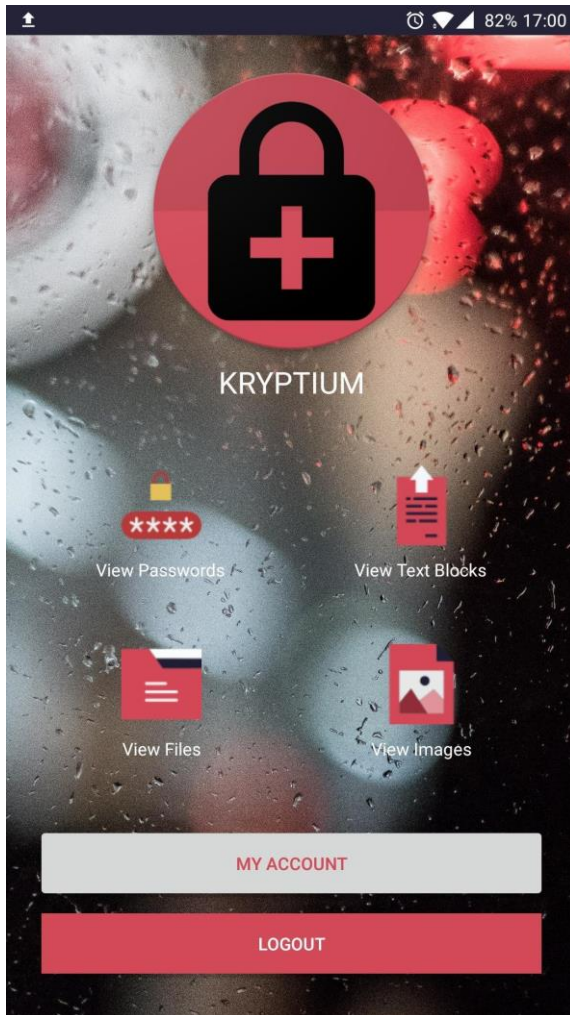
A large portion of the applications code will be reused across multiple sections of the application, this is to keep everything functioning and looking similar. It also dramatically reduces the development time as I will be using code I know is secure and works as intended.

2.11 Interface Requirements

2.11.1 GUI

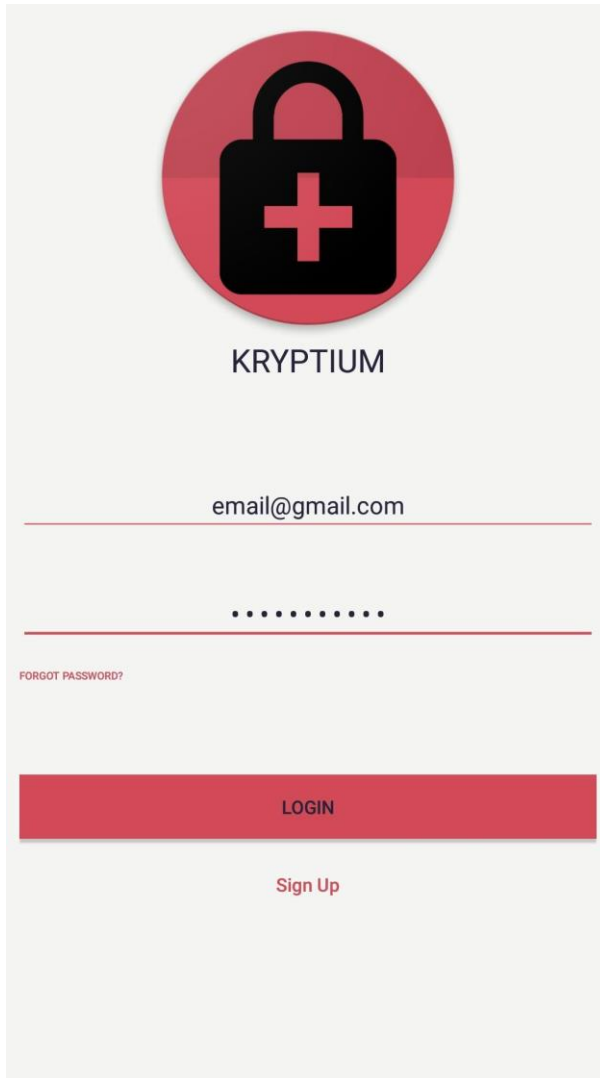
The android and web application GUI screenshots below show only some of the application as there are many different screens but they all follow the same template and all function the same way with respect to how the data is shown in a list and how the user would encrypt, decrypt or delete their data.

User Home Page GUI - Android



This screen is the android home screen, the UI the user sees after logging in. This screen allows navigation to the various sections of the android application

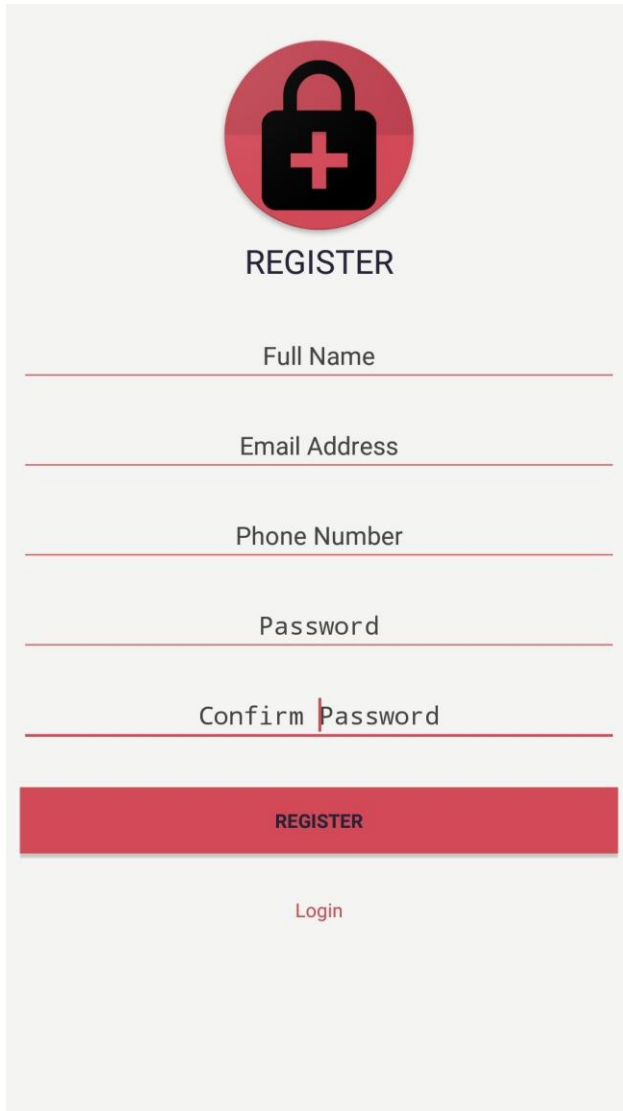
Login GUI - Android



The image shows a mobile login screen for an application named 'KRYPTIUM'. At the top center is a circular logo with a red background and a black padlock icon containing a red plus sign. Below the logo, the word 'KRYPTIUM' is written in a bold, black, sans-serif font. Underneath the name is a text input field containing the placeholder text 'email@gmail.com'. Below this is another text input field for a password, represented by a series of ten dots. To the left of the password field, the text 'FORGOT PASSWORD?' is displayed in a small, red, sans-serif font. At the bottom of the screen, there is a prominent red rectangular button with the word 'LOGIN' in white, bold, sans-serif text. Below the button, the text 'Sign Up' is displayed in a red, sans-serif font.

The screen is the login view where the user can enter their login credentials and log in. The view also contains buttons which allow the user to create an account or recover their account if they have forgot their password.

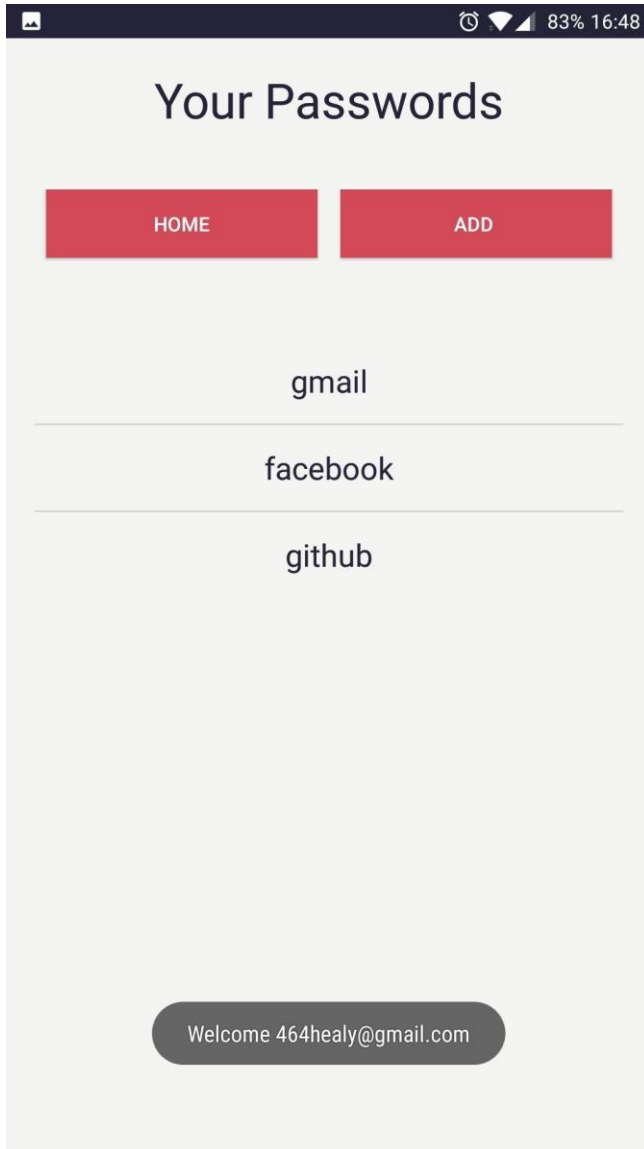
Register GUI - Android



The image shows a mobile application registration screen. At the top, there is a red circular icon containing a black padlock with a red cross. Below the icon, the word "REGISTER" is displayed in a bold, black, sans-serif font. The screen features five input fields, each with a red horizontal line below the text: "Full Name", "Email Address", "Phone Number", "Password", and "Confirm Password". Below these fields is a prominent red rectangular button with the word "REGISTER" in white, bold, uppercase letters. At the bottom of the screen, the word "Login" is written in a smaller, red, sans-serif font.

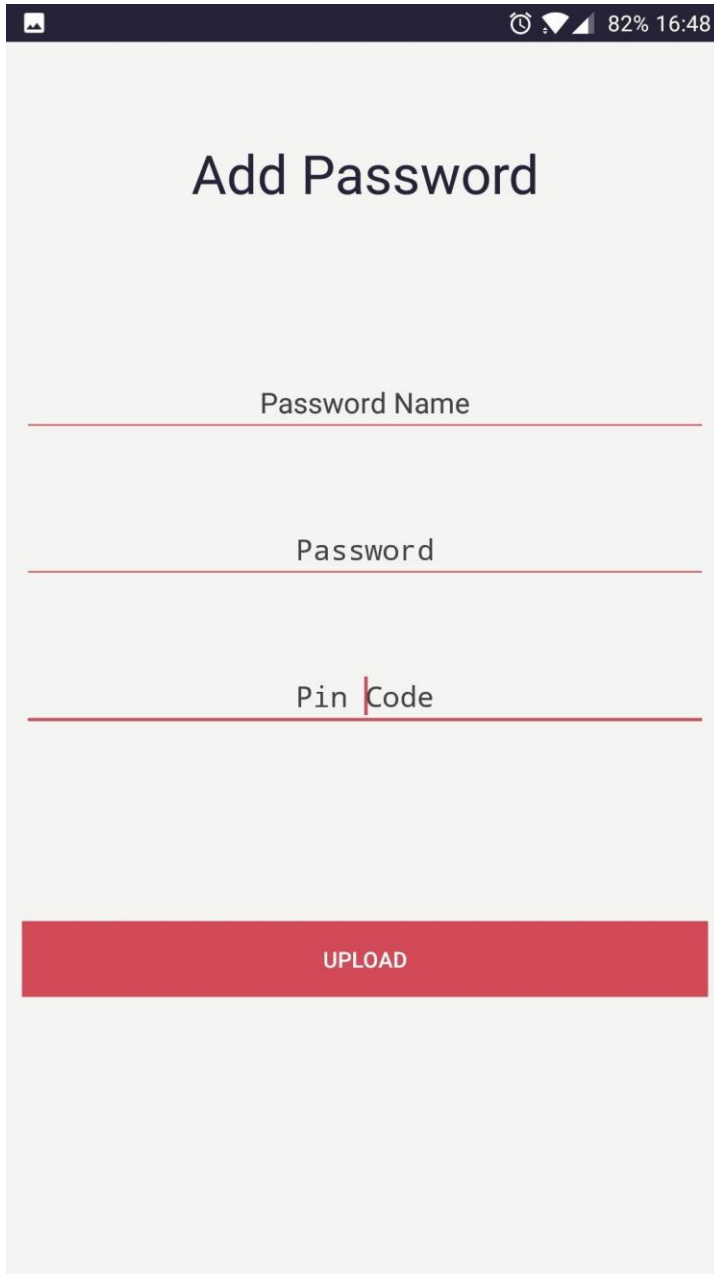
This is the screen where the user will create a new account. All inputs must be filled, and the email and password inputs are validated before the user can register.

View Data GUI - Android



When the user clicks the view passwords button on the home screen they are brought here where they can view a list of already encrypted passwords. They can, from this screen, decrypt a password or encrypt a new one.

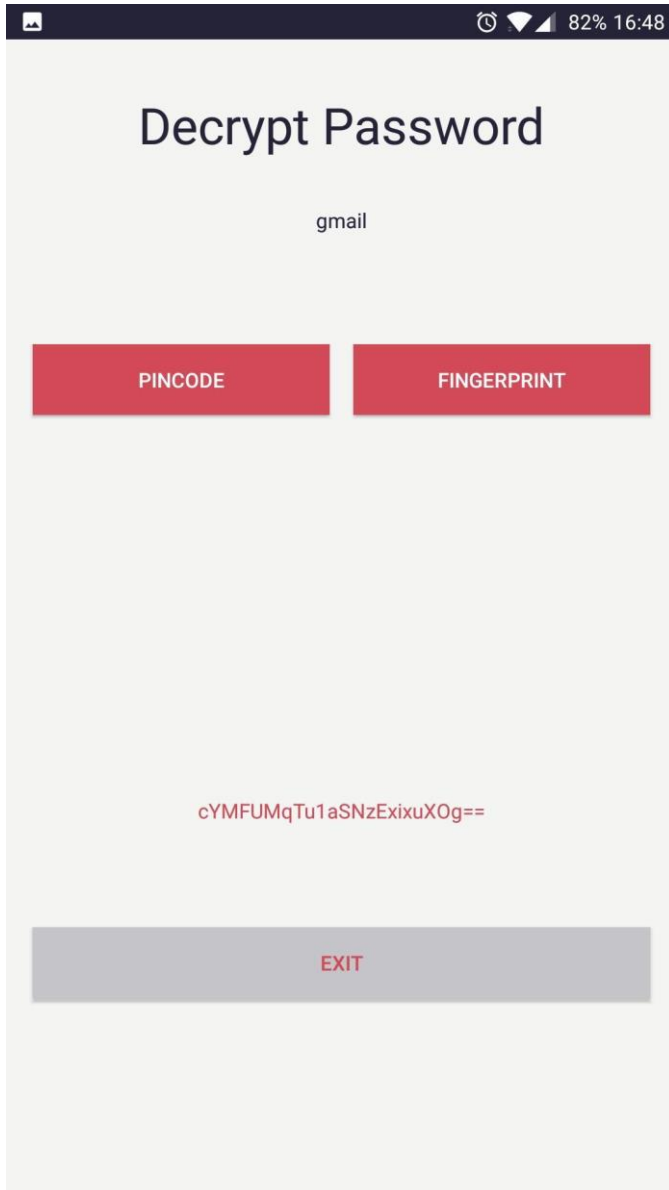
Add New Data GUI - Android



The screenshot shows an Android application interface with a dark status bar at the top displaying a clock icon, signal strength, Wi-Fi, and 82% battery at 16:48. The main content area has a light gray background and is titled "Add Password" in a large, dark font. Below the title are three input fields, each with a red underline: "Password Name", "Password", and "Pin Code". The "Pin Code" field has a vertical red line indicating the cursor position. At the bottom of the form is a prominent red button with the text "UPLOAD" in white capital letters.

