Mobile Phone Self Encryption

By

Yawar Khan

Hamza Yousaf

Ali Anwar Siddiqui

Submitted to the Faculty of Computer Software Engineering

National University of Sciences and Technology, Islamabad in partial fulfillment

For the requirements of a B.E. Degree in Computer Software Engineering

JUNE 2018

ABSTRACT

Mobile Phone Self Encryption

This Application is aimed to provide a platform for the end user to store data in encrypted form on an Android device and also provide backup of data.

This project allows users to store sensitive data on their mobile phones without having to worry about its confidentiality even if the mobile phone is lost. This system is developed so that employees and other mobile users can store and operate on sensitive data on their mobile phones without having to worry of it being leaked. This software project concentrates on securing data on mobile phones by storing it in an encrypted form. This data is encrypted with a stream cipher whose key is stored on a trusted server. When the mobile device is lost, it sends a report to the server and the server then destroys the respective key so that the data on the mobile can never be decrypted and remains confidential.

CERTIFICATE FOR CORRECTNESS AND APPROVAL

Certified that work contained in the thesis – Mobile Phone Self Encryption carried out by Yawar Khan, Hamza Yousaf and Ali Anwar under supervision of A/P Mr. Mian Muhammad Waseem Iqbalfor partial fulfilment of Degree of Bachelor of Software Engineering is correct and approved.

Approved by

A/P Mr. Mian Muhammad Waseem Iqbal

CSE DEPARTMENT MCS

DATED:

_____

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose support and cooperation,
a work of this magnitude would not have been possible

To our supervisor A/P Mr. Mian Muhammad Waseem Iqbal who
has given us great support and valuable suggestions throughout
the implementation process.

And finally, to our Friends and siblings for their encouragement.

ACKNOWLEDGEMENTS

There is no success without the will of ALLAH Almighty. We are grateful to ALLAH, who has given us guidance, strength and enabled us to accomplish this task. Whatever we have achieved, we owe it to Him, in totality. We are also grateful to our parents and family and well-wishers for their admirable support and their critical reviews. We would like to thank our supervisor. A/P Mr. Mian Muhammad Waseem Iqbal, for his continuous guidance and motivation throughout the course of our project. Without their help we would have not been able to accomplish anything.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Overview

This poject allows users to store sensitive data on their mobile phones without having to worry about its confidentiality even if the mobile phone is lost. This system is developed so that employees and other mobile users can store and operate on sensitive data on their mobile phones without having to worry of it being leaked. This software project concentrates on securing data on mobile phones by storing it in an encrypted form. This data is encrypted with a stream cipher whose key is stored on a trusted server. When the mobile device is lost, it sends a report to the server and the server then destroys the respective key so that the data on the mobile can never be decrypted and remains confidential.

1.2 Problem Statement

Smart phone usage is increasing day by day in every sector. Its growing so fast. The world is becoming more digital. They do make life of everyone easier. With this increase in usage, the vulnerabilities also increase with time. Smart phones are more prone to attacks, which is becoming a real problem for smart phone users now-a-days. So, what should we do? Stop smart phone usage? No, that doesn't sound like a solution. We can provide more security, than there is today, for smart phones to overcome this problem.

1.3 Approach

We are working with data that is selected by users that can be Images, Videos, Audio, Documents, Messages and Contacts, respectively. Selected data will go through stream cipher and will be encrypted. It will be saved in encrypted form and will be removed/hidden from its location. Also, this data will be saved on server in encrypted form to provide backup incase mobile phone is lost. We aim to provide an application that can provide confidentiality and integrity.

1.4 Scope

This project will assist users who are most concerned with their mobile phone's security and want their data to remain confidential. It will be done by securing the data of the mobile phones in an encrypted form. This data is encrypted with a stream cipher whose key is stored on a trusted server. When the mobile device is lost, the user sends a report to the server simply by logging into his account to set the status of phone as "Lost" and the server then destroys the respective key so that the data on that mobile phone can never be decrypted and remains confidential. In this way, user does not have to worry about the confidential data even if the mobile phone is lost.

1.5 Objectives

The Objectives of this project are following:

- All mobile data will be in encrypted form. User doesn't have to worry about data theft.
- Data Remains Secured: - Data stored in database is in encrypted format, so malicious user would not able to understand the data even if hacks the database.
- Data Retrieval: - User can retrieve his confidential data by logging in to the system
- Access from anywhere: - User can access the data even if his mobile phone is lost anywhere at any time.

1.6 Deliverables

Table 1

Sr.	Tasks	Deliverables
1	Literature Review	Literature Survey
2	Requirements Gathering	SRS Document
3	Application Design	Design Document (SDS)
4	Implementation	Implementation on computer with a live test to show the accuracy and ability of the project
5	Testing	Evaluation plan and test document
6	Deployment	Complete application along with

1.7 Overview of the document

This document shows the working of our application Mobile Phone Self Encryption. It starts off with the system architecture which highlights the modules of the software and represents the

system in the form of component diagram, Use Case Diagram, Sequence Diagram and general design of the system. Then we move on to discuss the detailed Description of all the components involved. Further we discuss the dependencies of the system and its relationship with other products and the capacity of it to be reused.

1.8 Purpose of the document:

This document aims to elaborate the idea and design of the project that is Mobile Phone Self Encryption. This document will highlight all the specifications of our project i.e. how it will be used, what will be the scenario in which the project will be useful. This document will help in defining functional and non-functional requirements.

Chapter 2: Literature Review

There was a lot research work done and also different projects were made that were based on the idea of Stream cipher-based encryption following is a detailed description of research work and projects previously carried out in this context.

- **AMIT BANERJEE, MAHAMUDUL HASAN, MD. AUHIDUR RAHMAN, AND RAJESH CHAPAGAIN**

Department of Computer Science, South Asian University, New Delhi 110021, India Corresponding author: Mahamudul Hasan (m.hasan@students.sau.ac.in)

CLOAK: A Stream Cipher Based Encryption Protocol for Mobile Cloud Computing

IEEE Access.

Received July 26, 2017, accepted August 4, 2017, date of publication August 25, 2017, date of current version September 27, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2744670

- **Andreas Klein**

Dept. of Pure Mathem. & Computer Algebra State University of Ghent, Ghent, Belgium

Stream Ciphers

Springer London Heidelberg New York Dordrecht

ISBN 978-1-4471-5079-4(eBook)

Library of Congress Control Number: 2013936538

Mathematics Subject Classification: 94A60, 68P25, 11T71

- **Software Projects**
  Following are some project based on encryption:
  - Text Secure Private Messenger (Free)
  - Red Phone: Private Calls (Free)
  - Crypt4All Lite (AES) - (Free File Encryption Software)
  - Orbot: Proxy with Tor (Free Privacy and Online Anonymity)
  - AppLock (Free Application Lock Utility)
  - BitVise XTS-AES Based Disk Encryption Software

Conclusion

Keeping our private data secure, it means only user can access and use the data. The purpose of this project is to allow users to store sensitive data on their mobile phones without having to worry about its confidentiality even if the mobile phone is lost. This system is developed so that employees and other mobile users can store and operate on sensitive data on their mobile phones without having to worry of it being leaked. Security is the main purpose of the project and security provided using encryption scheme.

Chapter 3: Software Requirement Specification

3.1Introduction

The purpose of this part is to describe the project titled “Mobile Phone Self Encryption”. This part contains the functional and non-functional requirements of the project. It contains the guidelines for developers and examiners of the project.

This purpose of this project is to allow users to store sensitive data on their mobile phones without having to worry about its confidentiality even if the mobile phone is lost. This system is

developed so that employees and other mobile users can store and operate on sensitive data on their mobile phones without having to worry of it being leaked.

3.2 Overall Description

3.2.1 Product Perspective

Data needs to be secured just like any of your other business assets, like your property, your stock/merchandise, your hardware, etc. Depending on the sort of data you're storing, there may well be security and privacy regulations to follow, particularly when it comes to personal data. If the personal data gets in hand of a hacker he could black mail the person using that person. In business the financial data being in wrong hands could lead to huge loses to the company and much more. This shows how valuable the data is and therefore there is a need for some mechanism to secure this valuable data. Mobile phone self-encryption using stream cipher encrypts all of the data and even when the phone is lost the hacker cannot get hold of the data as the data is in encrypted form and the key is on the trusted server. We can also delete the key from the server upon mobile phone loss so that there is no way left to recover the data on the lost phone.

3.2.2 Product Functions

The mobile phone self-encryption have the following functionalities:

- We will also provide a website where user can login.
- The user can lock the phone from the website and sync data.
- The data will be stored in a database in encrypted form.
- The data on the mobile phone and the server will be synced every time the data is modified, deleted or added.

3.2.3 User Classes and Characteristics

Following are the user classes and their brief description.

- **Company Employees**

Employees that are working in a company are having very important data of the company in the form of emails which needs protection.

- **Government Officials**

  Government Employees have secret Intel in their phones which needs protection from hackers.

- **General Public**

  Securing personal data like photos, contacts, emails etc. is everyone's need nowadays.

3.2.4 Operating Environment

The operating environment required for this project is:

Android Studio

Software requirements:

OS: Android Operating System

3.2.5 Design and Implementation Constraints

Following are the constraints of design and implementation in our project

- Mobile Phone Self-encryption is based on client-server architecture that is Mobile phone act as client and Firebase act as server. They are connected through the internet.
- The user must decrypt all the files before decrypting the key.
- If the user changes the key he cannot retrieve the data on server because it's with old key so it cannot be decrypted.

3.2.6 User Documentation

- A user manual will be provided which will help new users to get started with the Mobile Phone self-encryption. The user manual will provide the instructions on how to work with this Mobile Phone self-Encryption.
- A summary will also be provided to the user which will highlight the features and limitations of this language

3.2.7 Assumptions and Dependencies

- The mobile phone needs to have internet access to be able to get synced with the server.
- The amount of data to be encrypted depends on the storage space at the server.
- The user has to login in to the server to delete data on his phone when it gets lost.

3.3 External Interface Requirements

3.3.1 User Interfaces

The System comprises of an android based application, using java and XML, which shall provide a graphical user interface for user friendly environment. The user would be asked for input i.e. select images etc. that need to be processed.

The user interface for the android application of the System, shall be compatible to all android devise but for best user experience the following versions are preferable

- Jelly beans 4.1.2
- ICS 4.1.1

User Interface:

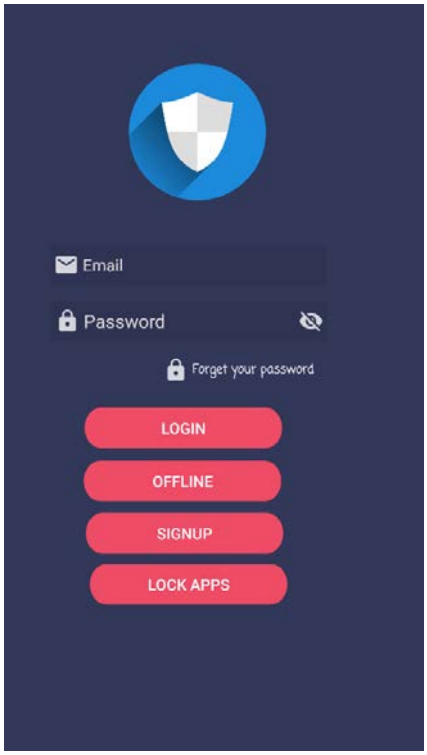

Figure 3.3.1 Interfaces

3.3.2 Software Interfaces

- Android App will be installed on android device with android version Jelly beans 4.1.1 or later.
- The android app would be built using Android Studio.
- For this system, following API and external libraries will be used:
    - Google APIs
    - Android APIs

3.4 System Features

SF-1 Decrypt

Description	The user would be allowed to decrypt the selected file
Priority	High
Pre-Condition	User has an account and the application is running

Stimulus/Response	File is decrypted
Post-Conditions	Selected file is decrypted, saved and encrypted file is deleted
Risk	Medium

Functional Requirements:

REQ-1 App shall decrypt the file and save it in original form.

SF-2 Encrypt

Description	The user would be allowed to encrypt selected/desired data
Priority	High
Normal Course	User add a file and file is encrypted.
Pre-Condition	User has files on the device to be encrypted and is logged in
Stimulus/Response	Data would be encrypted
Post-Conditions	Selected file is encrypted, saved and original file is deleted
Risk	High

Functional Requirements:

REQ-2 App shall encrypt the data, save and delete original file.

SF-3 Synchronization

Description	The user would be allowed to upload data on server for backup
Priority	High
Pre-Condition	User has a working internet connection
Normal Course	All user data is synchronized with the server
Post-Condition	The data on phone and server is identical
Alternate Course	Error occurred if user connection to the internet is interrupted
Post-Condition	The data on phone and server is identical
Risk	Medium

Functional Requirements:

REQ-3 Android phone must be connected with the internet.

SF-4 Update Status

Description	The user would be allowed to upload data on server for backup
Priority	High
Normal course	User sets the status of his/her mobile phone
Pre-condition	User has an account and is logged in from browser
Post-condition	The status of the mobile phone is updated
Risk	Low

3.5 Other Nonfunctional Requirements

3.5.1 Performance Requirements

- The system shall not crash accidently even if a program fails to execute.
- The response time of the system shall be less than 100ms

3.5.2 Security Requirements

- The password of the user to login into the server will contain letters A-Z and a-z and alphanumeric and must be minimum 8 characters.

3.5.4 Software Quality Attributes

Usability

The application will be easy to operate for any user. The graphical user interface will be designed, organized and presented in a manner that is both visually appealing and easy to use.

Accuracy

To ensure accuracy and correctness of file, the file processing algorithms will be written with no tolerance for error.

Availability

The application will available to the user until the phone is in working state and the application is installed and configured properly.

Flexibility

The design and architecture of the application will be flexible enough for catering any new requirements, if any at some later stage or for the application enhancement.

Data Integrity

If the application crashes during any phase, the input images will stay safe in phone's memory/SD card.