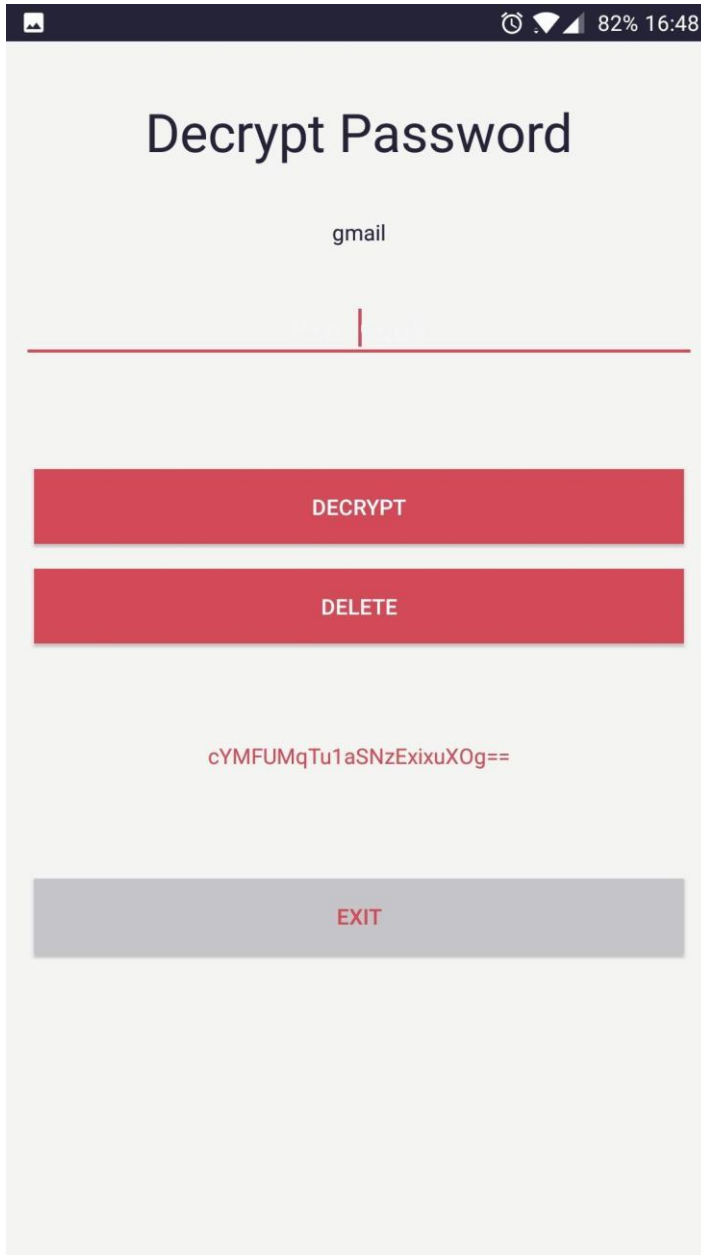
This view is where the user will encrypt a new password, the password input is the field that gets encrypted. The password name is what you see in the GUI screenshot above this one.

Decrypt Data GUI - Android



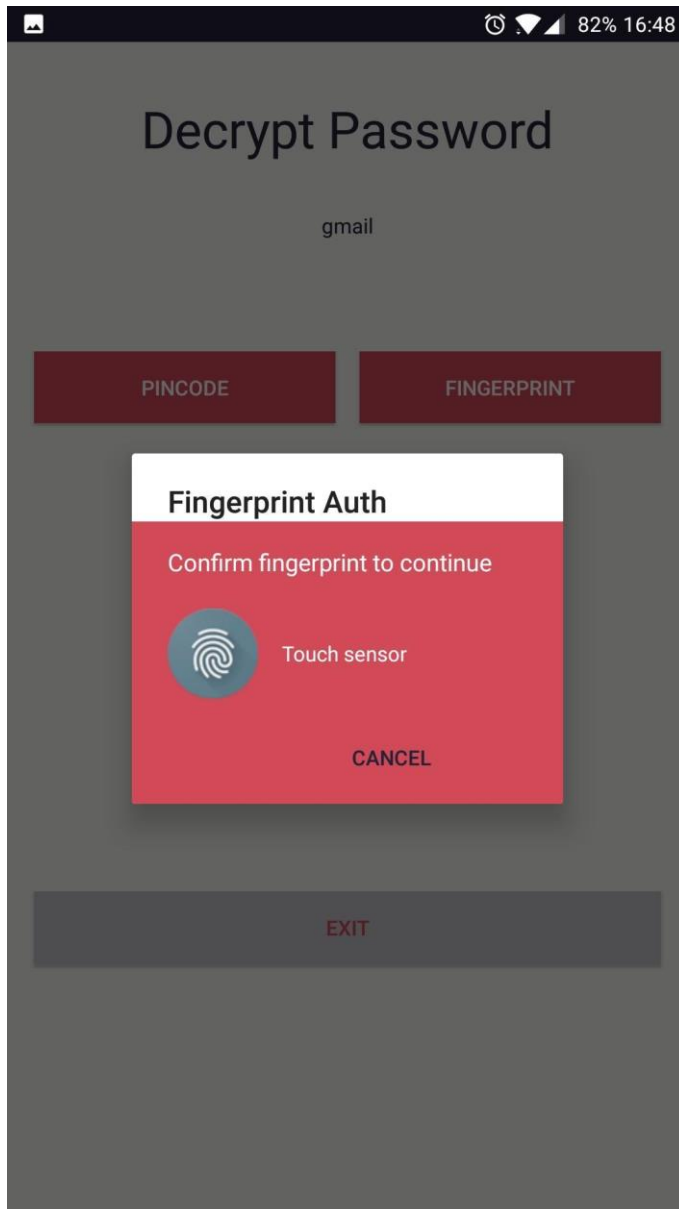
This is the screen the user is brought to after clicking on a password in the list. Here they can choose to decrypt a password using the set pin code or their fingerprint.

Decrypt with pin code GUI - Android



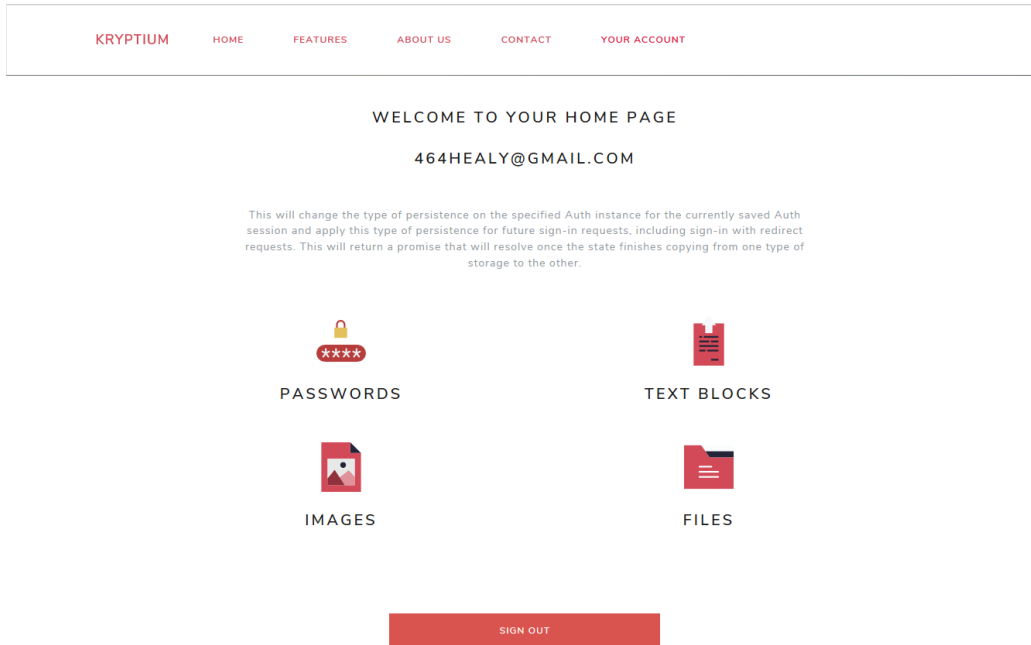
If the user selected decrypt with pin code the UI changes to show the decrypt and delete buttons as well as the pin code input.

Decrypt with Fingerprint GUI - Android



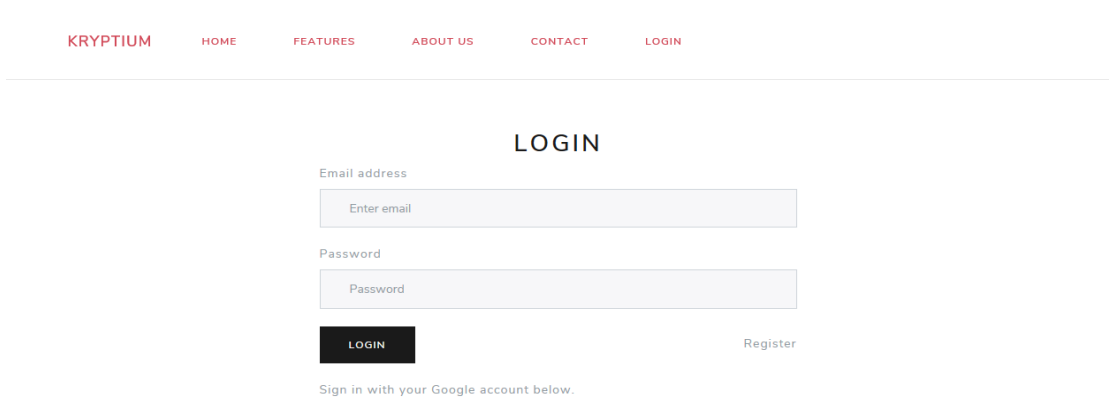
If the user selects to proceed with fingerprint authentication they see this popup. If the fingerprint matches the one stored by the android fingerprint manager, then they can decrypt or delete a password.

User Home Page GUI - Web



This screenshot shows the layout of the user's home page on the web application from which they can navigate through the different areas of the website. It is similar to the android application.

Login GUI - Web



The screen above shows the login UI of the web application

Register GUI - Web

REGISTER

Email address

We'll never share your email with anyone else.

Password

REGISTER

View Data GUI - Web

YOUR PASSWORDS

PASSWORD NAME	PASSWORD
gmail	cYMFUMqTu1aSNzExixuXOg==
facebook	NZQ5I3kiaBKqdIS20WRTf9eMbK8vc/jogdqjCZtUU4=
github	qYgt9HiyzFSzbr3+TxixsveEo8CP6QQk5yvAPNZ3BXU=

This screenshot shows the UI for displaying the user data in a table. In the screen the users encrypted passwords are in the table.

Add New Data GUI - Web

ADD A NEW PASSWORD

ENCRYPT UPLOAD

To encrypt a new password, or any data, the user must correctly fill each input. First the inputs are encrypted then uploaded to the database.

Decrypt Data GUI - Web

DECRYPT A PASSWORD

github

qYgt9HiyzFSzbR3+TxixsveEo8CP6QQk5yvAPNZ3BXU=

DECRYPT

This screenshot shows the UI of the web application where users can decrypt their data. They must provide the correct pin code, stored in the database, to decrypt the data.

2.11.2 Application Programming Interfaces (API)

My application will be using several API's with most of the them being Google API's.

Google Firebase Authentication is used to authenticate users who are attempting to login to the application. When a user creates an account the user's data is stored in Google Firebase Authentication where the user's password is encrypted, and the account is assigned a user ID, W6QDouOYFgNTmlq UgBPYR7n3Mvk1 is an example of a user ID. When a user tries to login the credentials the user entered is compared in Firebase Authentication, if they match then user can login.

The user will also have other methods of creating an account and logging in. Users can choose to use an email and password to login, mentioned above, or they can use the Google Sign-In API which uses their Gmail account to create an account and login. It is much faster as once you are authenticated you can log in with a single button click.

Google Firebase Realtime Database will also be used to store the user's information, passwords, text blocks as well as the images and files metadata. The Firebase Database is a NoSQL database which allows me to change the schema and structure of the data if I need to simply by changing a line of code. Also, because it is a real-time database the data which is displayed in the application updates almost instantly and is constantly synced to show the user the data in real time.

Google Firebase Storage is used to store the actual image and file. In the Database where the image or file metadata is stored, it also contains a download URL, a link to the image in Storage which allows me to call the data stored in the database as well as the actual image or file.

To add Firebase in your application includes creating a class which allows Firebase to be in a persistent state and to check it is functioning properly as well as including Firebase maven artifacts in the gradle file, these maven files can be seen in figure 5.

```
compile 'com.google.firebase:firebase-auth:11.8.0'  
compile 'com.google.android.gms:play-services-auth:11.8.0'  
compile 'com.google.firebase:firebase-database:11.8.0'  
compile 'com.google.firebase:firebase-storage:11.8.0'  
compile 'com.google.firebase:firebase-crash:11.8.0'  
compile 'com.google.firebase:firebase-core:11.8.0'  
compile 'com.firebaseui:firebase-ui:0.4.3'
```

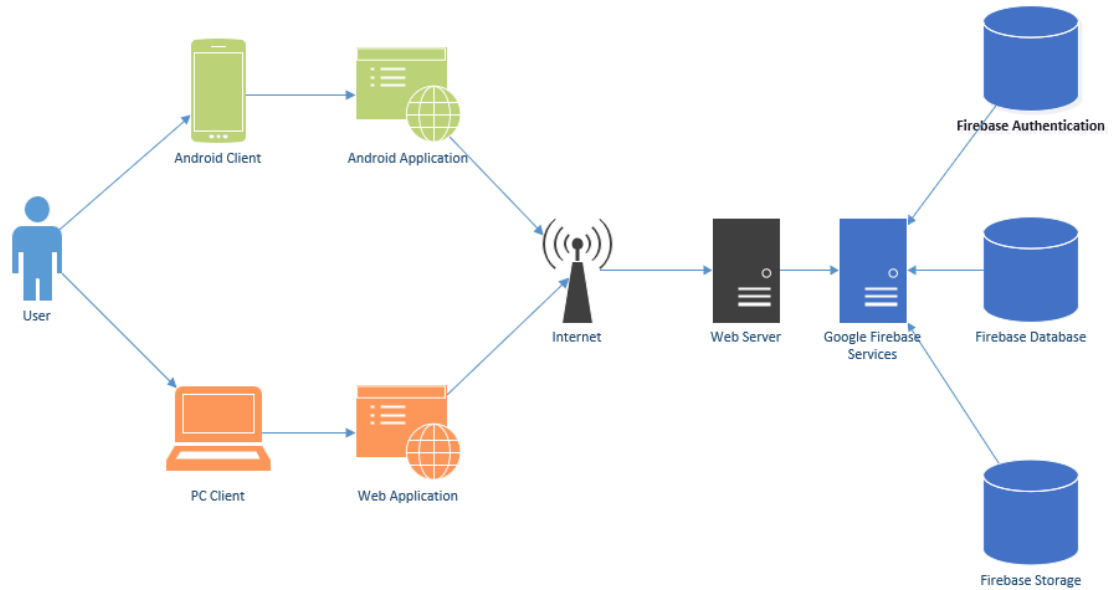
Figure 5 - Firebase Gradle

SpongyCastle and BouncyCastle APIs which provide cryptography support in android applications as well as C# applications. The Legion of the Bouncy Castle created and maintain these APIs as well as provide documentation on what encryption, hashing and ciphers are supported by it. Using these APIs involved adding the maven artifacts, seen in figure 6, in the gradle file of the android application and also creating a new security provider on the classes where these APIs will be used.

```
compile 'com.madgag.spongycastle:core:1.58.0.0'  
compile 'com.madgag.spongycastle:prov:1.58.0.0'  
compile 'com.madgag.spongycastle:pkix:1.54.0.0'  
compile 'com.madgag.spongycastle:pg:1.54.0.0'  
  
compile 'org.bouncycastle:bcpkix-jdk15on:1.56'
```

Figure 6 - SpongyCastle Gradle

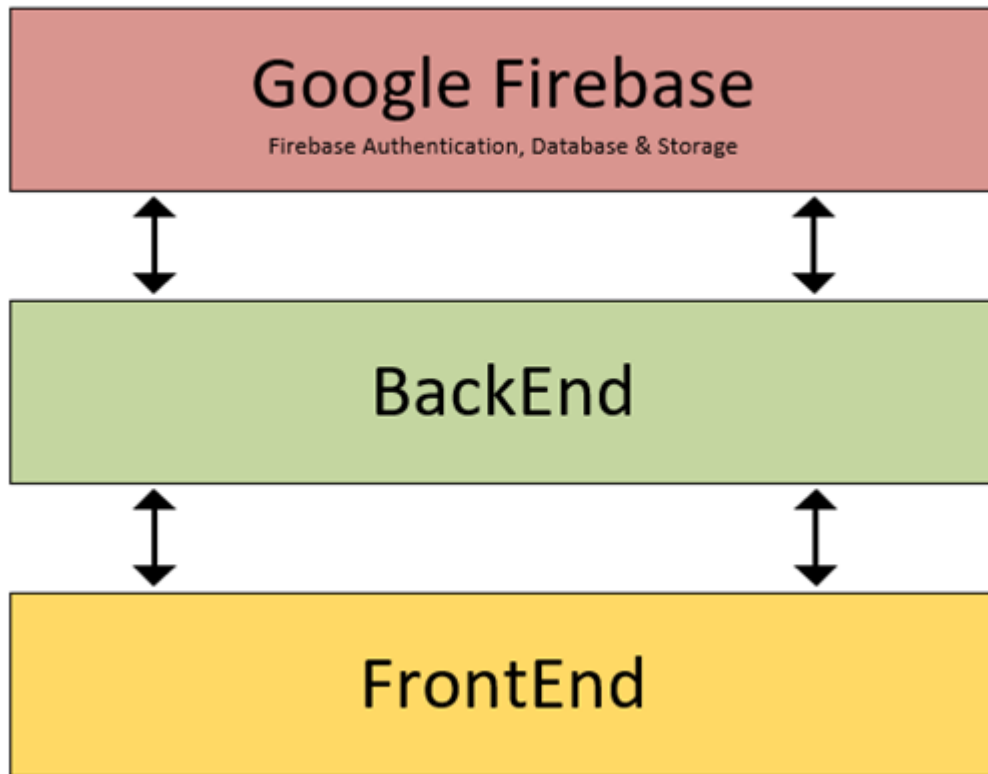
2.12 System Architecture



The system architecture diagram above shows how a user can use either an android mobile device or a PC client to gain access to the android or web application. The android and web application are only accessible from the internet as the data is stored in a Google Firebase Database, Storage and Authentication. The android or PC clients can now access the server which the Database, Storage and Authentication are hosted on.

Below is a more detailed view of the software design of the application which is presented by a class diagram.