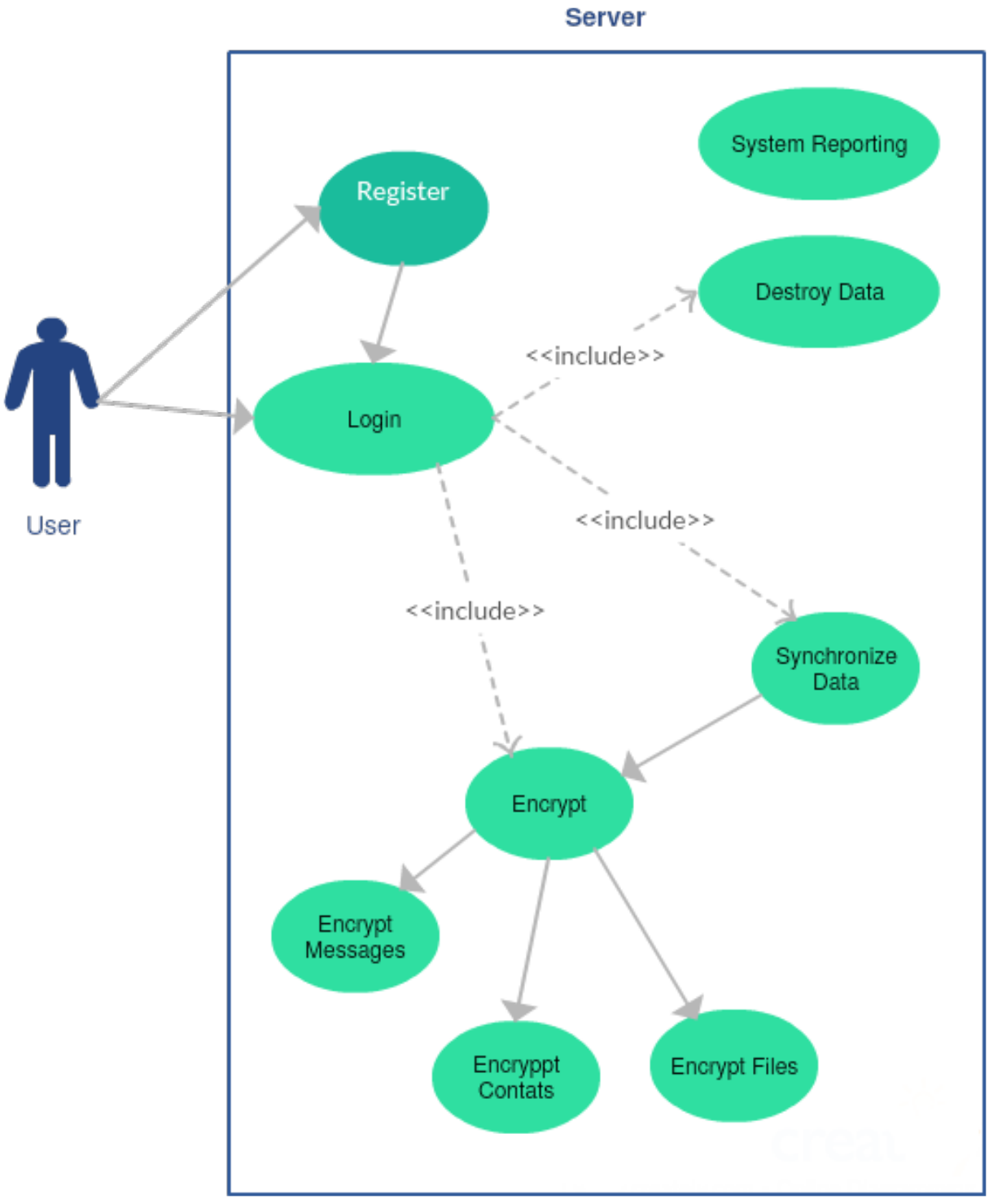
Server
System Reporting
Register
Destroy Data
<<include>>
Login
User
<<include>>
<<include>>
Synchronize Data
Encrypt
Encrypt Messages
Encryppt Contats
Encrypt Files

***Figure** 3.5*

Chapter 4: Design and Development

4.1 INTRODUCTION:

This document includes software design for Mobile Phone Self-Encryption. It specifies the detailed architectural design of Mobile Phone Self-Encryption which is being developed. It will act as a guideline for developers and all the other stakeholders throughout the development. Document include classes and their inter-relationships, use cases with detailed descriptions, sequence diagrams, activity diagrams and various others.

This document is intended for developers, testers, users, documentation writers, project clients, project supervisor and project evaluators. A copy of this document will be made available to all stakeholders.

4.2 Scope of the Development Project

This project will assist users who are most concerned with their mobile phone's security and want their data to remain confidential. It will be done by securing the data of the mobile phones in an encrypted form. This data is encrypted with a stream cipher whose key is stored on a trusted server. When the mobile device is lost, the user sends a report to the server simply by logging into his account to set the status of phone as "Lost" and the server then destroys the respective key so that the data on that mobile phone can never be decrypted and remains confidential. In this way, user does not have to worry about the confidential data even if the mobile phone is lost.

A stream cipher is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (key stream). In a stream cipher, each plaintext digit is encrypted one at a time with the corresponding digit of the key stream, to give a digit of the

cipher text stream. Since encryption of each digit is dependent on the current state of the cipher. In practice, a digit is typically a bit and the combining operation an exclusive-or (XOR).

4.3 System Architecture Description

This Section overview of application, its higher and lower levels details and user interfaces.

4.3.1OVERVIEW OF MODULES/COMPONENTS

Here we will give brief overview of all the modules.

4.3.1.1 Add Module:

This module allows users to encrypt data on their phones.

4.3.1.2 Remove Module:

This module allows users to decrypt data on their phones.

4.3.1.3 Sync Module:

This module allows users to sync files with server for backup.

4.3.1.4 Settings Module:

This module allows users to set key for encryption/decryption and set time for synchronization.

4.3.1.5 Login Module:

This module allows users to login to their respective accounts that are registered on the server.

4.3.1.6 Registration Module:

This Module allows users to register themselves on the server and create an account.

4.3.2 Structure and Relationships

This section covers the overall technical description of Mobile Phone Self Encryption. It shows the working of application in perspective of different point-of-views and also shows relationships between different components.