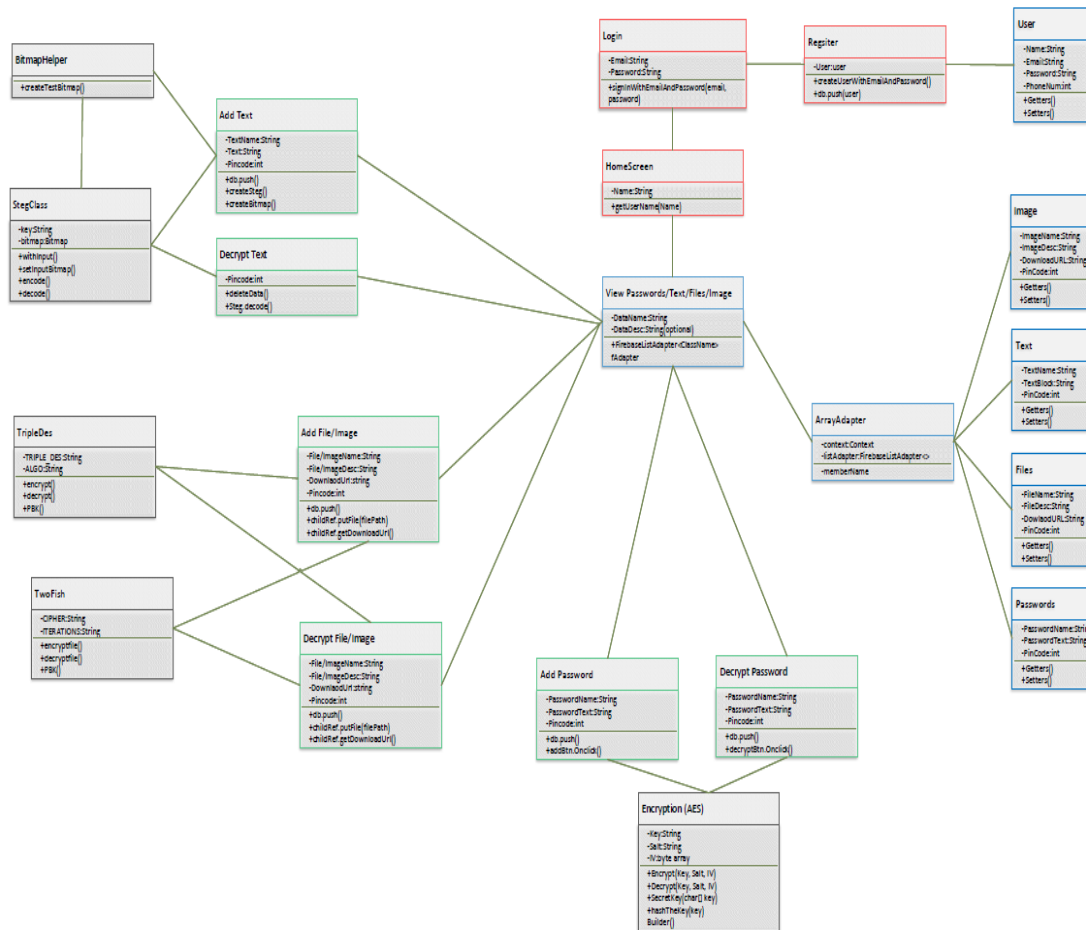
2.13 Logical View Architecture



The logical view of both applications can be split into three parts.

The frontend which creates the user interface of both applications, the backend which implements the functionality of the applications and the Google Firebase which is a cloud service. The backend connects to Google Firebase which provides authentication as well as data storage.

2.14 Design and Architecture



The Android application is built using Java and XML, the application consists of multiple activities which contain the UI and functionality of the application. The web application will be created using HTML5, Bootstrap, CSS, JavaScript and C#. Both the android and web application will essentially be carbon copies of each other just developed for different platforms and using different programming languages, but the functionality and features will be the same nonetheless.

I will be creating the UI of the android application myself but will be using Bootstrap in the web application to help me create the UI, I will also be using a theme from Bootswatch called LUX to help me create a web UI that is both easy to navigate but also appealing to look at.

A Firebase Database and Storage is used to store the user's information and encrypted data. Firebase Authentication is used to create the user accounts but also provide the login functionality by accessing the Auth API. A user session will also be created when the user logs in successfully and is stored until the user logs out or the application is closed.

Register Class

Before a user can access the application they first must create an account, the register class handles the creation of a new account as well as adding the user details to the database.

User Class

The user class is used when the user is creating an account but also used by the array adapter class when the user is viewing their account information in the application.

Login Class

The login class allows a user to login using the email and password they set in the register page. The class accesses the Firebase Auth API which handles comparing the user's inputs to what is stored in the Firebase Auth Database.

Password Class

The password class is used when the user is creating a new password but also used by the array adapter class when the user is viewing their passwords already in the database in the application.

Text Class

The text class is used when the user is creating a new text block but also used by the array adapter class when the user is viewing their text blocks already in the database in the application.

Image Class

The image class is used when the user is creating a new image but also used by the array adapter class when the user is viewing their images already in the database in the application.

File Class

The file class is used when the user is creating a new file but also used by the array adapter class when the user is viewing their files already in the database in the application.

View Passwords/Text/Files/Images Class

These classes are essentially all the same, they use an Array Adapter to get the data from the database and show the data in a list view, the list view only shows the title of each row in the database, such as password name, text name, file name and image name. The items in the list view are clickable, when the user clicks on an item they are brought to the decrypt and delete activity where the data from the list view is passed to this activity.

Delete/Decrypt Data Class

The delete/decrypt class handles the decryption or/and the deletion of the user's data. The user can choose between using a pin code or their fingerprint. The fingerprint button allows the user to use the devices fingerprint scanner while the pin code button allows a user to enter a pin code to decrypt or delete the data.

For the password and text block decryption the decrypted text is outputted on the activity screen while the image and file decryption the file is decrypted and downloaded to the user's device.

Add Password Class

The add password class handles the creation of a new password which is encrypted using AES 256 and is uploaded to the Firebase Database.

Add Text Class

The add text class handles the creation of a new text block which is hidden inside an image file using steganography techniques. The text block info is uploaded to the Firebase Database along with a link to the image which is stored in Firebase Storage.

Add File/Image Class

The add File class handles the creation of a new file which is encrypted using TwoFish, the file information is uploaded to the Firebase Database along with a link to the actual file which is stored in Firebase Storage. This class also allows the user to open the devices local memory and allow them to choose a file to upload.

The add Image class handles the creation of a new image which is encrypted using TripleDES, the file information is uploaded to the Firebase Database along with a link to the actual file which is stored in Firebase Storage. This class also allows the user to open the devices local memory and allow them to choose a file to upload.

AES/Twofish/TripleDES Class

These classes contain the code needed to allow the users data to be encrypted and decrypted using the methods and algorithms in these classes.

2.15 Implementation

The purpose of this section is to describe the technologies used in the implementation of the Android and Web application.

2.15.1 Technology Overview

The android application was created using Java and XML and developed on Android Studio, the web application was created using HTML, CSS, JavaScript and C# and developed on Visual Studio. Google Firebase because of the number of features it had such as authentication, database and storage as well as analytics and Crashlytics. Google Firebase is a cloud service which meant the data stored there could be access on both platforms in real time.

2.15.2 Security

2.15.2.1 Google Firebase - SSL

Firebase supports SSL¹ (Secure Socket Layer), this means that the Firebase JavaScript include and the packets that are sent between the Firebase server and the users browser will be encrypted. To include this in the application it required adding one line of JavaScript to the Firebase JavaScript File in the web application this can be seen in figure 7 below.

```
<script src="https://cdn.firebase.com/js/client/2.4.2/firebase.js"></script>
```

Figure 7 - JavaScript Firebase SSL

¹ <https://firebase.googleblog.com/2012/07/firebase-databases-now-support-ssl.html>

2.15.2.2 Firebase Authentication

Firebase authentication² is a backend service which is implemented in the application to provide login and register functionality as well as create the user's session in the applications. When a user logs in to the application that user becomes the current user of the Auth instance, the Auth instance functions similar to a session. The Firebase Auth instance persists the users state, logged in, even when refreshing the page or closing the application.

When a user logs out, the Auth instance stops and there is no current user, so the application closes or redirects the user.

The credentials that user will need to have when creating an account or logging in are an email address and a strong password. These are used to create the account in Firebase Authentication and create the User UID. The password is hashed and salted using scrypt, a strong and secure hashing algorithm, which can be seen in figure 8 below.

```
hash_config {
  algorithm: SCRYPT,
  base64_signer_key: os3YnH3IeIaIJu8hkBuyX/81NRn1xNauFk1+TWNbk7aUoVZH95tS/BdUz60GEYcnftA1krU21ER3NUVL0KYtG==
  base64_salt_separator: Bw==,
  rounds: 8,
  mem_cost: 14,
}
```

Figure 8 - Firebase Authentication SCRYPT Hashing

Firebase Authentication also allowed me to create rules such as only one email address can be used per account, this can be seen in figure 9 below, which means users cannot create multiple accounts with the same email address.


One account per email address
Preventing users from creating multiple accounts using the same email address with different authentication providers.
[Learn more](#) 

Figure 9 - Authentication Rules

²https://firebase.google.com/docs/auth/?gclid=CjwKCAjw_tTXBRBsEiwArqXyMv3j1mVd3mNqyWiDLpDtWkRWRHS_N9boaBQtsMsH__Hic_07j2bmPRoCUYUQAvD_BwE

2.15.2.3 Firebase Database Rules

Firebase Database allows the creation of rules for the database, that means data being read or written must abide by these rules or no data is transferred. Below in figure 10 are the basic rules I used in the database which requires the user be logged in and authenticated before any data is read from or wrote to the database. Rules can also be used to validate or transform data, a lot of this is done in the backend of the applications.

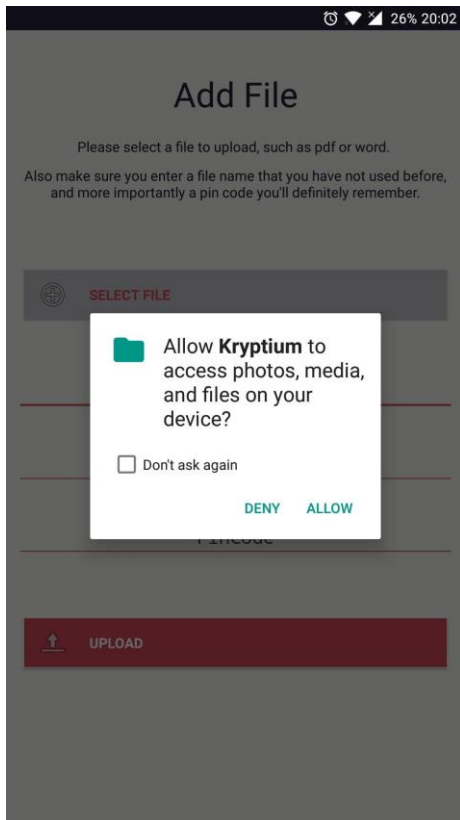


```
1 {
2   "rules": {
3     ".read": "auth != null",
4     ".write": "auth != null"
5   }
6 }
```

Figure 10 - Firebase Database Rules

2.15.2.4 Android Permission Checks

The purpose of the permissions is to protect the privacy of Android users. Android applications above SDK version 5.0 must request permission from the user to access potentially private or sensitive information such as contacts, SMS, and in the case of Kryptium the user's local storage. To use the image and file encryption features in the Android application, the application must first have permission to access the user's storage. When a user first tries to encrypt a new image or file, the dialog as seen in the screenshot below pops up asking for the correct permissions. If the user gives the permissions then they can encrypt and decrypt images and files, but if they deny giving the permissions they are brought back to the home screen. Every time the user tried to encrypt an image or file the dialog pops up again.



This method is in each class that need permissions to function properly. If the permission is not granted it displays the popup, but if the permission is given then the feature is usable. The method used can be seen in figure 11.

```
public void onRequestPermissionsResult(int requestCode, @NonNull String permissions[], @NonNull int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_STORAGE: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Snackbar.make(linearLayout, text: "Permission Granted", Snackbar.LENGTH_SHORT).show();
            } else {
                Intent intent = new Intent( packageContext: DecryptFiles.this, ViewFiles.class);
                Snackbar.make(linearLayout, text: "Please Enable Application Permissions", Snackbar.LENGTH_SHORT).show();
                startActivity(intent);
            }
        }
    }
}
```

Figure 11 - Android Permission Request

When a user first logs in they are also prompted with a permission check but this time for the permission to use the fingerprint scanner on their phone. Below is a screenshot of the phone asking for permission as well as the code used.

2.15.2.5 Fingerprint Authentication

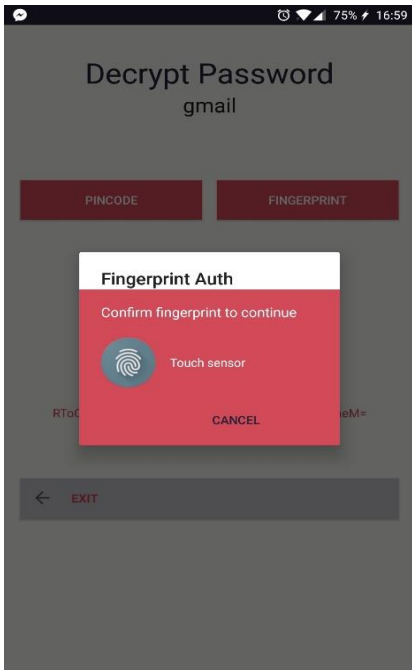
Most modern android mobile phones have a fingerprint scanner on them which allow users to unlock their phone with their fingerprint. Android applications can use the fingerprint scanner as a way of authenticating users for in-app purchases, login or in this case the decryption of private information. Kryptium allows users to choose between decrypting their data using a pin code or their fingerprint. To use the mobile devices fingerprint scanner requires adding permissions to the manifest and creating two classes which access the devices fingerprint manager and allow the fingerprint scanner to be used by the application. When the user logs in for the first time they are asked to give the application the permissions needed to use this feature, if the device doesn't have a fingerprint scanner or if they don't want to use it they can deny the request and still be able to use the pin code functionality.

The code below is used to check if the application has the permission needed to use the fingerprint scanner.

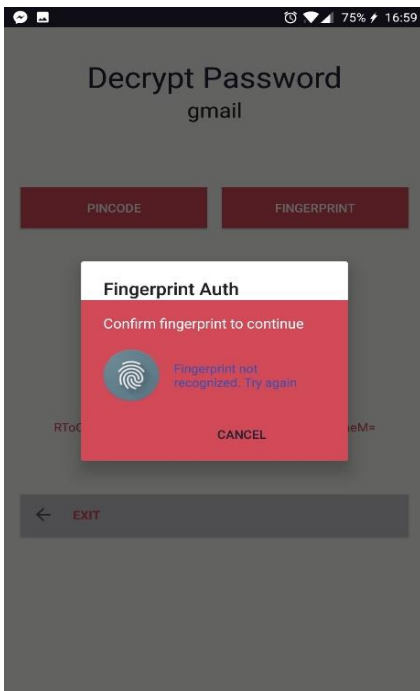
```
ActivityCompat.requestPermissions(activity: HomeScreen.this, new String[]{
    Manifest.permission.USE_FINGERPRINT}, MY_PERMISSIONS_REQUEST_READ_STORAGE
);

public void onRequestPermissionsResult(int requestCode, @NonNull String permissions[], @NonNull int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_STORAGE: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Snackbar.make(linearLayout, text: "Permission Granted", Snackbar.LENGTH_SHORT).show();
            } else {
                Intent intent = new Intent(packageContext: HomeScreen.this, ViewImage.class);
                Snackbar.make(linearLayout, text: "Please Enable Application Permissions", Snackbar.LENGTH_SHORT).show();
                startActivity(intent);
            }
        }
    }
}
```

The screenshot shows the dialog which is shown when the user selects to decrypt using the fingerprint scanner.



If the fingerprint does not match the one stored in the android fingerprint manager the user cannot decrypt the data, after three failed attempts the user cannot use the fingerprint again for a certain amount of time.



2.15.2.6 Strong Passwords

The method below is used to make sure that the user always enters a strong password when creating an account on the application (NIST, SP 800-63)³.

The password must:

- be at least 12 characters long
- Password must match confirm password input
- Contain an uppercase letter
- Contain a number
- Contain a special character (@, #, £, \$)

On the web application this is created using regular expression validators, seen in the figure 12 below, which validate an input, in this case the password input, to make sure it contains a special character, a number and be at least 12 characters long.

```
ControlToValidate="newPassword" ErrorMessage="Password must contain one special character." ValidationExpression=".*[!@#$%^&*~/.]" /><br />
ControlToValidate="newPassword" ErrorMessage="Password must be 12 characters." ValidationExpression="[^\s]{12,}" />
ControlToValidate="newPassword" ErrorMessage="Password must contain one number." ValidationExpression="[0-9]" />
```

Figure 12 - Regular Expression Validators

JavaScript is used to compare the password and the confirm password inputs as well as makes sure the password is at least 12 characters. This happens before any account is registered.

```
var password = document.getElementById('<%=newPassword.ClientID%>').value;
var passwordconfirm = document.getElementById('<%=newPasswordConfirm.ClientID%>').value;

if (email.length = 0) {
    return;
}
if (password.length < 12) {
    return;
}
if (password != passwordconfirm) {
    return;
}
```

³ <https://www.nist.gov/itl/tig/projects/special-publication-800-63>

On android the method in the screenshot below is used to enforce the strong password requirements.

```
public boolean isValid(String password, String confirm, List<String> errorList) {

    Pattern specialCharPatten = Pattern.compile( regex: "[^a-z0-9 ]", Pattern.CASE_INSENSITIVE);
    Pattern UpperCasePatten = Pattern.compile("[A-Z ]");
    Pattern lowerCasePatten = Pattern.compile("[a-z ]");
    Pattern digitCasePatten = Pattern.compile("[0-9 ]");
    errorList.clear();

    boolean flag=true;

    if (TextUtils.isEmpty(password)) {
        Toast.makeText( context: Register.this, text: "Enter password!", Toast.LENGTH_SHORT).show();
        flag=false;
    }

    if (!password.equals(confirm)) {
        errorList.add("Password and confirm password does not match");
        flag=false;
    }

    if (password.length() < 12) {
        errorList.add("Password length must have at least 8 character !!");
        flag=false;
    }

    if (!specialCharPatten.matcher(password).find()) {
        errorList.add("Password must have at least one special character !!");
        flag=false;
    }

    if (!UpperCasePatten.matcher(password).find()) {
        errorList.add("Password must have at least one uppercase character !!");
        flag=false;
    }

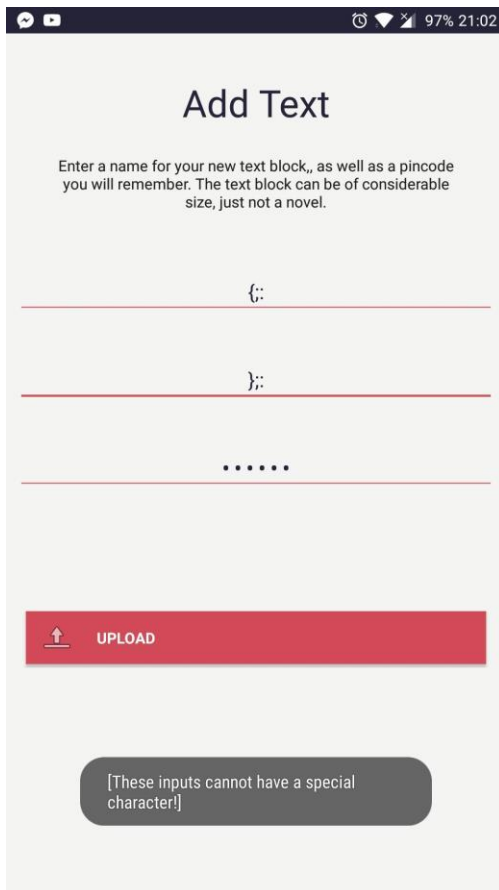
    if (!lowerCasePatten.matcher(password).find()) {
        errorList.add("Password must have at least one lowercase character !!");
        flag=false;
    }

    if (!digitCasePatten.matcher(password).find()) {
        errorList.add("Password must have at least one digit character !!");
        flag=false;
    }

    return flag;
}
```

2.15.2.7 Input Validation - NoSQL Injections

Firestore Database is a NoSQL database, so no SQL is used when reading and writing to the database, but injections are still possible. NoSQL database calls are written in the application's programming language, in this case Java and JavaScript. Before any user submitted inputs are written to the database, input validation is used to filter the user's inputs so that no `< > & ; / { } :` characters are allowed to be written to the database⁴. These special characters are needed to write a NoSQL injection. The screenshot below shows the android application filtering for these characters.



⁴ https://www.owasp.org/index.php/Testing_for_NoSQL_injection

The method below is used to filter and validate all users input which do not allow any special characters.

```
public boolean NoSpecialChars(String text, List<String> errorList) {  
  
    Pattern specialCharPatten = Pattern.compile( regex: "[^a-z0-9 ]", Pattern.CASE_INSENSITIVE);  
    errorList.clear();  
  
    boolean flag=false;  
  
    if (specialCharPatten.matcher(text).find()) {  
        errorList.add("These inputs cannot have a special character!");  
        flag=true;  
    }  
  
    return flag;  
}
```

This method is called on each class that handle any of the user's data being written to the database. The method, as well as other validation, checks to see if special characters are present before the upload method is reached, if one is found the error message shows and no upload or encryption occurs.

```
InputValidation validation = new InputValidation();  
List<String> errorList = new ArrayList<>();  
boolean isValid = validation.NoSpecialChars(passName, errorList);  
if (isValid) {  
    Toast.makeText(getApplicationContext(), "text: "+errorList, Toast.LENGTH_SHORT).show();  
    pd.dismiss();  
    return;  
}
```

The Web Browser XSS Protection was enabled, this turns on the XSS filter provided by web browsers, if any scripts from user inputs are detected the browser will sanitize the page as well as block the render of the webpage. This is done to prevent any scripts being submitted or rendered on the web application. Figure 13 below shows the http protocol which is added to the Web.config file.

```
<httpProtocol>  
  <customHeaders>  
    <add name="X-Xss-Protection" value="1; mode=block" />  
  </customHeaders>  
</httpProtocol>
```

Figure 13 - XSS Protection

2.15.2.8 User ID Checks

On both the web and android application a Firebase method is used, figure 14 and 15 below show the methods used on each platform, which checks to see if there is currently a user logged in if there is an Auth instance. If there is a user then the data on the page will be shown or the user can write to the database but if there is no Auth instance that the method can find then UI will change, no data from the database will be read and no data can be written to the database. On the android app the user is redirected to the login page, on the web application the UI changes to a message saying, "Please Login".

Android

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
if (user != null) {
    String email = user.getEmail();
    Toast.makeText( context: this, text: "Welcome " + email, Toast.LENGTH_SHORT).show();
} else {
    // No user is signed in
    Toast.makeText( context: this, text: "You shouldn't be here", Toast.LENGTH_SHORT).show();

    Intent intent = new Intent( packageContext: blogHome.this, MainActivity.class);
    startActivity(intent);
}
```

Figure 14 - Android User Checks

Web

```
<script>
function logout() {
    firebase.auth().signOut();
    alert("logged out");
}

firebase.auth().onAuthStateChanged(function (user) {
    if (user) {
        var email = user.email;

        document.getElementById('userName').textContent = email;
        document.getElementById('userBox').style.display = 'inline';
        document.getElementById('loggedoutBox').style.display = 'none';
    } else {
        document.getElementById('userBox').style.display = 'none';
        document.getElementById('loggedoutBox').style.display = 'inline';
    }
});
</script>
```

Figure 15 - Web User Checks

2.15.3 Encryption

2.15.3.1 AES

AES is a FIPS approved cryptographic symmetric block cipher that encrypts data into unreadable ciphertext, then can decrypt the data back to plaintext. The AES algorithm can use crypto key lengths of 128, 192 and 256 bits to encrypt and decrypt data in blocks of 128 bits⁵ (NIST FIPS 197).

The features of AES in Kryptium are:

- 256-bit cryptographic key
- 128-bit data blocks
- PKCS5Padding
- CBC Mode, Cipher Block Chaining
- PBKDF2 Hashing
- 128-bit random IV

CBC mode makes it more difficult to perform known-plaintext attacks on your ciphertext as well as leaks less information compared to ECB mode (Jeff Nelson, 2014). PKCS5Padding causes the cipher to pad the data, making it more difficult to attack the ciphertext directly, but also so the data being encrypted is in 8-byte block sizes. If the data is not in 8-byte blocks when trying to decrypt a bad padding exception will be thrown and the ciphertext will not be decrypted.

The method below, figure 16, is used to generate a new random IV.

```
private static byte[] generateRandomIV(int length)
{
    byte[] iv = new byte[length];
    SecureRandom random = new SecureRandom();
    random.nextBytes(iv);

    return iv;
}
```

Figure 16 - Random IV Method

⁵ <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>

Figure 17 shows the AES instance being used, the key size and the IV size.

```
AESCipher() throws NoSuchAlgorithmException, NoSuchPaddingException {
    _cx = Cipher.getInstance("AES/CBC/PKCS5Padding");
    _key = new byte[32]; //256 bit key space
    _iv = new byte[16]; //128 bit IV
}
```

Figure 17 – Android AES

The code in figure 18 below is part of the AES method used to combine the cryptographic key, the IV and the plaintext password together to create the ciphertext. The IV is combined to the ciphertext so it can be used again for the decryption.

```
byte[] iv = generateRandomIV(IV_LENGTH);
IvParameterSpec ivSpec = new IvParameterSpec(iv);

if (_mode.equals(EncryptMode.ENCRYPT)) {
    _cx.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
    byte[] results = _cx.doFinal(_inputText.getBytes( "UTF-8"));

    byte[] combined = new byte[IV_LENGTH + results.length];
    System.arraycopy(iv, 0, combined, 0, IV_LENGTH);
    System.arraycopy(results, 0, combined, IV_LENGTH, results.length);

    _out = Base64.encodeToString(combined, Base64.DEFAULT); // ciphertext
}

if (_mode.equals(EncryptMode.DECRYPT)) {
    System.arraycopy(_inputText.getBytes( "UTF-8"), 0, iv, 0, iv.length);
    IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);

    _cx.init(Cipher.DECRYPT_MODE, keySpec, ivParameterSpec);
    byte[] decodedValue = Base64.decode(_inputText.getBytes(), Base64.DEFAULT);
    byte[] decryptedVal = _cx.doFinal(decodedValue);

    System.arraycopy(decryptedVal, 0, decryptedVal, 0, decryptedVal.length);
    String decryptedPassword = new String(decryptedVal);

    String newText = decryptedPassword.substring(iv.length, decryptedPassword.length());

    _out = newText;
}
```

Figure 18 – Android AES

Below figure 19 and 20 shows the C# implementation of AES in the web application. Figure 13 shows the AES parameters being created which matches the Java AES parameters, this is so passwords encrypted on either platform can be decrypted on both.

```
public AESCipher()
{
    _enc = new UTF8Encoding();
    _rcipher = new RijndaelManaged();
    _rcipher.Mode = CipherMode.CBC;
    _rcipher.Padding = PaddingMode.PKCS7;
    _rcipher.KeySize = 256;
    _rcipher.BlockSize = 128;
    _key = new byte[32];
    _iv = new byte[_rcipher.BlockSize / 8]; //128 bit / 8 = 16 bytes
    _ivBytes = new byte[16];
}
```

Figure 19 - AES C#

The method of combining the key, plaintext and IV is very similar to the Java version with the IV being combined with the ciphertext so it can be decrypted later. This is seen in Figure 20.

```
if (_mode.Equals(EncryptMode.ENCRYPT))
{
    //encrypt
    byte[] plainText = _rcipher.CreateEncryptor().TransformFinalBlock(_enc.GetBytes(_inputText),
        0, _inputText.Length);
    byte[] combined = new byte[_iv.Length + plainText.Length];
    System.Buffer.BlockCopy(_iv, 0, combined, 0, _iv.Length);
    System.Buffer.BlockCopy(plainText, 0, combined, _iv.Length, plainText.Length);
    _out = Convert.ToBase64String(combined);
}
if (_mode.Equals(EncryptMode.DECRYPT))
{
    byte[] plainText = _rcipher.CreateDecryptor().TransformFinalBlock(Convert.FromBase64String(_inputText),
        0, Convert.FromBase64String(_inputText).Length);
    _out = _enc.GetString(plainText);
}
_rcipher.Dispose();
```

Figure 20 - AES C#

AES comprises of a series of linked operations and performs all its computations on bytes rather than bits, this means that 128 bits of plaintext is treated as 16 bytes. The key will be 256-bit length which uses 14 rounds. Each of these rounds uses a different 256-bit key which is calculated from the original AES key (Advanced Encryption Standard, Tutorialspoint.com).

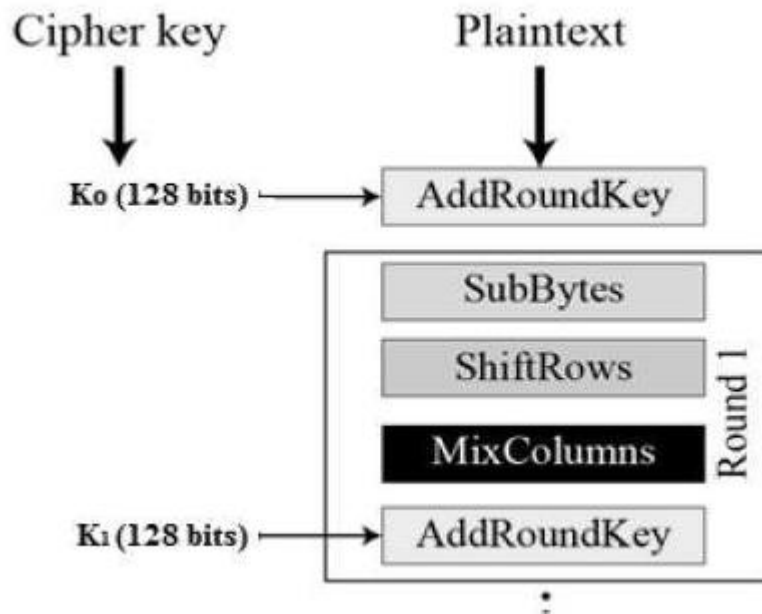


Figure 21 - AES Method Diagram <https://www.tutorialspoint.com/cryptography/advancedencryptionstandard.htm>

The encryption process is as follows:

- Byte Substitution
 - The 16 bytes are substituted by looking up a fixed table. The result is a matrix of four rows and four columns each representing a byte.
- ShiftRows
 - Each of the four rows are shifted to the left and entries that fall off are inserted on the right of the row and mixed to create a new matrix.
- MixColumns

- Each column of four bytes is now transformed using the AES mathematical function. The result is now a new matrix different from the previous matrixes in the first two steps.
- AddRoundKey
 - The 16 bytes of the matrix are now the 128 bits round key.

Figure 13 above shows the process of AES with the decryption process of the ciphertext back to plaintext is the same number of steps but in reverse.

2.15.3.2 Twofish

To use Twofish in my application required the use of the SpongyCastle API. SpongyCastle is a collection of APIs which contains cryptography libraries which support numerous encryption types, one of which is Twofish⁶. To use SpongyCastle, all I needed was to include SpongyCastle as a security provider in my application which can be seen in Figure 22.

```
Security.addProvider(new org.spongycastle.jce.provider.BouncyCastleProvider());
```

Figure 22 - Spongy Provider

Twofish is a symmetric block cipher, has a block size of 128 bits, and supports crypto key lengths of 128, 192 and 256 bits to encrypt and decrypt data in blocks of 128 bits. Twofish is an improvement on the Blowfish encryption algorithm.

The Twofish cipher I chose to implement is:

- 256-bit key
- 128-bit random IV
- Cipher Block Chaining (CBC)
- PKCS5Padding
- PBKDF2 Hashing

⁶ https://www.schneier.com/academic/archives/1998/12/the_twofish_encrypti.html

The reason for this implementation choice is that CBC makes it more difficult to perform known-plaintext attacks on your ciphertext (Jeff Nelson, 2014). Moreover, PKCS5Padding causes the cipher to pad the data, making it more difficult to attack the ciphertext directly. Figure 23 shows the Twofish parameters such as the algorithm used, the key size and the IV size.

```
private String KEY = "Twofish";
private String CIPHER = "Twofish/CBC/PKCS5Padding";

private byte[] KEY_SIZE;
private byte[] IV_SIZE;
private String IV;

TwoFish() {
    Security.addProvider(new org.spongycastle.jce.provider.BouncyCastleProvider());

    KEY_SIZE = new byte[32]; //256
    IV_SIZE = new byte[16]; //128
    IV = "038e271f445d92f8";
}
```

Figure 23 - Twofish

The method below is used to generate a new random IV.

```
private static byte[] generateRandomIV(int length)
{
    byte[] iv = new byte[length];
    SecureRandom random = new SecureRandom();
    random.nextBytes(iv);

    return iv;
}
```

Figure 24 - Random IV

The encryption method, seen in figure 25, takes both a file in byte format and a string pin code, so the file which is user selected is converted to a byte array before being passed to the method, the method handles the string pin code being converted to bytes, which is used to create the key. Before the pin code is converted to a byte array is it hashed and salted using PBKDF2 to a length of 256 bits.

The encrypted bytes are then uploaded to Firebase Storage with the pin code needed to decrypted being stored in the Firebase Database.

In Firebase Storage, uploaded bytes that are not encrypted can be seen as whatever content type they are, such as .png or .pdf. The encrypted bytes content type is “application/octet-stream”, this is because Firebase does not know what content-type it is, because the bytes are encrypted.

```
int passBytes = password.getBytes( charsetName: "UTF-8").length;
byte[] iv = generateRandomIV(IV_LENGTH);
IvParameterSpec ivSpec = new IvParameterSpec(iv);

if (password.getBytes( charsetName: "UTF-8").length > KEY_SIZE.length)
    passBytes = KEY_SIZE.length;
int ivlen = iv.length;
if(iv.length > IV_SIZE.length)
    ivlen = IV_SIZE.length;

System.arraycopy(iv, 0, IV_SIZE, 0, ivlen);
System.arraycopy(password.getBytes( charsetName: "UTF-8"), 0, KEY_SIZE, 0, passBytes);

SecretKeySpec secretKey = new SecretKeySpec(KEY_SIZE, KEY);

// get Cipher and init it for encryption
Cipher cipher = Cipher.getInstance(CIPHER);

if(mode.equals(EncryptMode.ENCRYPT)) {
    cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
    byte[] results = cipher.doFinal(file);

    byte[] combined = new byte[IV_LENGTH + results.length];
    System.arraycopy(iv, 0, combined, 0, IV_LENGTH);
    System.arraycopy(results, 0, combined, IV_LENGTH, results.length);

    return combined;
}

if(mode.equals(EncryptMode.DECRYPT)) {
    byte[] keyBytes = secretKey.getEncoded();
    SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, KEY);

    System.arraycopy(file, 0, iv, 0, iv.length);
    IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);

    cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, ivParameterSpec);

    out = cipher.doFinal(file);
}
```

Figure 25 - Twofish Algorithm

2.15.3.3 Triple DES

Triple DES is used to encrypt the user's images in the application. Triple DES is another mode of the DES operation. It uses three 64-bit keys for an overall key length of 192 bits. Triple DES breaks the original key into three separate keys and then pads these three keys to a 64-bit length which are then combined. Triple DES is a slow encryption algorithm but is still considered secure and more secure than the standard DES when implemented properly. Figure 26 shows the algorithm used and the methods used. It uses CBC mode, Cipher Block Chaining and is padded using PKCS5Padding⁷.

CBC makes it more difficult to perform known-plaintext attacks on your ciphertext (Jeff Nelson, 2014). PKCS5Padding causes the cipher to pad the data, making it more difficult to attack the ciphertext directly.

```
private static String TRIPLE_DES_TRANSFORMATION = "DESede/CBC/PKCS5Padding";
private static String ALGORITHM = "DESede";
```

Figure 26 - 3DES

Figure 27 shows the main part of the Triple DES process, the image is converted to a byte array before being passed to the method, the key is converted to bytes and a new IV is created each time. The key, image bytes and IV are combined to create the ciphertext, the encrypted byte array.

```
final byte[] keyBytes = PBK(pin);
Log.d(TAG, msg: "encrypt: " + Arrays.toString(keyBytes));
final SecretKey key = new SecretKeySpec(keyBytes, ALGORITHM);

final IvParameterSpec iv = new IvParameterSpec(new byte[8]);
final Cipher cipher = Cipher.getInstance(TRIPLE_DES_TRANSFORMATION);
cipher.init(Cipher.ENCRYPT_MODE, key, iv);

final byte[] plainTextBytes = message.clone();
final byte[] cipherText = cipher.doFinal(plainTextBytes);
```

Figure 27 - 3DES Encryption

⁷ https://www.tutorialspoint.com/cryptography/triple_des.htm

The key is hashed and salted using PBKDF2, the PBK method used can be seen in figure 28, from which a key is generated. This implementation of PBKDF2 is different to others in the application as it creates a 192-bit long hash instead of a 256-bit hash.