Use Case Diagram

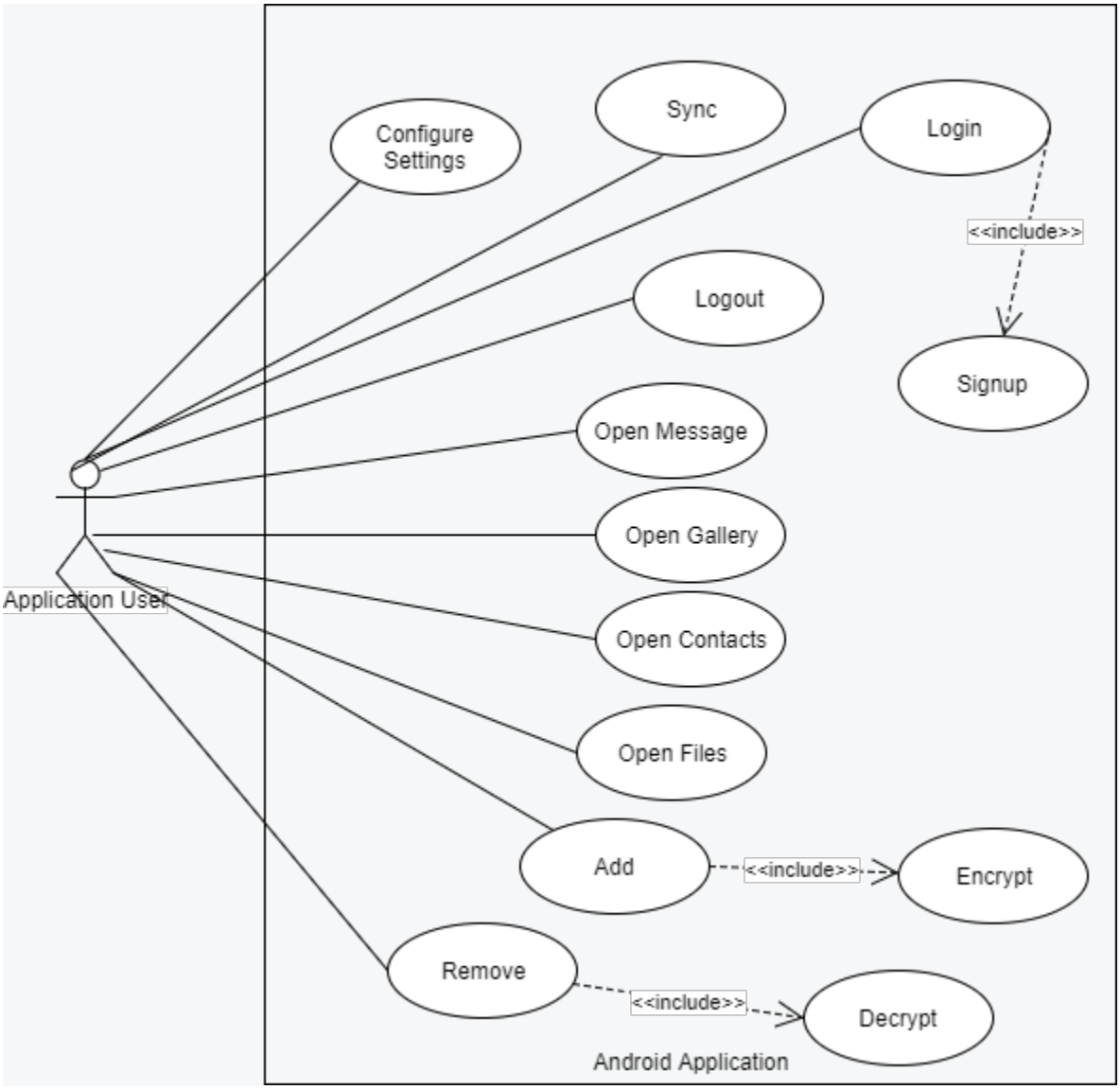

Fig. 4.3.2.1:*Use Case*

Actors

Primary Actor(s): Application User *Secondary Actor(s):* Server

Use Cases

1. Login
2. Signup

3. Open Gallery
4. Open Messages
5. Open Files
6. Open Contacts
7. Add
8. Remove
9. Encrypt
10. Decrypt
11. Logout
12. Sync
13. Configure Settings
14. Update Status

Use Case Description

Use Case 1: Login

Use case name	Login
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User logs in successfully
Alternate course	Error occurred due to wrong username or wrong password
Pre-condition	User has an account and the application is running
Post-condition	Menu is displayed to the user
Extend	N/A
Include	Signup
Assumptions	User has a working internet connection

Use Case 2: Sign Up

Use case name	Sign Up
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User creates account successfully
Alternate course	Error occurred due to invalid username or username already taken or password length less than 8 characters
Pre-condition	User has the app installed and running
Post-condition	User is created.
Extend	N/A
Include	N/A
Assumptions	User has a working internet connection

Use Case 3: Add

Use case name	Add
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User adds a file and the file is encrypted
Alternate course	Error occurred if the file format is not supported or user is out of memory
Pre-condition	User has files on the device to be encrypted and is logged in
Post-condition	Selected file is encrypted, saved and original file is deleted
Extend	N/A

Include	Encrypt
Assumptions	User has selected a valid file and has enough memory to save encrypted file

Use Case 4: Remove

Use case name	Remove
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User removes the files and the file is decrypted
Alternate course	Error occurred if the user is out of memory
Pre-condition	User has an account and the application is running
Post-condition	Selected file is decrypted, saved and encrypted file is deleted
Extend	N/A
Include	Decrypt
Assumptions	User has enough memory to save decrypted file

Use Case 5: Sync

Use case name	Sync
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	All user data is synchronized with the server
Alternate course	Error occurred if user connection to the internet is interrupted
Pre-condition	User has a working internet connection

Post-condition	The data on phone and server is identical
Extend	N/A
Include	N/A
Assumptions	User has an account and a working internet connection, and the application is running

Use Case 6: Configure Settings

Use case name	Configure Settings
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	The key is configured, sync time is set
Alternate course	Error occurred if invalid key entered (invalid length)
Pre-condition	User is logged in
Post-condition	Key is set and sync time is set
Extend	N/A
Include	N/A
Assumptions	User has the app running

Use Case 7: Open Messages

Use case name	Open Messages
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User clicks “messages” button and message activity is displayed

Alternate course	N/A
Pre-condition	User has an account and the application is running
Post-condition	Message activity shown
Extend	N/A
Include	N/A
Assumptions	User has an account and the application is running

Use Case 8: Open Gallery

Use case name	Open Gallery
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User clicks “gallery” button and gallery activity is displayed
Alternate course	N/A
Pre-condition	User has an account and the application is running
Post-condition	Gallery activity shown
Extend	N/A
Include	N/A
Assumptions	User has an account and the application is running

Use Case 9: Open Contacts

Use case name	Open Contacts
Primary actor	Deploying Person
Secondary actor	N/A

Normal course	User clicks “contacts” button and contact activity is displayed
Alternate course	N/A
Pre-condition	User has an account and the application is running
Post-condition	Contact activity shown
Extend	N/A
Include	N/A
Assumptions	User has an account and the application is running

Use Case 10: Open Files

Use case name	Open Files
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User removes the files and the file is decrypted
Alternate course	N/A
Pre-condition	User has an account and the application is running
Post-condition	Selected file is decrypted, saved and encrypted file is deleted
Extend	N/A
Include	N/A
Assumptions	User has an account and the application is running

Use Case 11: Encrypt

Use case name	Encrypt
Primary actor	Deploying Person

Secondary actor	N/A
Normal course	The file is encrypted successfully
Alternate course	N/A
Pre-condition	User has an account and the application is running, user has files to encrypt and has free memory
Post-condition	Selected file is encrypted, saved and original file is deleted
Extend	N/A
Include	N/A
Assumptions	User has an account and the application is running, user has soe free memory

Use Case 12: Decrypt

Use case name	Decrypt
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	The file is decrypted successfully
Alternate course	Error occurred if the user is out of memory
Pre-condition	User has an account and the application is running, user has files to decrypt and has free memory
Post-condition	Selected file is decrypted, saved and encrypted file is deleted
Extend	N/A
Include	N/A
Assumptions	User has an account and the application is running, user has soe free memory

Use Case 13: Log Out

Use case name	Log Out
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User logged out successfully
Alternate course	N/A
Pre-condition	User has an account and the application is running, and user is logged in
Post-condition	Logged out and log in screen shown
Extend	N/A
Include	N/A
Assumptions	User has an account and the application is running

Use Case 14: Update Status

Use case name	Update Status
Primary actor	Deploying Person
Secondary actor	N/A
Normal course	User sets the status of his/her mobile phone
Alternate course	N/A
Pre-condition	User has an account and is logged in from browser
Post-condition	The status of the mobile phone is updated

Extend	N/A
Include	N/A
Assumptions	User has a working internet connection

Sequence Diagrams

Following sequence diagrams show the sequence of activities performed in all use cases described above.