```
String PBK(String pin) throws NoSuchAlgorithmException, InvalidKeySpecException {  
  
    char[] chars = pin.toCharArray();  
    byte[] salt = getSalt(); //8 bytes  
  
    SecretKeyFactory factory = SecretKeyFactory.getInstance(PBKDF2);  
    KeySpec spec = new PBEKeySpec(chars, salt, ITERATIONS, BIT_LENGTH); //10000, 192 bit length/24 bytes  
    SecretKey key = factory.generateSecret(spec);  
    byte[] hash = key.getEncoded();  
  
    return toHex(salt) + ":" + toHex(hash);  
}
```

Figure 28 - 3DES & PBKDF2

The TripleDES encryption in C# can be seen below in figure 29, this implementation of Triple DES is the exact same as the Java one. This method allows users to encrypt images on the web application and then decrypt that image on the android application.

```
private byte[] desEncrypt(byte[] image, string pin)  
{  
    byte[] keyBytes = Encoding.UTF8.GetBytes(pin);  
    byte[] newBytes = new byte[24];  
    Array.Copy(keyBytes, 0, newBytes, 0, 24);  
    byte[] IV = new byte[8];  
  
    TripleDES tdes = TripleDES.Create();  
  
    tdes.Key = newBytes;  
    tdes.Mode = CipherMode.CBC;  
    tdes.Padding = PaddingMode.PKCS7;  
    tdes.IV = IV;  
  
    var Encryptor = tdes.CreateEncryptor();  
    byte[] output = Encryptor.TransformFinalBlock(image, 0, image.Length);  
    tdes.Clear();  
  
    return output;  
}
```

Figure 29 - C# Triple DES

2.15.3.4 Steganography

Steganography is the art of hiding information in plain sight, unlike encryption, where it is obvious that the information is encrypted and hidden. In the android application steganography is used to hide the users text blocks inside a generated bitmap which is 200x200 and will be a random colour. This image with the text hidden in it is uploaded to Firebase Storage, the user can then get the text back. The steganography class uses LSB, Least Significant Bit, this technique changes the last few bits in a byte to hide the message. If the last two bits of a red or blue pixel are changed to hide the text, the change in the colour is not noticeable in any way to the human eye. Figure 30 shows an example of the LSB operation⁸.

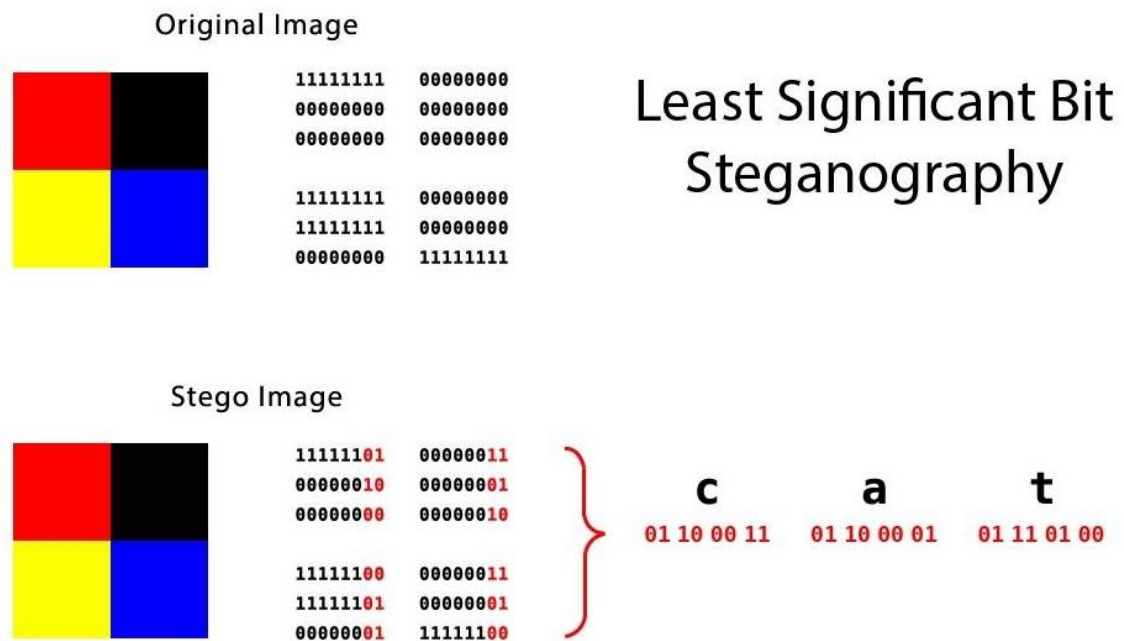


Figure 30 - Steg colour operation example

Source: <https://null-byte.wonderhowto.com/how-to/steganography-hide-secret-data-inside-image-audio-file-seconds-0180936/>

⁸<https://null-byte.wonderhowto.com/how-to/steganography-hide-secret-data-inside-image-audio-file-seconds-0180936/>

The method `testBitmapEncoder` uses the `Bitmap Helper` class to create a bitmap seen in figure 31, the class `Steg` is then used to combine the new bitmap and the users inputted text together. This new bitmap is then converted to a byte array and these bytes are returned.

```
private byte[] testBitmapEncoder(String textValue) {  
  
    byte[] hiddenBytes = textValue.getBytes();  
  
    Bitmap bitmap = BitmapHelper.createTestBitmap( w: 200, h: 200, color: null);  
  
    Bitmap newbit = null;  
    try {  
        newbit = Steg.withInput(bitmap).encode(textValue).intoBitmap();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    // BitmapEncoder.encode(bitmap, hiddenBytes);  
  
    ByteArrayOutputStream baos = new ByteArrayOutputStream();  
    assert newbit != null;  
    newbit.compress(Bitmap.CompressFormat.PNG, quality: 100, baos);  
  
    byte[] data = baos.toByteArray();  
  
    return data;  
}
```

Figure 31 - Steg Method

Figure 32 below shows the method which is used to create the bitmap which the text will be hidden in.

```
public static Bitmap createTestBitmap(int w, int h, @ColorInt Integer color) {  
    Bitmap bitmap = Bitmap.createBitmap(w, h, Bitmap.Config.ARGB_8888);  
    Canvas canvas = new Canvas(bitmap);  
  
    if (color == null) {  
        int colors[] = new int[] { Color.BLUE, Color.GREEN, Color.RED, Color.YELLOW, Color.WHITE };  
        Random rgen = new Random();  
        color = colors[rgen.nextInt( bound: colors.length - 1)];  
    }  
  
    canvas.drawColor(color);  
    return bitmap;  
}
```

Figure 32 - Bitmap Creator

2.15.3.5 PBKDF2

The hashing algorithm I will be using in the android and web application is PBKDF2, PBKDF2WithHmacSHA1 to be exact. The input, a six-digit pin code, is salted and hashed which is used as the key for other encryption algorithms, such as AES, Twofish and Triple DES, but is also so used so that the pin code the user sets is not stored in the database in plaintext.

The reason for choosing this hashing algorithm is because it is one of the recommended password based key derivation algorithms to use according to the NIST SP800-132, this is because it is slower than any of the older hashing techniques, can have a high number of iterations as well as a large bit length. All of these things make brute forcing attacks much slower and require more computing power to crack⁹.

The number of iterations, 10000, is also a high iteration count and is the recommended number from the OWASP Password Storage Cheat Sheet¹⁰ and NIST SP800-132 recommending a minimum of 1000. NIST SP800-132 also state that the salt should use an approved random bit generator, in this case secure random, and be at least 128 bits. The get salt method in figure 33 adheres to these guidelines and creates a cryptographic random 16-byte array which is used as the salt.

⁹ <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf>

¹⁰ https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

The method below in figure 33 is the Java implementation of the hashing algorithm PBKDF2 which takes the user supplied pin code, generates a secure random salt which is then used to generate the pin code hash. The salt is appended to the start of the hash, so it can be used again when validating the hash.

```
private int ITERATIONS = 10000;
private int KEY_LENGTH = 256;
private String ALGORITHM = "PBKDF2WithHmacSHA1";

public String PBK(String pin) throws NoSuchAlgorithmException, InvalidKeySpecException {

    char[] chars = pin.toCharArray();
    byte[] salt = getSalt();

    SecretKeyFactory factory = SecretKeyFactory.getInstance(ALGORITHM);
    KeySpec spec = new PBEKeySpec(chars, salt, ITERATIONS, KEY_LENGTH);
    SecretKey key = factory.generateSecret(spec);
    byte[] hash = key.getEncoded();

    return toHex(salt) + ":" + toHex(hash);
}

private byte[] getSalt() throws NoSuchAlgorithmException
{
    String SALT_ALGO = "SHA1PRNG";
    SecureRandom sr = SecureRandom.getInstance(SALT_ALGO);
    byte[] salt = new byte[16];
    sr.nextBytes(salt);
    return salt;
}

private static String toHex(byte[] bytes) throws NoSuchAlgorithmException
{
    StringBuilder builder = new StringBuilder();

    for (byte b: bytes) {
        builder.append(String.format("%02x", b));
    }
    return builder.toString();
}
```

Figure 33 - PBKDF2

The method below in figure 34 is the C# implementation of the hashing algorithm PBKDF2 which takes the user supplied pin code, generates a secure random salt which is then used to generate the pin code hash. This hashing algorithm generates the same hash text as the Java implementation, this is so that the user data can be decrypted on both platforms.

```
public static string Hash(string pin)
{
    int ITERATIONS = 10000;
    int BIT_LENGTH = 32;

    byte[] salt = GetSalt();

    var kf = new Rfc2898DeriveBytes(pin, salt, ITERATIONS);

    byte[] key = kf.GetBytes(BIT_LENGTH);

    string hashed = ByteArrayToHexString(salt) + ":" + ByteArrayToHexString(key);

    return hashed;
}

2 references | 0 exceptions
private static string ByteArrayToHexString(byte[] byteArray)
{
    StringBuilder builder = new StringBuilder();
    foreach (byte b in byteArray)
    {
        builder.Append(b.ToString("x2"));
    }
    return builder.ToString();
}

1 reference | 0 exceptions
private static byte[] GetSalt()
{
    RNGCryptoServiceProvider provider = new RNGCryptoServiceProvider();
    byte[] salt = new byte[16];
    provider.GetBytes(salt);

    return salt;
}
```

Figure 34 - PBKDF2 C#

The hashes are validated using a hash validation method, which can be seen below in figure 35 and 36, when a user tries to decrypt a password or other data. If the hashes match, then the data can be decrypted. The reason for this method is because each hash is unique because of the random salt.

Figure 35 shows the Java implementation of the hash validation. The salt and the hash are separated by a “.” character so the salt is part 0 of the string and the actual hash is part 1. The method creates a new hash using the same salt as well as the user inputted pin code. If the pin code is correct it will create the same hash as all the variables are the exact same. If the hashes match the user can decrypt the data, if they don’t match the user has to enter the pin code again.

```
public boolean validatePassword(String originalPassword, String storedPassword) throws NoSuchAlgorithmException
{
    String[] parts = storedPassword.split( regex: "." );
    int iterations = 10000;
    byte[] salt = fromHex(parts[0]);
    byte[] hash = fromHex(parts[1]);

    PBESpec spec = new PBESpec(originalPassword.toCharArray(), salt, iterations, keylength: 256);
    SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
    byte[] testHash = skf.generateSecret(spec).getEncoded();

    int diff = hash.length ^ testHash.length;
    for(int i = 0; i < hash.length && i < testHash.length; i++)
    {
        diff |= hash[i] ^ testHash[i];
    }
    return diff == 0;
}

private static byte[] fromHex(String hex) throws NoSuchAlgorithmException
{
    byte[] bytes = new byte[hex.length() / 2];
    for(int i = 0; i < bytes.length ;i++)
    {
        bytes[i] = (byte)Integer.parseInt(hex.substring(2 * i, 2 * i + 2), radix: 16);
    }
    return bytes;
}
```

Figure 35 - Hash Validation Java

Figure 36 shows the C# implementation of the hash validation method, it has to operate the exact same way as the Java version so that the users pin codes can be authenticated on both platforms.

```
public bool ValidateHash(string originalPassword, string storedPassword)
{
    String[] parts = storedPassword.Split(':');
    byte[] salt = fromHex(parts[0]);
    byte[] hash = fromHex(parts[1]);

    int ITERATIONS = 10000;
    int BIT_LENGTH = 32;

    var kf = new Rfc2898DeriveBytes(originalPassword, salt, ITERATIONS);

    byte[] testHash = kf.GetBytes(BIT_LENGTH);

    int diff = hash.Length ^ testHash.Length;
    for (int i = 0; i < hash.Length && i < testHash.Length; i++)
    {
        diff |= hash[i] ^ testHash[i];
    }
    return diff == 0;
}
```

Figure 36 - Hash Validation C#

2.15.4 Technologies

2.15.4.1 HTML5, Bootstrap & JavaScript

HTML is a markup language that describes the structuring of the content seen on web pages. I used HTML and custom CSS for developing the frontend of the web application in order to create a clean, highly usable and professional looking UI.

Bootstrap is an open source toolkit for developing applications in HTML, CSS and JavaScript, it allowed me to create a highly responsive and professional looking website using the various CSS classes. I also used a Bootstrap theme from bootswatch.com called LUX which gave the site a flat, simplistic but more modern look.

To use Firebase in the web application the Firebase methods had to be coded in JavaScript, JavaScript is a cross platform, object-oriented scripting language used

to make web pages interactive. I used client-side JavaScript throughout the web application to communicate with the database and storage, to provide the login and register functionality and to change the way the webpages(DOM) look¹¹.

2.15.4.2 Google Firebase

Google Firebase is a cloud service which gives you functionality such as analytics, databases, crash reporting and authentication as well as many others. Firebase is built on Google Infrastructure and scales automatically to accommodate a growing user base. Firebase supports android, iOS, web, Unity and C++, this is the reason I choose it as it allowed me to share functionality and data across two platforms¹².

For my project I used Firebase Authentication, Database and Storage. The Firebase database is a cloud hosted NoSQL Realtime database that lets users read and write data in Realtime, even across different platforms. The database also integrates with Firebase Authentication to provide security features such as the database rules. Because the database is a NoSQL database it allowed me to redesign my database layout with ease while not altering the data already there, also I don't need to worry about SQL injections

Firebase Storage allowed me to quickly and easily store and serve the users data, such as files, images and bitmaps. The user's files and images are stored in buckets with the Firebase SDK allowing me to design the layout of each bucket

Firebase Authentication is a user authentication system which provides end to end identification. It supports email and password, phone Auth, Google, Twitter, Facebook and GitHub login and sign-up functionality. In my application I used the email and password sign-up methods on both applications as well as the Google

¹¹ <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>

¹² firebase.google.com

2.15.4.3 Java & C#

Java is a high level object-oriented programming language. Java programs contain classes which are used to define objects and methods. Android applications are developed in Java and it is used to create methods and interact with the apps frontend. Java allowed me to implement a huge amount of functionality and interactive features in the application. Firebase also supports Android, so I could set up the Firebase SDK in my application and implement the login and register functionality, the encryption algorithms and the methods which were used to read and write data to the database. Using Java and Firebase allowed me to implement Auth instances, sessions, which provided some security by making users log in to use the features in the application¹³.

C# is also a high level object-oriented programming language used to develop secure and robust applications that run on the .NET Framework. The C# syntax is very similar to the Java syntax so if a developer knows one they can pick up the other in a short amount of time. The web application will use C#, as well as JavaScript, to create the methods and functionality of the application. The reason I chose C# is because most encryption algorithms are interoperable which means I can code an AES or PBKDF2 method in Java and then code the same methods in C# and produce the same output, which is the key to my cross-platform application¹⁴.

¹³ <https://www.w3schools.in/java-tutorial/intro/>

¹⁴ <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>

2.16 Testing

Testing was carried out on both the Android and web application. Testing included security testing, UI testing and usability testing.

2.16.1 Code Analysis

Android Studio allows the developer to inspect the code in the application which will provide the developer with an overview of all issues and warnings in the code, seen in figure 37. It also allows you to quickly and easily remedy these warnings which makes the code much more streamline, readable and helps mitigate against errors or bugs in the future.



Figure 37 - Android Studio Code Analysis

Some of errors and warnings that were fixed using this were methods not being declared private, classes that could be declared as package-private and variables that could be local or declared as final.

2.16.2 Unit Testing

Unit tests were created to test the features and functionality of the android application. The tests mainly included encryption unit tests, authentication unit tests as well as others. Some of the unit tests where designed to fail.

Validate Password Unit Test

The unit test shown in figure 38 was designed to verify that the hash validation method works as intended and only passed if the pin code entered matched the stored hash. The unit tests passed.

```
@Test
public void validatePassword() throws Exception {

    Hash hash = new Hash();
    String testVal = "085713";
    String savedHash = "2fea20deb993a8a62737f346da5d0751:040f5d8f12c9fe0e7a7a486e46476e7a849a515f835e358b31a0e4efd5952fal";
    boolean match = hash.validatePassword(testVal, savedHash);

    assertEquals( expected: true, match);
}
}
```

Figure 38 - Hash Validation Unit Test

Changing the hash or the pin code causes the unit test to fail, which is how it should work.

```
▼ HashTest (x14532757.softwareproject.Utils) 177ms "C:\Program Files\Android\Android Studio\jre\bin\
  validatePassword 177ms
  java.lang.AssertionError:
  Expected :true
  Actual   :false
  <Click to see difference>
```

PBKDF2 Hashing Unit Test

This unit test shown in figure 39 was created to fail. I hashed the pin code “085085” which was stored in the database, then created a unit test to hash the same pin code multiple times. The same hash shouldn’t be created.