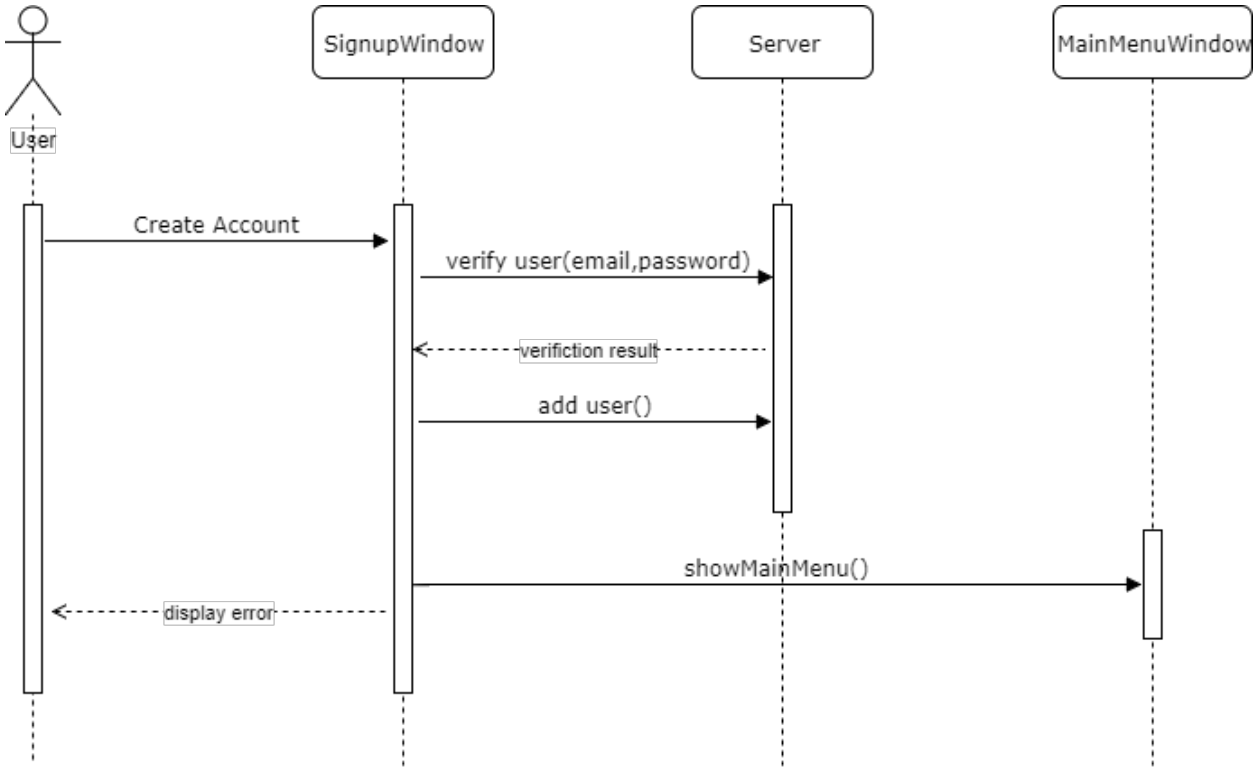

Figure 5– Sign up Sequence Diagram

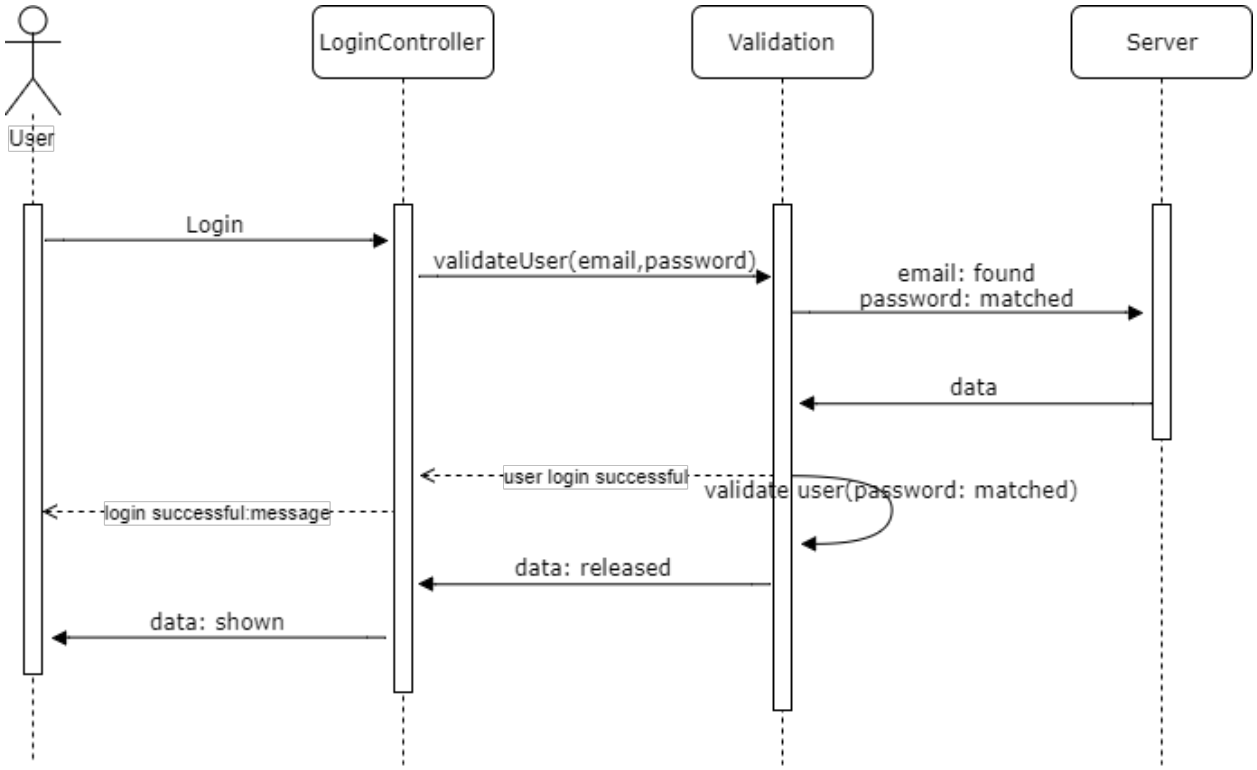

Figure 6 – Login Sequence Diagram

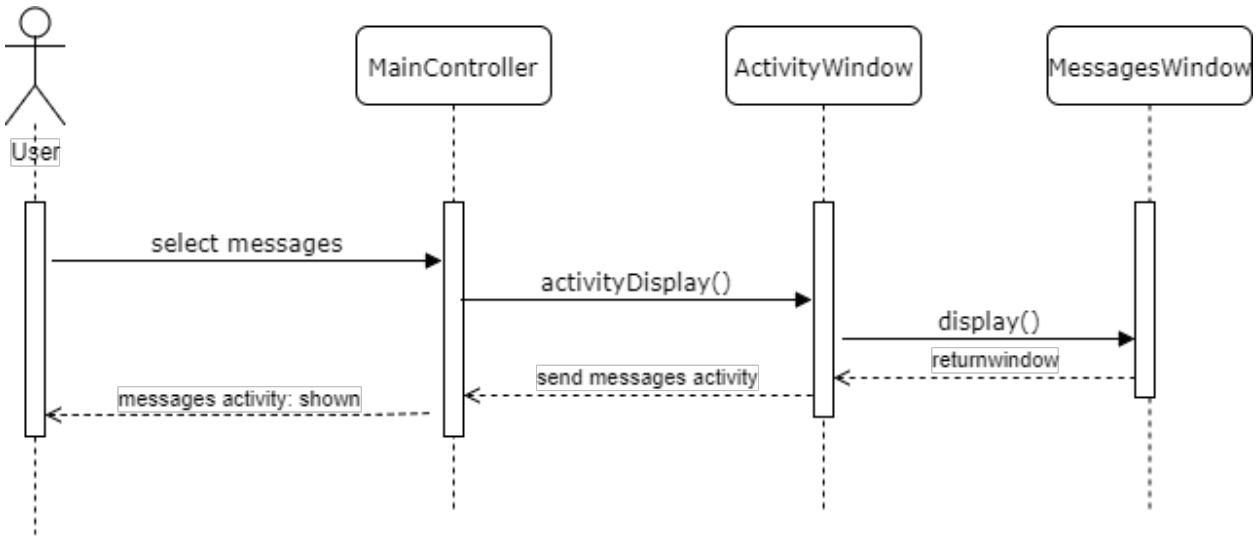

Figure 7 – Messages Sequence Diagram

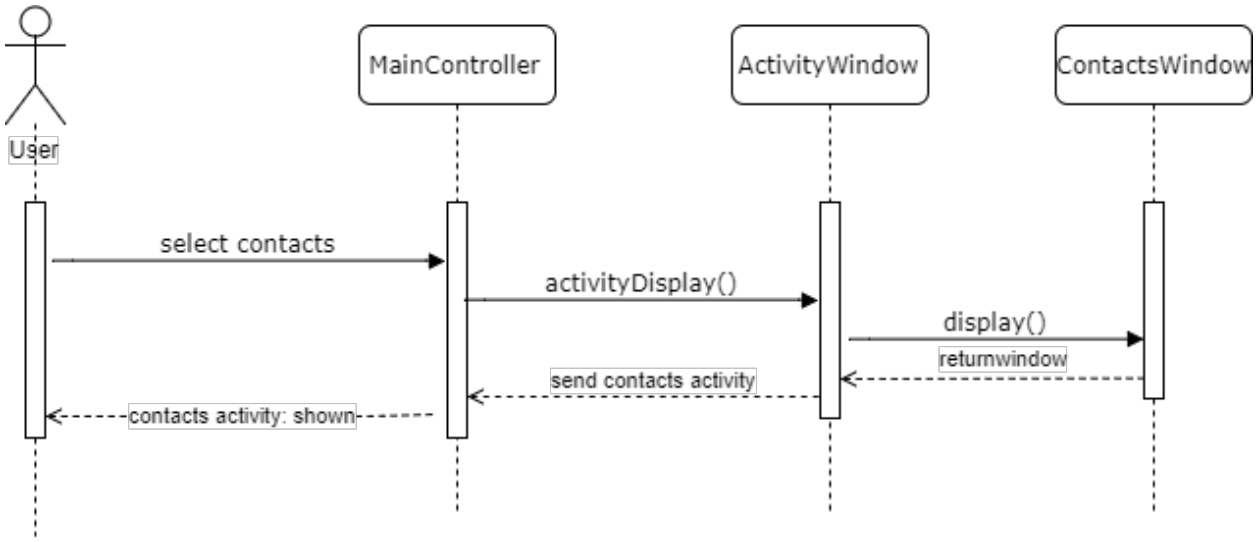

Figure 8 – Contacts Sequence Diagram

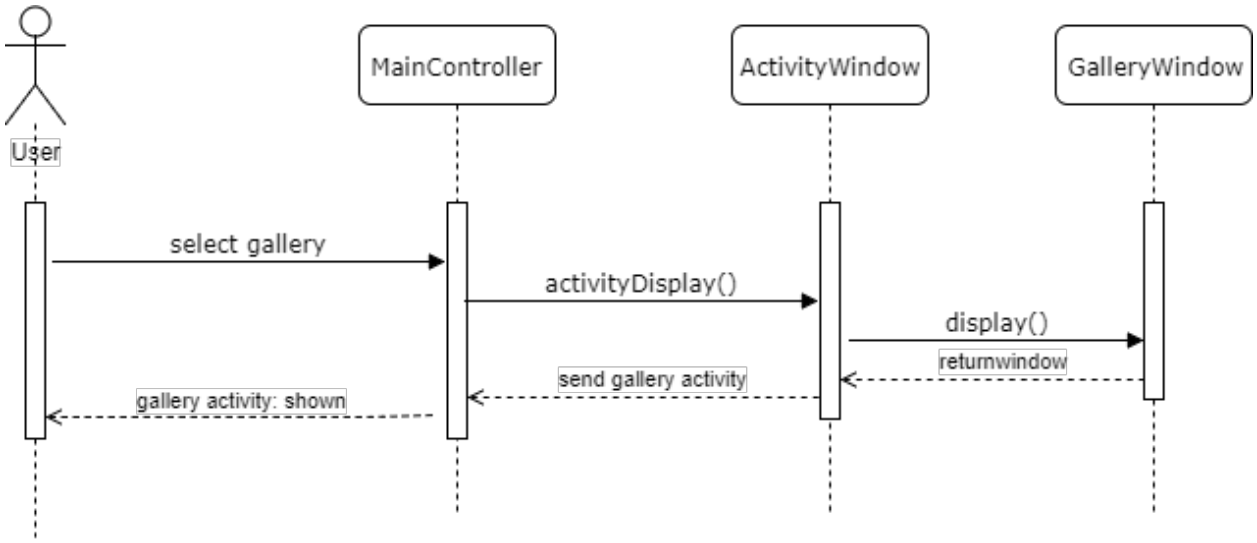

Figure 9 – Gallery Sequence Diagram

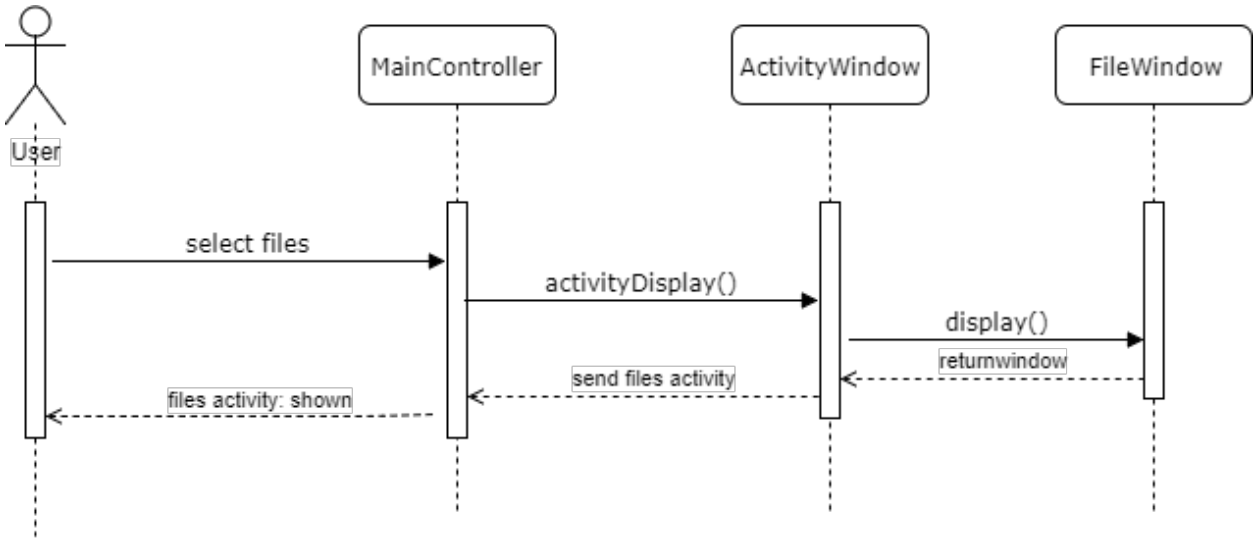