```
@Test
public void PBK() throws Exception {

    Hash hash = new Hash();
    String testVal = "085085";
    String hashed = hash.PBK(testVal);
    String savedHash = "63920b7396989acbae80100e0f7fe231:28055a01714dbbbfdf5c7eac48598727e800a304d7e63d061a67c09945b23293";

    assertEquals(savedHash, hashed);
}
```

Figure 39 - PBKDF2 Hash Unit Test

The test fails each time you run the test which shows each hash is unique even when hashing the same pin code.

```
HashTest (x14532757.softwareproject.Utils) 167ms "C:\Program Files\Android\Android Studio\jre\bin\java" ...
  PBK 167ms
org.junit.ComparisonFailure:
Expected :63920b7396989acbae80100e0f7fe231:28055a01714dbbbfdf
Actual   :f8e484e259fa48e95de045ef2b53f895:9701645d7127aeeb64
<Click to see difference>
```

Special Characters Input validation

These unit tests were created to verify the input validation method created to make sure that no characters needed for NoSQL injections were allowed. Both tests were created to pass with one containing special characters and one containing no special characters. Figure 40 shows the unit tests created to test the input validation methods.

```
@Test
public void SpecialChars() throws Exception {

    String textValue = "Password!";

    InputValidation validation = new InputValidation();
    List<String> errorList = new ArrayList<>();
    boolean isValid = validation.NoSpecialChars(textValue, errorList);

    assertEquals( expected: true, isValid);

}

@Test
public void noSpecialChars() throws Exception {

    String textValue = "Password";

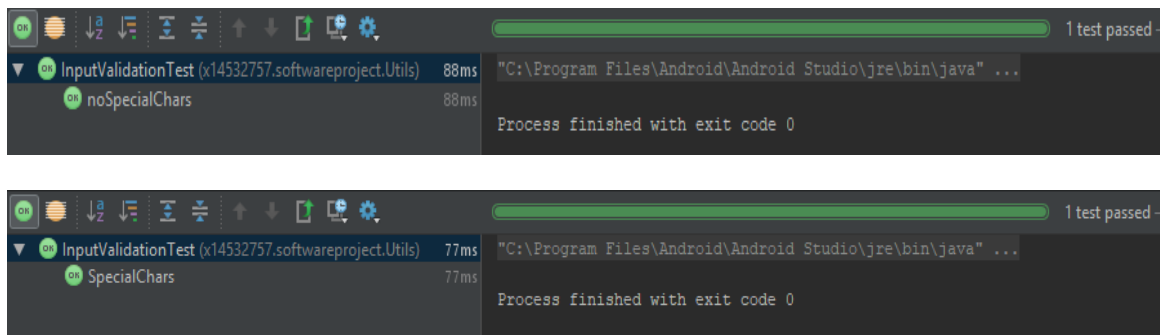
    InputValidation validation = new InputValidation();
    List<String> errorList = new ArrayList<>();
    boolean isValid = validation.NoSpecialChars(textValue, errorList);

    assertEquals( expected: false, isValid);

}
```

Figure 40 - Input Validation Unit Test

Both of the tests in figure 40 passed which can be seen in the result screenshots below.



2.16.3 Usability Testing

The usability testing is done by getting users to use the application to learn how user friendly the GUI is and how easily the users can navigate and use the features in the application. The testing used falls under the method of black box testing, where the user only sees the GUI and the internal structure and design of the application is unknown to the tester. The tests can be functional or non-functional, the test I will create will be functional tests such as a tester using the login, encrypting data and decrypting data functionality. I chose two users to test application, these testers were ones I could sit down with and go through the test with in person, this allowed me to get some extra feedback as well. I created a small questionnaire which they had to fill out and then asked them to move through the application using the features and I documented whether the features and functionalities passed, failed or crashed. The questions for both the android and web application are very similar, the questions and the answers from the survey are seen in figure 41 and figure 42 below.

Questionnaire - Android

What is your first impression of the application?
Tester 1 - It looks very professional.
Tester 2 - Splash screen is a nice touch, well designed.
Do you like the design of the application? (Colours, Layout)
Tester 1 - Colour scheme is very good, consistent colour scheme. Layout is very clear and features clearly laid out.
Tester 2 - Yes, colours are bright and vibrant, all the features are clearly laid out and easy to understand.
Is it easy to navigate through the application?
Tester 1 - Yes
Tester 2 - Yes

How easy is it encrypt and decrypt data?
Tester 1 - Very Easy
Tester 2 - Very Easy
Would you use this application to keep your data safe?
Tester 1 - Yes
Tester 2 - Yes

Figure 41 - Android Questionnaire

Questionnaire - Web

What is your first impression of the application?
Tester 1 - Consistent with the android app colour scheme, has a very business/industrial feel, professional.
Tester 2 - Sleek and Minimalistic.
Do you like the design of the application? (Colours, Layout)
Tester 1 - Yes, the colour scheme is very easy on the eyes, the layout is very easy to understand, and the landing page contains all relevant information about features and contact information.
Tester 2 - Yes, the colour scheme is very coordinated with the app, webpages laid out very well.
Is it easy to navigate through the application?
Tester 1 - Yes
Tester 2 - Yes
How easy is it encrypt and decrypt data?
Tester 1 - Very Easy
Tester 2 - Easy
Would you use this application to keep your data safe?
Tester 1 - Yes
Tester 2 - Yes

Figure 42 - Web Questionnaire

User Application Testing

I asked both testers to go through the steps lined out below and document whether the tasks passed or failed, or if something unintended happened such as a crash or a bug. The android application test which can be seen in figure 43 details the steps, expected results, actual result and whether the step passed or failed. Both the user's tests where combined as all steps on both tests passed.

Step #	Step Details	Expected Results	Android Actual Result	Pass/Fail/Crash
1	Open Application	Splash Screen, then to login choice screen	As expected	PASS
2	Click Login	Should go to login screen	As expected	PASS
3	Register	Should allow an new user account to be created	As expected	PASS
4	Login	Should allow a user to login with correct details, and go to home page	As expected	PASS
5	Click View Account	Account page displays users account information	As expected	PASS
6	Click View Password	Should display empty list	As expected	PASS
7	Encrypt new Password	User encrypts a new password which is stored in Database	As expected	PASS
8	Click list with password	Encrypted password now populate list, when a user clicks on a password, application goes to decrypt page	As expected	PASS
9	Decrypt password	User, using pincode, should be able to decrypt a password and decrypted password display on screen	As expected	PASS
10	Click View Images	Should display empty list	As expected	PASS
11	Encrypt new Image	User should be able to select an image from phone storage and encrypt it, saved to database and storage.	As expected	PASS
12	Click list with image	Encrypted images now populate list, when a user clicks on an image name, application goes to decrypt page	As expected	PASS
13	Decrypt image	User, using pincode, should be able to decrypt a image and decrypted image display on screen	As expected	PASS
14	Logout	User should be able to logout, then taken to login screen.	As expected	PASS

Figure 43 - Android Tests

The web application passed all tests except the last one, for some reason when the user clicked the logout button they were still logged in, but if logged out again then they would be logged out properly. I didn't get this error before, so I later found the cause and fixed it. The cause appeared to be when the web application was running on Microsoft Edge, this error didn't occur in Google Chrome. The error was caused by a popup that told the user they were logged out, this seemed to interrupt the logout process, removing it solved the issue. The details of each step expected result, actual result and whether the step passed or failed can be seen in figure 44.

Step #	Step Details	Expected Results	Web Actual Result	Pass/Fail/Crash
1	Open Application	homescreen displays	As expected	PASS
2	Click Login	Should go to login screen	As expected	PASS
3	Register	Should allow an new user account to be created	As expected	PASS
4	Login	Should allow a user to login with correct details, and go to user home page	As expected	PASS
5	Click View Password	Should display empty list	As expected	PASS
6	Encrypt new Password	User encrypts a new password which is stored in Database	As expected	PASS
7	Click list with password	Encrypted password now populate list, when a user double clicks on a password, decrypt section fills with relevant data	As expected	PASS
8	Decrypt password	User, using pincode, should be able to decrypt a password and decrypted password display on screen	As expected	PASS
10	Click View Images	Should display empty list/or encrypted images from android	As expected	PASS
11	Click View Text	Should display empty list/or encrypted text blocks from android	As expected	PASS
12	Click View Files	Should display empty list/or encrypted files from android	As expected	PASS
13	Logout	User should be able to logout, then taken to login screen.	Have to log out twice.	FAIL

Figure 44 - Web Tests

2.16.4 GUI Testing

GUI testing is the task of testing the systems user interface of the application. This involves checking each screen that contain menus, icons, lists, buttons etc. for GUI bugs. The GUI determines how user friendly the application is, so testing was conducted on both the android and web application to ensure that the UI behaved how it is meant to and that the user can move seamlessly through the applications screens. I performed testing on each screen making sure that the various buttons performed their tasks effectively and that all lists of data and intents functioned correctly. The GUI was also tested after each new view was created on both platforms. All GUI's of the applications function correctly and as intended.

Robo Test

Robo test is a tool integrated into Firebase Test Labs for Android. Robo tests analyze the structure of the applications UI and moves through the application simulating user's activities. Robo test captures log files, screenshots and creates an activity map, these can be used to help determine causes of application crashes or where GUI bugs occur. I ran multiple tests using the Kryptium APK on SDK versions 23 and 26.

SDK version 23 is the minimum SDK version while 26 is the target SDK version. Both Robo tests passed and the results can be seen below in figure 45.

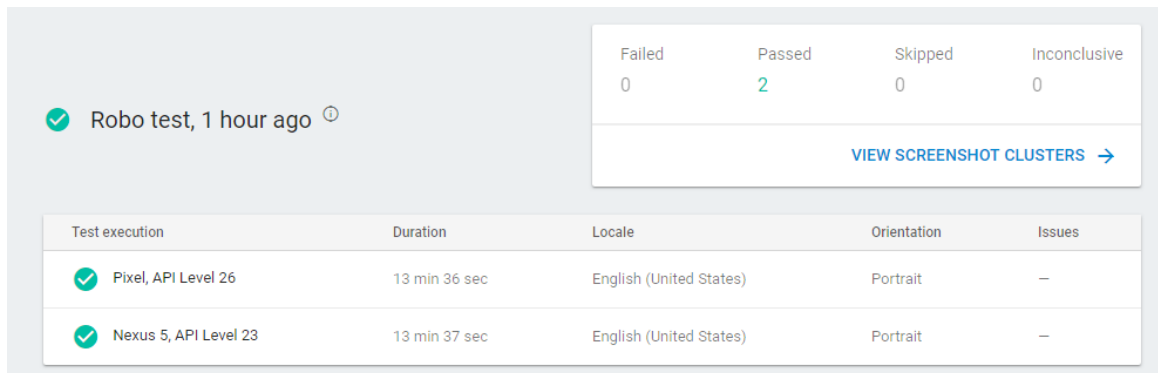


Figure 45 - Robo Test Result

2.16.5 Security Testing

ImmuniWeb - Android

High-Tech Bridge, a web security company, which provide several free and premium services, one of which is ImmuniWeb. ImmuniWeb allows you to upload an APK file which it will perform a security audit on. It tests the security and privacy of mobile applications as well as detects OWASP Mobile Top 10 weaknesses. Below is a summary of the results of the test.

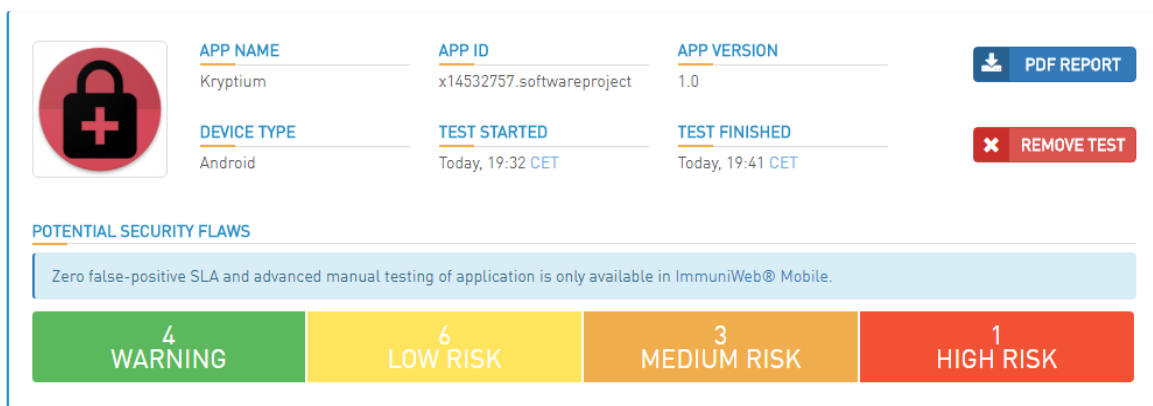


Figure 46 - Android Security Test Overview Source: <https://www.htbridge.com/mobile/?id=2DD4UonF>

The one potentially high-risk security flaw, figure 47, is that my application is writing data to the devices public storage, internal memory or SD card, which could then be accessed by other applications or by other users. This flaw was found in the decrypt files class of the application, but it is nothing to worry about as the “flaw” is the class downloading the decrypted file (pdf, word) to the downloads folder on the device. This must occur if the user is to use the file again.

EXTERNAL DATA STORAGE [M2] [CWE-921] [SAST]

HIGH

Description:

The mobile application can access external storage (e.g. SD card) in read or write mode. The application's data stored on the external data storage may be accessed by other applications (including malicious ones) under certain conditions and bring risks of data corruption or tampering.

Figure 47 - High Risk

The next potential flaw was deemed a medium security risk, seen below in figure 48. It was that my application gradle file included some third-party libraries. I went through my gradle file and removed some of the libraries I was no longer using or no longer needed. The rest of the gradle file consisted of the Google Firebase, SpongyCastle, BouncyCastle and android libraries which are all fine as well as needed for the application to function correctly.

SOFTWARE COMPOSITION ANALYSIS [M7] [SAST]

MEDIUM

Description:

The mobile application uses third-party libraries that may represent a security and privacy risk if they come from untrusted source or are outdated. Trusted and commonly accepted libraries (e.g. Google SDK, Facebook SDK, Signal SDK) are not displayed.

Details:

There are 8 used libraries:

- junit.extensions
- bolts
- junit.runner
- junit.textui
- se.simbio
- third.part
- org.hamcrest
- junit.framework

Figure 48 - Medium Risk 1

The next medium flaw was in one of my steganography classes I was using Random, which is a predictable random number generator, instead of SecureRandom. This was fixed by changing it to SecureRandom instead.

PREDICTABLE RANDOM NUMBER GENERATOR [M5] [CWE-338] [SAST] MEDIUM

Description:
The mobile application uses a predictable Random Number Generator (RNG). Under certain conditions this weakness may jeopardize mobile application data encryption or other protection based on randomization. For example, if encryption tokens are generated inside of the application and an attacker can provide application with a predictable token to validate and then execute a sensitive activity within the application or its backend.

Figure 49 - Medium Risk 2

Another flaw introduced a possible exploit an attacker could use to create unintended touch activities on the android application called Tap jacking.

MISSING TAPJACKING PROTECTION [M1] [CWE-451] [SAST] LOW

Tap jacking is when a user touches the screen, the application may pass the touch event to another application below its user interface layer that the user does not see. To mitigate against this potential attack involved adding a line code, seen in figure 50, below each button that could have an overlay above it, such as the fingerprint authentication overlay.

```
Button upload = findViewById(R.id.uploadBtn);
upload.setFilterTouchesWhenObscured(true);
```

Figure 50 - Tap Jacking Fix

Another low threat vulnerability was that the application was still in debug mode. Debug mode can be set from true to false easily if the application was moved from development to production.

ENABLED DEBUG MODE [M2] [CWE-921] [SAST] LOW

Figure 51 - Low Risk

The remainder of the low threat flaws and warnings were mainly due to the fact the application was still in its development configuration as seen in figure 52. It also showed some android libraries and code as vulnerable. Below is a screenshot of the low threat vulnerabilities, two of which are addressed above.



ENABLED DEBUG MODE [M2] [CWE-921] [SAST]	LOW
HARDCODED DATA [M2] [CWE-200] [SAST]	LOW
MISSING TAPJACKING PROTECTION [M1] [CWE-451] [SAST]	LOW
EXPORTED SERVICES [M1] [CWE-926] [SAST]	LOW
EXPOSURE OF POTENTIALLY SENSITIVE DATA [M2] [CWE-200] [DAST]	LOW

Figure 52 – Additional Issues

To be sure these were the only flaws and vulnerabilities I also used two other mobile application vulnerability tools hosted on the internet. One being Osterlab and Quixxi. Both security scans returned more or less the same results.

Quixxi- Android

The Quixxi results shown in figure 53 show that that the app is debuggable and the only high-risk vulnerability being that the app is signed with a 'Android Debug' certificate. These issues can be easily solved if I moved the application to a production phase. I also noticed that I was still logging sensitive information, a medium risk vulnerability, from when I was testing the encryption methods, so I went through the classes and removed logging from these methods that were not needed. The risks and issues produced by the Quixxi scan were remedied mostly by what I had done to remedy the issues introduced by the ImmuniWeb Scan.








 PROPERLY SIGNED CHECK Severity: High
 IS APP DEBUGGABLE Severity: Medium
 MISSING USAGE OF NATIVE(C, C++) CODE Severity: Medium
 OUTPUTTING LOGS TO LOGCAT/ LOGGING SENSITIVE INFORMATION Severity: Medium
 PROTECTION OF TEXT FIELDS FROM COPYING THE TEXT AND PASTE OUTSIDE YOUR APP Severity: Medium
 PROTECTION OF CAPTURING SCREENSHOTS & SHARING SCREENS OUTSIDE YOUR APP Severity: Medium
 PROTECTION OF APP SCREENS BY BLURRING WHEN THE APP IS RUNNING IN BACKGROUND Severity: Medium

Figure 53 - Quixxi Results

OWASP ZAP - Web

I used OWASP ZAP¹⁵, version 2.7.0, to conduct a vulnerability scan of the web application. I set ZAP to attack mode, so it would automatically fill any input fields with random strings and do a full scan of the web application. In figure 54 below you can see five potential vulnerabilities returned, one medium and four low risk threats with no high-risk threats found.

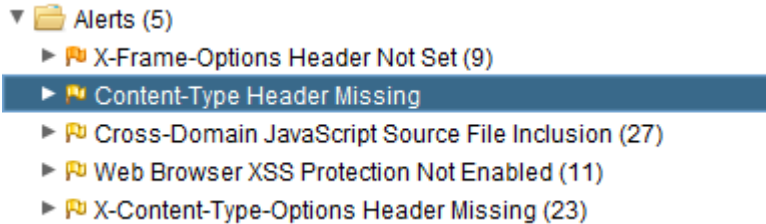


Figure 54 - OWASP ZAP Overview

The Web Browser XSS Protection Not Enabled warning is an alert to tell the developer that there is no XSS protection enabled on the web application. So, to fix this I had to modify the Web.config to including the code seen the in figure 55 below. This turns on the XSS filter provided by web browsers, if any scripts from user inputs are detected the browser will sanitize the page as well as block the render of the webpage.

```
<httpProtocol>
  <customHeaders>
    <add name="X-Xss-Protection" value="1; mode=block" />
  </customHeaders>
</httpProtocol>
```

Figure 55 – XSS Protection

¹⁵ OWASP ZAP Software - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

The potential Cross-Domain JavaScript Source File Inclusion vulnerability is a warning from OWASP ZAP saying that I have third party JavaScript on the web application, all of which are the required Google Firebase JavaScript files needed to use Google Firebase on the web application. This warning was ignored as I made sure to only use what was needed for the functionality of each page.

The last warning, X-Content-Type-Options Header Missing, is because the web application is vulnerable to drive by downloads and MIME type sniffing. To fix this issue also required adding the code seen in figure 56 below to the Web.config file and declaring all content types used by the web application as “no sniff”.

```
<httpProtocol>
  <customHeaders>
    <remove name="X-Content-Type-Options" />
    <add name="X-Content-Type-Options" value="nosniff" />
  </customHeaders>
</httpProtocol>
```

Figure 56 - No Sniff

The highest warning from OWASP ZAP was that X-Frame-Options Header Not Set. Ignoring this warning means the web application is vulnerable to clickjacking attacks but can be mitigated against by adding the code in figure 57 to the Web.config file. Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites (X-Frame-Options, MDN web docs).