Figure 10 – Files Sequence Diagram

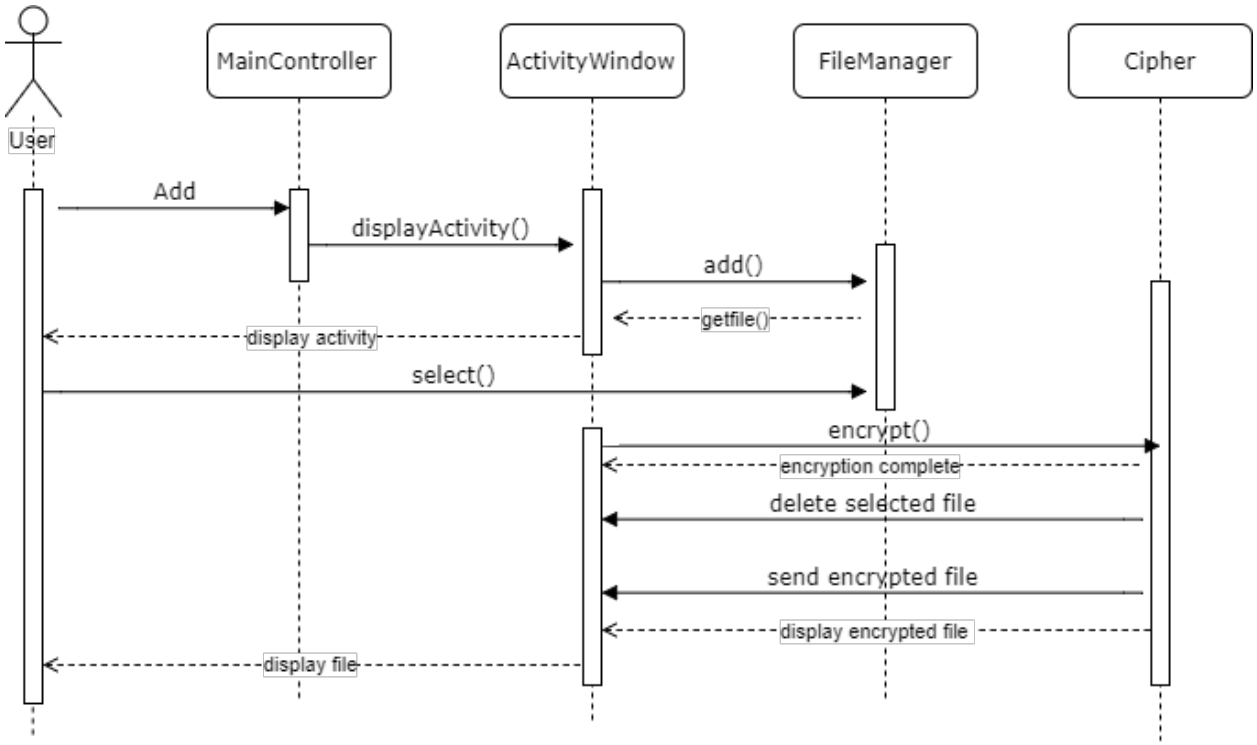

Figure 11 – Add Sequence Diagram

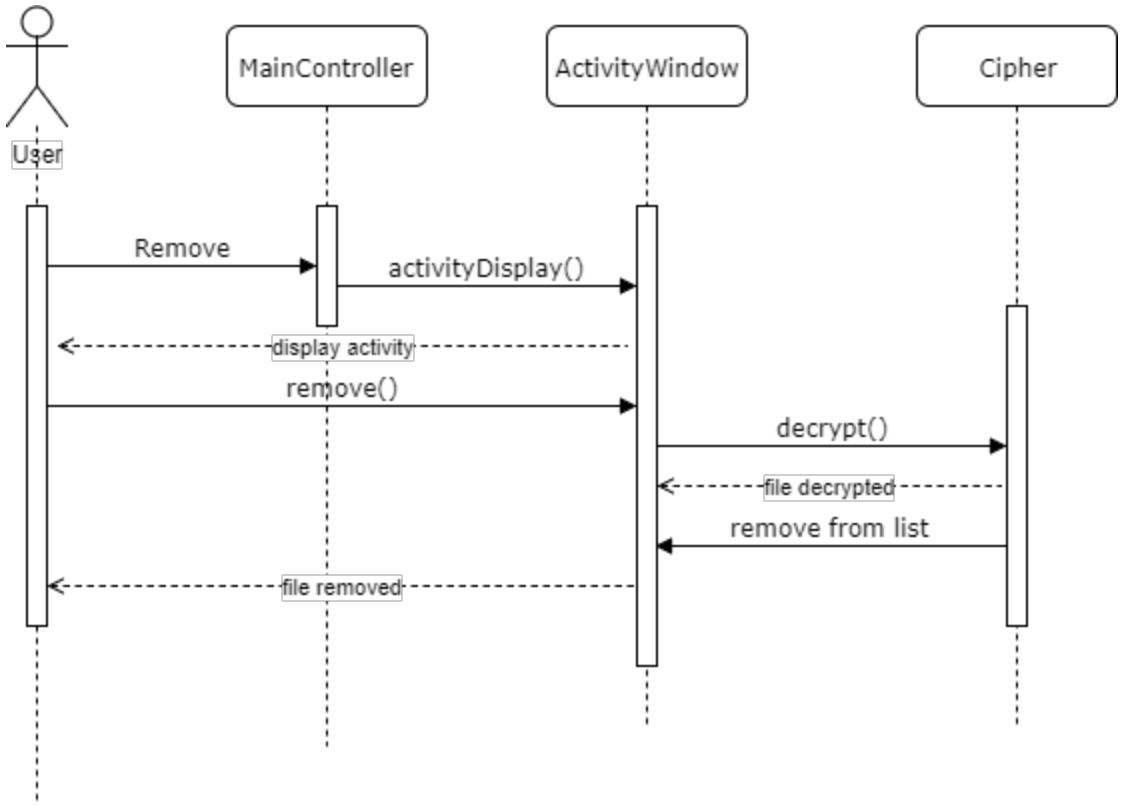

Figure 12 – Remove Sequence Diagram