```
<httpProtocol>
  <customHeaders>
    <add name="X-Frame-Options" value="SAMEORIGIN" />
  </customHeaders>
</httpProtocol>
```

Figure 57 - Click Jacking

After testing the web application and scanning it a number of times I was satisfied I had addressed all the issues presented by OWASP ZAP.

3 Further Development

While the android application is completed and the web application being in a beta stage further development can still be made in both.

Web

One of the further developments that could be made is the completion of the web application including all its features.

More Encryption

The system could evolve to include more features on both the android application and the web application. Features such as more encryption options, allowing the user to decide what to encrypt with, and more types of data to encrypt such as video or audio.

Language

For a further development, the application could include multiple languages allowing users all over the world to select their own language to view the app in.

4 Conclusion

The application Kryptium allows users on both android and web platforms to encrypt personal information safely, overall the android application was completed with the web application being a beta stage and not containing all but two encryption algorithms. The design and technologies chosen for implementation, such as Google Firebase, allowed for the development of cross platform authentication and data storage as user credentials and the users data was stored in the cloud. Using HTML5, JavaScript and C# allowed me to create a professional looking and usable web application while Java and XML allowing me to create a native android application.

5 References

- Brijesh Thumar. 2017. *Firestore Database for Android Application*. [ONLINE] Available at: <http://androidkt.com/firebase-firestore/>. [Accessed 11 October 2017].
- Kryptotel. 2017. *Encryption Algorithms*. [ONLINE] Available at: <http://learning.kryptotel.net/encryption-algorithms/>. [Accessed 9 October 2017].
- B. Schneier. 1998. *The Twofish Encryption Algorithm*. [ONLINE] Available at: https://www.schneier.com/academic/archives/1998/12/the_twofish_encrypti.html. [Accessed 16 October 2017].
- tutorialspoint. 2018. *Advanced Encryption Standard*. [ONLINE] Available at: https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm. [Accessed 16 January 2018].
- BLACK SLASH. 2017. *How to Hide Secret Data Inside an Image or Audio File in Seconds*. [ONLINE] Available at: <https://null-byte.wonderhowto.com/how-to/steganography-hide-secret-data-inside-image-audio-file-seconds-0180936/>. [Accessed 31 January 2018].
- Microsoft. 2015. *Introduction to the C# Language and the .NET Framework*. [ONLINE] Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>. [Accessed 22 February 2018].
- Google. 2018. *Cloud Storage for Firebase*. [ONLINE] Available at: <https://firebase.google.com/products/storage/>. [Accessed 2 May 2018].
- Google. 2018. *Firebase Realtime Database*. [ONLINE] Available at: <https://firebase.google.com/products/realtime-database/>. [Accessed 2 May 2018].
- Google. 2018. *Firebase Authentication*. [ONLINE] Available at: <https://firebase.google.com/products/auth/>. [Accessed 2 May 2018].
- TechTerms. 2012. *Java*. [ONLINE] Available at: <https://techterms.com/definition/java>. [Accessed 9 April 2018].
- MDN web docs. 2018. *What is JavaScript?*. [ONLINE] Available at: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>. [Accessed 17 April 2018].
- ImmuniWeb® Platform. 2018. *Mobile App Scanner*. [ONLINE] Available at: <https://www.htbridge.com/mobile/?id=t8W7ZSL2>. [Accessed 2 April 2018].
- Software Testing Fundamentals. 2018. *Black Box Testing*. [ONLINE] Available at: <http://softwaretestingfundamentals.com/black-box-testing/>. [Accessed 28 March 2018].
- OWASP.org. 2017. *Testing for NoSQL injection*. [ONLINE] Available at: https://www.owasp.org/index.php/Testing_for_NoSQL_injection. [Accessed 9 April 2018].
- Android Developers. 2017. *Permissions Overview*. [ONLINE] Available at: <https://developer.android.com/guide/topics/permissions/overview.html>. [Accessed 13 March 2018].

- IT FREEDOM. 2017. *The New NIST SP 800-63 Password Guidelines*. [ONLINE] Available at: <https://www.itfreedom.com/nist-sp-800-63-password-guidelines/>. [Accessed 30 March 2018].
- keycdn. 2017. *X-XSS-Protection – Preventing Cross-Site Scripting Attacks*. [ONLINE] Available at: <https://www.keycdn.com/blog/x-xss-protection/>. [Accessed 17 April 2018].
- MDN web docs. 2018. *X-Content-Type-Options*. [ONLINE] Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>. [Accessed 17 April 2018].

6 Appendix

6.1 Project Proposal

6.1.1 Objectives

I will be developing an android and web application that will have a login and register and allow users to encrypt sensitive information or just content they want to make more secure.

The user must be able to register an account and login, from there they must be able to easily navigate the applications and encrypt and decrypt data with ease. The user will also can decrypt data using their fingerprint should their device have a fingerprint scanner. The user must also be able to view, encrypt and decrypt files on both the android application and web application.

The android application will allow the user to encrypt and save passwords, text blocks, entire files and images using secure encryption algorithms. This data will be saved in Google Firebase database and storage. To decrypt the data a user must enter a pin code they entered when they first encrypt the data, this acts as an extra safety feature. The user will also have the option of decrypting data user their fingerprint should their mobile device have a fingerprint scanner.

The web application will be a carbon copy of the android application and will have the same features and functionality as it. The web application will allow the user to login and see all the encrypted data they have in the database and if they wish to use the file on their PC or laptop they can decrypt and download it using the same pin code they set when they encrypted it on their mobile device and vice versa. The web application will only be a prototype not encompassing all features depending on how long the android application takes.

I will also be using different encryption algorithms for each section in the application, so the passwords won't be encrypted the same as the images, files or the text blocks. The objective of this is to show how different encryption algorithms can be used in the same application. I will also be using the Google Firebase Database and Firebase storage to save the text or files that have been encrypted as well as using Google Authentication for login and register.

The android application will be developed in Android Studio using Java, xml and Firebase. The web application will be developed in Visual Studio using HTML, CSS, jQuery C# and Firebase. The database I will be using is the Google Firebase Database.

6.1.2 Background

In my research I found there were very few encryption applications on the Google Play Store that could be considered up to par and usable. I found that most apps were old and had UI that didn't have usability in mind as well as some apps using non-secure encryption algorithms. While there were one or two good apps in terms of features they had outdated UI and were over complicated with too many options and buttons which probably left some users confused and wanting to uninstall the app. So, I wanted to make an app that had usability, good UI design as well secure encryption features in mind.

As I am in the Cyber Security stream I wanted to double down on this idea and include as many security and encryption algorithms as possible without making it too complex. In the end I settled for 5 different ways of encrypting data, including

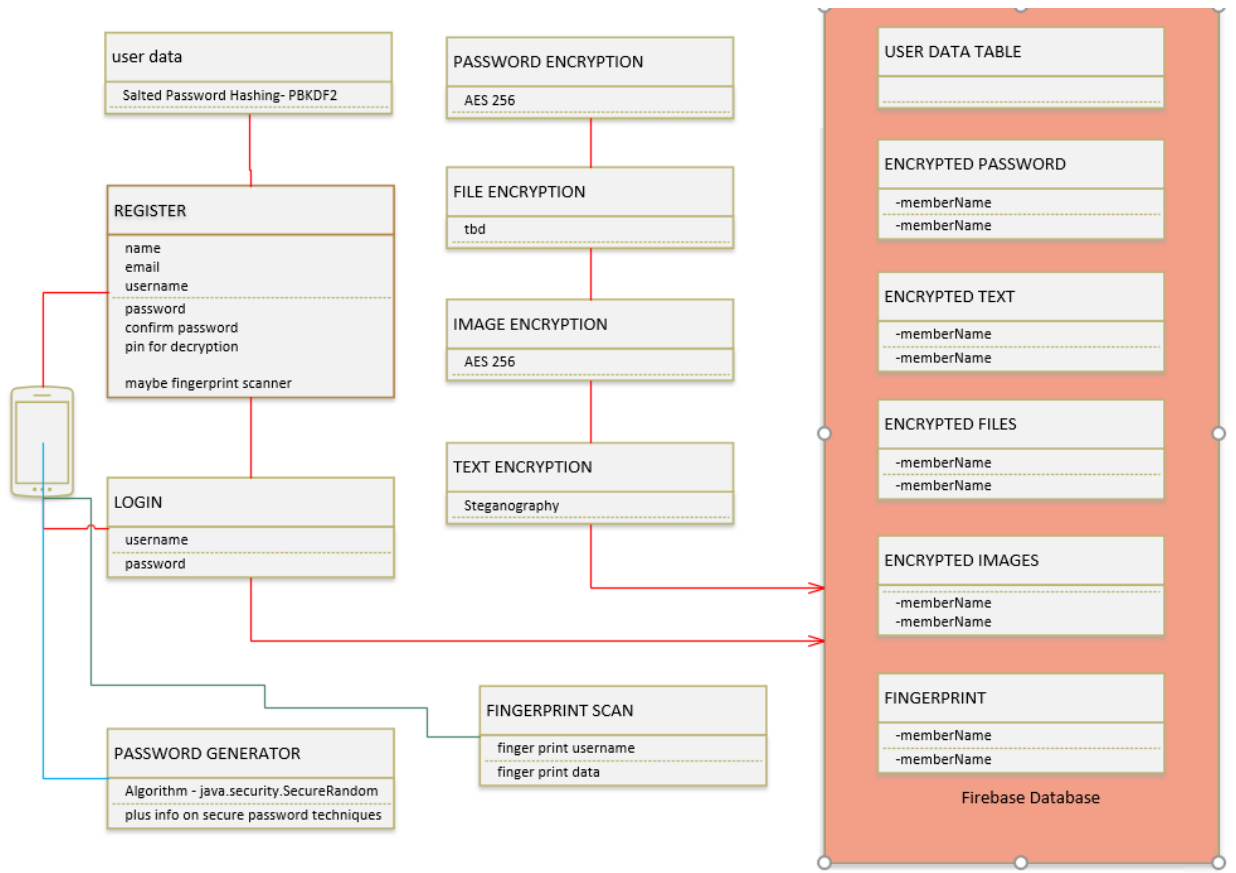
the login and register. The user will be able to encrypt passwords, text blocks, files and images so each of these sections in the application will be encrypted using different algorithms. My android phone also has a fingerprint scanner, so I wanted to include that somehow, so I decided it would be used as a method of decrypting a file the user had previously encrypted.

Although this seemed like a good idea I didn't feel like the complexity was there, so I decided I would make it cross platform by building a web application that registered users could log into to view their encrypted files that way and decrypt them on their PC or laptop. I would use the Google Firebase databases for this as it would allow data to be accessed by both mobile devices and PC's over the internet.

6.1.3 Technical Approach

My approach to this project will be to do as much research as possible on the secure implementations of the encryption algorithms I will use in the application. I will research secure encryption algorithms some of which I have named below as well as secure programming methods.

I will then move onto creating low-fidelity mockups of the android application to help with creating the design of the application. I will create class diagrams and use cases and over time adding relevant and new information to them keeping them up to date, these diagrams will help me keep track of where I am in the project but also help me visualize the functionality of the application. The diagram below is a very basic model of the android application which helps me map out how it will work.



I will create a simple UI spending no more than a day or two on it leaving the proper design of the UI and other features till the end, from there I will implement the functionality of the main features of the application starting with getting the database connected and implementing the login and register features. This will allow me to test the connection to the database as well as my java code. From this stage I will move onto the rest of the application and implement the encryption algorithms leaving the UI design till later in the project lifecycle.

The web application will not be started till later in the lifecycle of this project as I am much better at web design than I am at android development. I will be able to create a well-designed web application in a short period of time leaving more time for the implementation of the backend of the web application. Use cases and class diagrams will also be created for this part of the project as it is separate from the android application.

I will break the development of this project into phases with tasks to completed within these phases. After each phase of the project I will test to make sure that each encryption method or feature is working correctly and to make sure the files, text or image are truly encrypted and secure.

Throughout this project I will be meeting with my project supervisor to review the project as it develops and make any changes to it if my supervisor suggests any changes or improvements. When my supervisor and I feel the project is finished and all goals of the project met the application will be published online.

6.1.4 Special Resources Needed

I currently have good knowledge on building a web application, but android application development is new to me, so I will still need to use resources to enhance my skills in that area such as:

1. Tutorials on encryption algorithms, Java and C#
2. Tutorials on Android backend and UI development.
3. Tutorials on cross-platform applications
4. Google Firebase tutorials.

I will also require the following resources:

1. Laptop and/or PC.
2. Android mobile phone/s.
3. Google Drive to backup project files.
4. Android Java tutorials
5. Web application C# tutorials.
6. Icon creator tool for android application.
7. Android Studio
8. Visual Studio
9. Google Firebase Database Project.

6.1.5 Technical Details

I will be using Android Studio to code the android application, the backend and UI will be created using Android Studio. I will be using Java to code the backend of the application and xml to code the front end of the application. I will be using Visual Studios to create the web application using HTML, CSS and C#. I will be using a Google Firebase Database, Storage and Authentication to save the user's information when they create an account and all encrypted data and files will be stored here.

The android application and web application will both be using and linked to the Firebase database.

Both applications will encrypt files for mobile users and web users, the user can store and encrypt anything they feel needs to be more secure. It will be able to store passwords, text blocks (email addresses, names, bank details or other sensitive information), picture as well as word, PDF and excel files. It will also contain information on how to keep your files and personal information as safe as possible with help sections.

The user will have to register an account and login to access the application, only authenticated users can access the application with the correct credentials and have read and write permissions to the database. The user can encrypt appropriate files but will need to enter a six-digit code to decrypt anything as an extra safety feature or if the mobile phone has a fingerprint scanner, which a lot of phones now have, your fingerprint could be used instead of a pin to decrypt files, but the user will have the choice in case the device doesn't have fingerprint scanner. I believe that fingerprint scanner is good idea to implement as a user cannot forget their fingerprint and no one else can access it with their finger. All the user's information will be encrypted, and all the users encrypted files will be kept on a Google Firebase database. This allows the user to delete the original file on the phone only keeping the encrypted version which can be decrypted whenever needed.

The user interface of the application will be as user friendly as possible, able to navigate the app easily, easy to use the features in the app and look aesthetically pleasing using nice colours and well laid out pages.

I will be using as many different encryption methods for each of the types of files that can be encrypted without compromising security.

Encryption

- Login and Register: Salted Password Hashing
- Password Vault: AES 256
- Text: Steganography
- File: Twofish
- Pictures: TBD

Decryption

- Pin Code
- Fingerprint

Database

- Google Firebase Database

The features of the android application will also be implemented in a web application. The web application's functionality will be very similar to the android application except for the fingerprint scanner.

6.1.6 Evaluation

I will be using a variety of requirement elicitation techniques as a basis for the final requirements specification document. These techniques will most likely consist of surveys, brainstorming, interviews and interface analysis. These are the main techniques I will use but others may be used if they are needed. Implementing these techniques properly will create a great foundation for developing and writing a highly detailed and accurate requirements specification document.

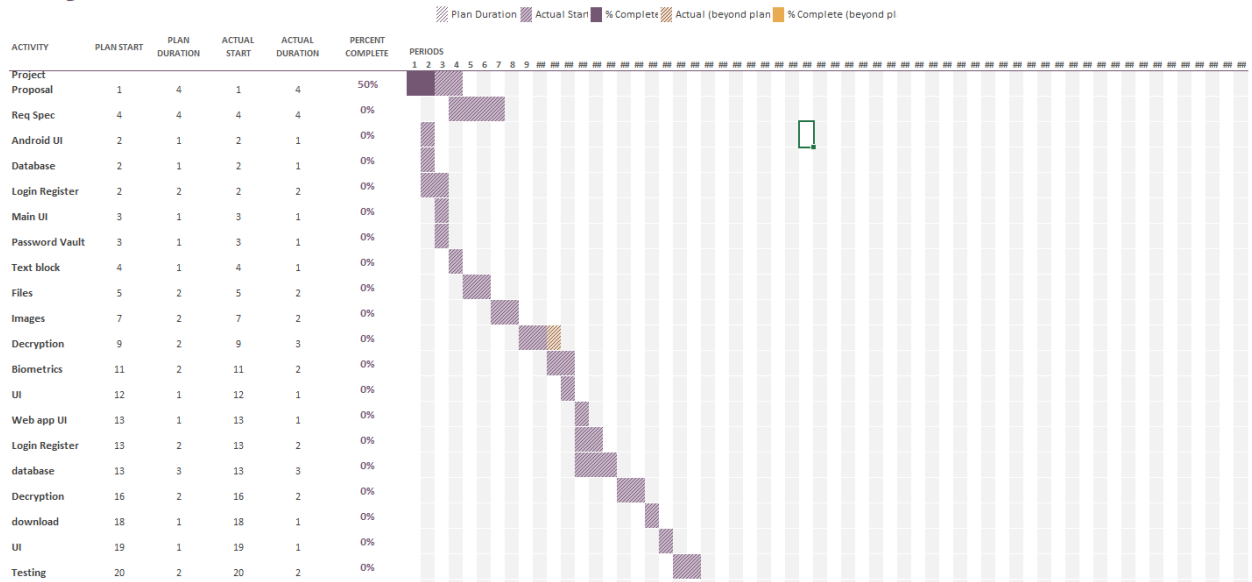
The requirements specification document will be written in conjunction with the development of the android application which will allow me to amend any details in it, this allows it to be completed with the highest attention to detail and accuracy. After the midpoint and prototype presentation information will continue to be added to the document this time based around the web applications, allowing me to make it more accurate and of a higher standard. I will continue to meet with my project supervisor to make sure the project and documentation is on the right track and progressing well. Any feedback from the meeting will be noted and changes made to the project in time. Ideally, I hope my supervisor and myself will be entirely satisfied with the project's progression with the final product when it is completed on the 18th of May 2018.

6.2 Project Plan

As you can see in the Gantt chart below I have given myself enough time to complete each part of the project but also enough time that should I fall behind or become stuck on the implementation of a part I have enough time to push other parts back. I have a plan on how I will move through the project and in which order I implement features which I have simplified below but there is a chance I will

become stuck, so I have allowed myself time to push back other parts of the project.

Project Planner



6.3 Monthly Journals

6.3.1 September 2017

Thursday 21st

Today I actually decided on an idea, an encryption android application. This is basically all I have so I will need to do my research and see what I can do to make mine stand apart from the others out there or cool features I can add to it.

Monday 25th

So, done some research and found that most apps like mine on the Google Play Store are garbage or look horrendous so that give warrant to my idea as I'll make

my app look great. Also, I have decided to implement 4 different things you can encrypt, passwords, text blocks, files and images. Also, a login and register feature of course.

When files are encrypted they are saved to a Firebase database.

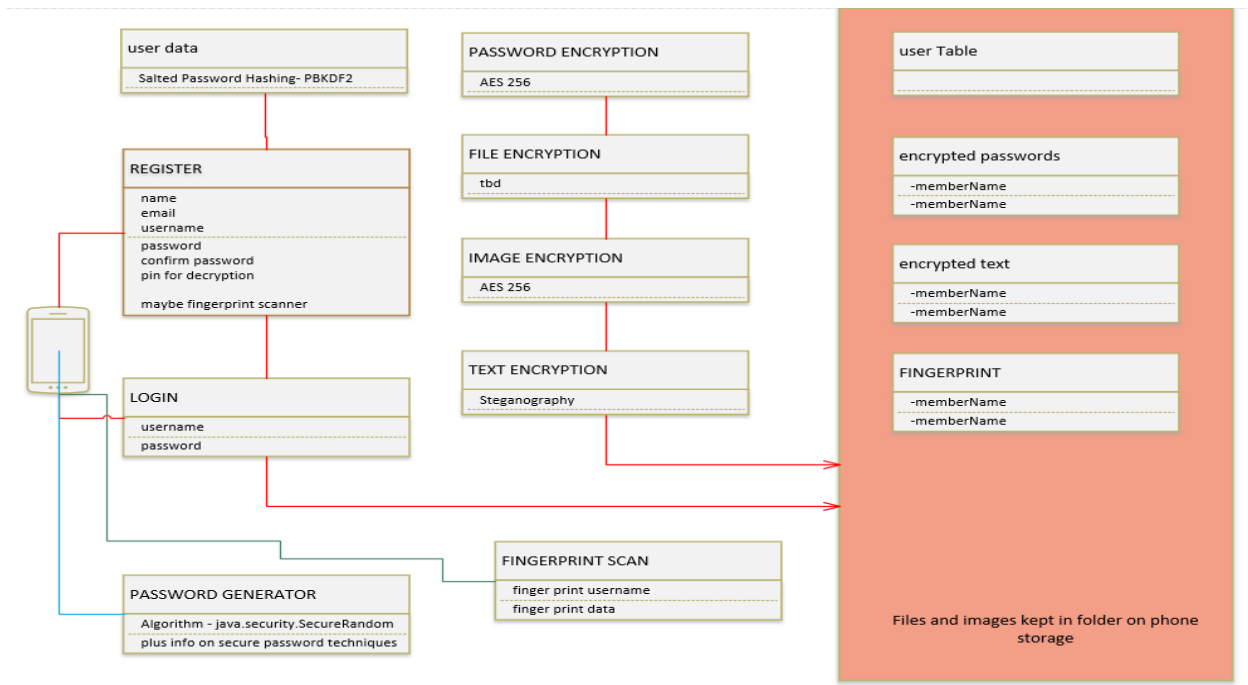
Other apps on the play store focus on one type of thing to encrypt while mine will encrypt a variety of things.

Wednesday 27th

So, I've gotten the general idea of what the app will do in my head, so I used Visio to make a basic model of the android app just to help me visualize what I need to do a little better.

Also, when decrypting files so you can see them again you have to enter a pin code which you enter when you encrypt it, my phone has a fingerprint scanner and so do a lot of newer phones. So, my girlfriend said why don't you use the fingerprint to authorize the decryption of the files? So that's what I am going to do. Phones without fingerprint scanners can use a pin code, phones with them can decrypt files with their thumb. I think this is a good idea as it's not on any of the apps I've researched, and you generally can't forget your finger were as you can forget a pin code.

I have also decided to use different encryption algorithms and methods for each thing that can be encrypted so that should be fun.



Saturday 30th

So today I have finalized my idea for the android application, all its features and functionality.

I've also decided that if I was to complete the application in a shorter time frame I would attempt to make it cross platform, so create a web application in Visual Studios that is the same as the android application and has the same functionality. The android application is the priority and will be completed where the web application could still be a prototype by the end of it all.

6.3.2 October 2017

Monday 2nd

Today I had my project pitch at 10.10 am. I felt prepared and hoped I wouldn't freeze up or get too nervous as I usually do with presentations, it also didn't help that I have the flu but whatever.

I explained my idea and got asked lots of questions but were very happy I was including different encryption algorithms in the application, I explained I wanted to showcase that I could implement different encryption types, that I was using the fingerprint scanner and that it was cross-platform, I explained the web application may end up being in its prototype stages. So, I got the go ahead on the project with a few recommendations. So now I know my idea is good and have the go ahead I can finally start project proposal and begin designing my application.

Tuesday 3rd

Today I have gotten a start on my project proposal which will describe the project, what it will do, how I will do it, what I will need to do it and all that stuff.

Thursday 5th

The project proposal is mostly done just need to add some finishing touches. I also started my research based around Google Firebase by reading through the documentation for it and looking at tutorials. I also created an android application to test the code and see if it works and basically learn by trial and error.

Sunday 8th

So, I learned that you can add Firebase through Android Studio easily and it adds all the gradle dependencies and files you need to use Firebase. I also managed to figure out Google Authentication and create a simple login and register which worked, I plan on adding google and Facebook login options as well.

Tuesday 10th

I got a simple feature working where the user, only if logged in and authenticated in Firebase, can add a password, not encrypted, and the name of the password, like Facebook or whatever they want, to the Firebase database. Most of the code is handled by Firebase API but I had done a lot of research to find out to do it properly.

Thursday 12th

I broke the entire app, so I had to delete the project on Google and make a new android project. It's probably for the best as the first one was a mess of random code, this one will be everything I learned and the code I wrote to make the app properly.

The login and register using email and password authentication now works fully. A simple home screen Ui was made and I made the classes and UI for adding passwords and text blocks to the database.

Sunday 15th

So, for Firebase Database is a real-time database best used for just strings or ints, but Firebase Storage is mainly for files, images, videos and audio which is perfect for my app as well. So, implemented adding an image to the Firebase Storage which didn't take too long as it was similar to the other parts. I found to allow the user to access their images the image is stored in storage and a reference to the image saved in the database.

Also found a nice colour scheme I want to use throughout the application.

Monday 16th

Made more changes to the code in the application such as form validation and capturing the Firebase User ID when uploading data to the database.

I also got the passwords to output in a simple list view, but it shows all the passwords in the database, which I don't want, but it's a start.

Tuesday 17th

The data the user upload to the database is now sorted by the user's firebase ID, this means the users data is all stored together, where before all user's data was stored together and there was no structure to the database. This means the users can now only see their own data when login which is a big step and makes outputting the data for the users to see a lot easier as they will be grouped by their ID's. I hope that makes sense.

I also implemented a splash screen and created the icon and logo for the application using a free online icon creator and I am going to call it Kryptium.

Friday 20th

I started my research into the encryption algorithms mainly AES 256 for the moment.

So far, I have login and register using Firebase Auth. The user can add passwords, text, images and files to the database, I used a Firebase List Adapter to show the users data as a list on each activity.

Sunday 22nd

I began implementing the AES 256 algorithm on the passwords section of the android app.

I also began creating the pages which allow the user to delete or decrypt passwords, this is done by making the list view clickable and when the user clicks a password name the intent grabs the data from the list view and passes it to the new activity where the information is displayed. The user can enter the pin code to delete or decrypt the password.

Wednesday 25th

Following various online resources, I managed, after many attempts and coffee breaks, to get the password to encrypt using and AES/CBC 256 algorithm. I decided to put the algorithm up on stack overflow to ask if the algorithm had any problems or if it could be improved.

Sunday 29th

Today I got the password to delete if the right pin code is entered, my only concern is the way that I got it too work isn't secure, and the pin code is not encrypted so it may be a temporary solution.

My next thing is to get the data, from when a user creates a new password, is pushed to the database to be encrypted and get the password decrypt and delete functions working properly.

6.3.3 November 2017

Friday 3rd

I began some research into android fingerprint manager and found its simple to use as there are loads of tutorials on it, I tried to implement it in my delete a password feature in the app and it didn't work and I just ended up more confused.

So, I added the Gmail login instead, so the user can now login with their Gmail account or their email and password.

Began my research on steganography.

Monday 6th

I found a good tutorial on the fingerprint scanner and how to implement the way I need to use it, so I will be implementing that over the next week or so.

I also have a few CA's and uploads due this week, so I don't think I'll be getting much done this week at all.

Friday 10th

I have implemented the fingerprint scanner functionality on the password decrypt section, it took me two days to get working as there is a lot of code to use. But now when the user goes to the decrypt page they see a pin code or fingerprint button. If they select pin code the pin code edit-text shows but if they press fingerprint a dialog popup appears, and they are asked to scan their finger, if it is recognized the user can delete or decrypt the data. If it's not an error message shows, and they can try again

Monday 13th

I have exams and other projects starting to pile up, so I only got to change the UI of some of the pages in the application and done some more research on steganography. Long four weeks ahead.

Friday 17th

I've been working on the requirement specification the last two weeks, more so this week doing a bit every day. I have the bulk of the content in it already just have to clean it up and make sure it all makes sense and so on.

I also met with Irina as I had a few questions about the system architecture, my use cases and the functional requirements. Most of the questions were just concerns I had, making sure I was doing it right. I just have to change some of the functional requirements and I should be fine.

Sunday 19th

Today I implemented the fingerprint scanner functionality in all the other sections of the application. Took a while as the code as it needs two classes to work for each section.

Tuesday 21st

I found a AES library on GitHub and decided to try use it just to see what happens because the one I currently have doesn't work correctly, to my surprise it worked and allows you to create a new AES builder, so I am able to code the AES how I want it to function, number of iterations, key length etc. This a big step as AES is what I will be using to encrypt the passwords as well as the user's details.

Friday 24th

Submitted the requirement spec and now to do the technical report in a week.

6.3.4 January 2018

December was a bit of a messy month as most of the CAs and projects were due as well as various presentations on various things, so I didn't get much done on

the project. The start of January was the same as I had my exams up until the 13th. I took a break for Christmas and new year's mostly and just chilled out until I started studying again.

Wednesday 17th

Today I created the web application in Visual Studio and created the pages I will need, or think I'll need, as well as add a few files and code segments needed for it to work with my Firebase Project.

Thursday 18th

Today I tried implementing the login functionality on the web app using the google documents and their GitHub documents as well.

It worked but also didn't at the same time so.

Saturday 20th

The way firebase is configured to operate on a web application is different than a standard login page, it creates the session and you have to set the page to change its UI rather than bringing you to another page. This was confirmed on their GitHub page, that's why it wasn't working so the login and register part of the web application works now.

Monday 22nd

The last few days I've spent getting the basics of the web application working. When the user logs in they can go to their home page and select what they want to see or add. I have the password data downloading and displaying in a list. Now I need to find a way to decrypt an aes ciphertext in C#.

Tuesday 23rd

One of the problems I was having was how to keep the pin code safe because I was saving it as plaintext in the database, so I am hashing it then using the hashed pin code as the key in the AES algorithm. I started using SHA but its not really secure enough so I changed it to PBKDF2 as it is slower making brute forcing much more tedious to do. Now I had to make sure that the hash created in java

was the same created in C#. After some research and finding some good tutorials I was able to create the same hash on both platforms.

Thursday 25th

The next problem was creating the same C# AES cipher in java and C#. I found a great GitHub library which had the aes algorithm in java and C#. After implementing which took a while...it didn't work.

Saturday 27th

Tried to get the AES to be cross platform again today..... didn't work.

Sunday 28th

I am sick of AES, so I did more research on steganography which is when I found a GitHub library which had to code I need to finish what I already had. Before I was following very disjointed tutorials on what I need to make this work but could never get it to work. Also, there actually isn't that much information on it.

I spend the day implementing what I needed from the GitHub and changing it to do what I wanted it to do.

I had to change it because I couldn't upload the bitmap as bytes to the database, I had to use a firebase function to upload the bitmap as bytes to firebase storage instead, the get the download URL and store that in the database instead.

Wednesday 31st

I am a idiot...the AES was not working because I had the variables in the wrong order in the C# algorithm so it was trying to get the key from the IV instead of the other way around.....it works now.

Now I can encrypt a password on the android app and decrypt it on the web application.

6.3.5 February 2018

Friday 2nd

After more trying with the Steganography i got it work, instead of using an image I used the method from the GitHub which creates a 200*200 bitmap of random colour, convert the bitmap to bytes, get the text which the user wants to hide and convert that to bytes, then use the steg method to combine them. The new bitmap created from this had the text in it and it is also converted to bytes.....these bytes are uploaded to storage. I spend the rest of the day doing the same method but in reverse for the getting the text back out. It all works. Thank you to the GitHub user who created this which helped me finish this part of my project.

Wednesday 7th

I spend the last few days just doing some more work on the web application such as UI stuff and getting some more of the session features working. The module projects are starting to be released so I will be focusing on them for a while.

Wednesday 14th

I finished the overall UI design of the web application such as the theme and colours I will use. I also, using JavaScript, managed to get the login and register working on the web application.

Wednesday 21st

I created the user sections of the web application and using the google documentation got the user data displaying in a table on the webpage. I only have this working on the passwords section, but the code can be reused in the other parts by just changing the path of the database.

Friday 23rd

Spend the day redesigning the UI for the android application, I didn't like the old one but the new one looks much cleaner and has better colour coordination.

6.3.6 March 2018

Saturday 3rd

Spend the last few days implementing AES in C# on the web application, if successful it should decrypt the passwords which were encrypted on the android app. It didn't work.

Sunday 4th

I'm an idiot I had the variables the wrong way in the decrypt method, it works now.

Wednesday 7th

I have the image encryption working, encrypting and decrypting, using 3DES. It took a while to do simply because of all the conversions that needed to happen during the encryption and decryption phases.

Sunday 11th

I have started doing the Twofish encryption implementation on the android app and I have added a few more features to the web application.

Thursday 15th

Twofish is now working on the android application, the implementation is very close to the AES one, so I was able to do it much quicker. I also did a lot of research prior to coding it so i had a lot of resources to use which I had found over the last few weeks.

Saturday 17th

Twofish now works fully and is encrypting and decrypting files.

Wednesday 21st

I spent the last few days doing the document and catching up on what needs to be in the document such as documenting the encryption algorithms now working in the applications.

Tuesday 27th

I have begun testing the android application, creating unit tests, security tests and UI tests.

The security scans revealed a few issues which I will try to fix, and the first UI test failed because of the splash screen but after removing it passed the Robo UI test. I put the splash screen back in because I like it.

Saturday 31st

After spending two days on this I managed to get the web application to encrypt and decrypt the images using C# triple DES. It doesn't work fully and i don't think it will but for the most part it does.

So now the application uses all four encryption algorithms and the website uses two of them, which I am happy with.

The UI of the android app is done, and website still needs some work.

6.3.7 April 2018

Saturday 7th

I haven't done much the last week because of the other projects but I have started writing the rest of the unit tests for the android app.

6.3.8 May 2018

Tuesday 1st

For the past month I just done bits and bobs on both the android application and web application mainly because of the module CAs and then the end of year

exams. I then took a few days of to just chill out but today I spent going over everything that still needs to be done and surprisingly it's a small list.

I spend the day doing vulnerability scans of the web application and fixed any issues I could and wrote about it in the report so.

I also planned on what needs to be done on the website itself but overall, I am happy with it as it is kind of a beta website, a proof of concept for a cross platform application.

Thursday 10th

Making the final touched to the technical report.