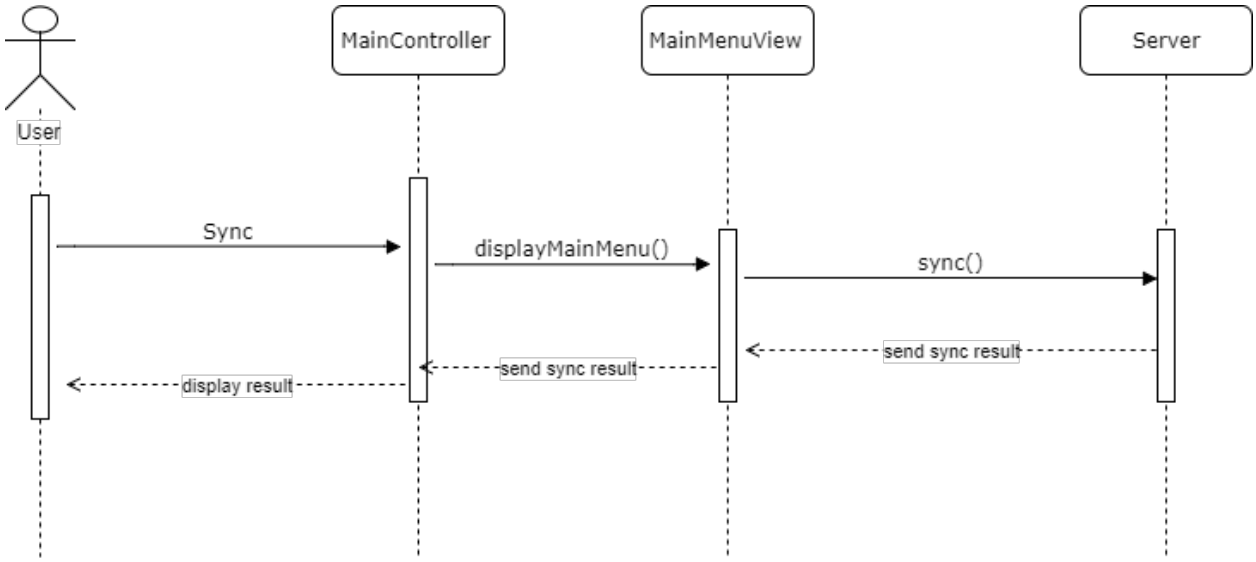

Figure 13 – Sync Sequence Diagram

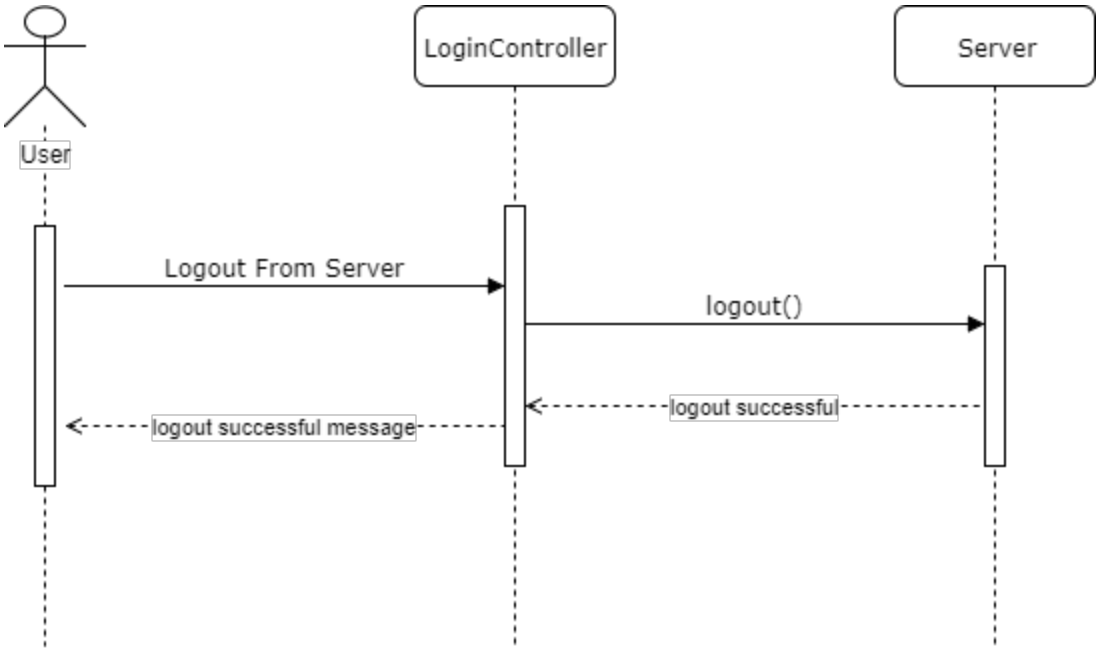

Figure 14 – Logout Sequence Diagram

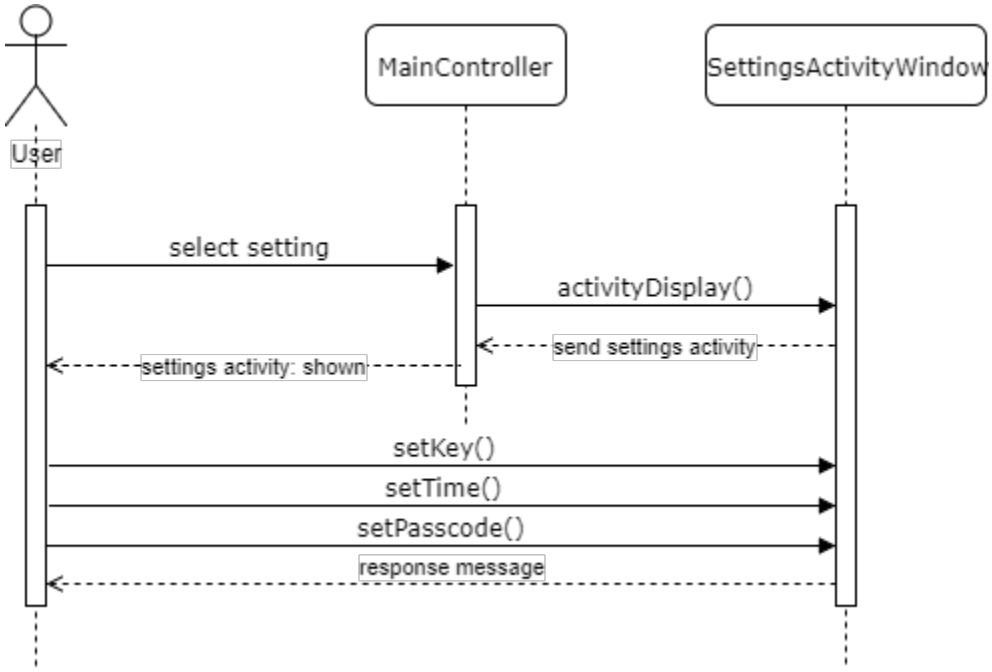

Figure 15 – Settings Sequence Diagram

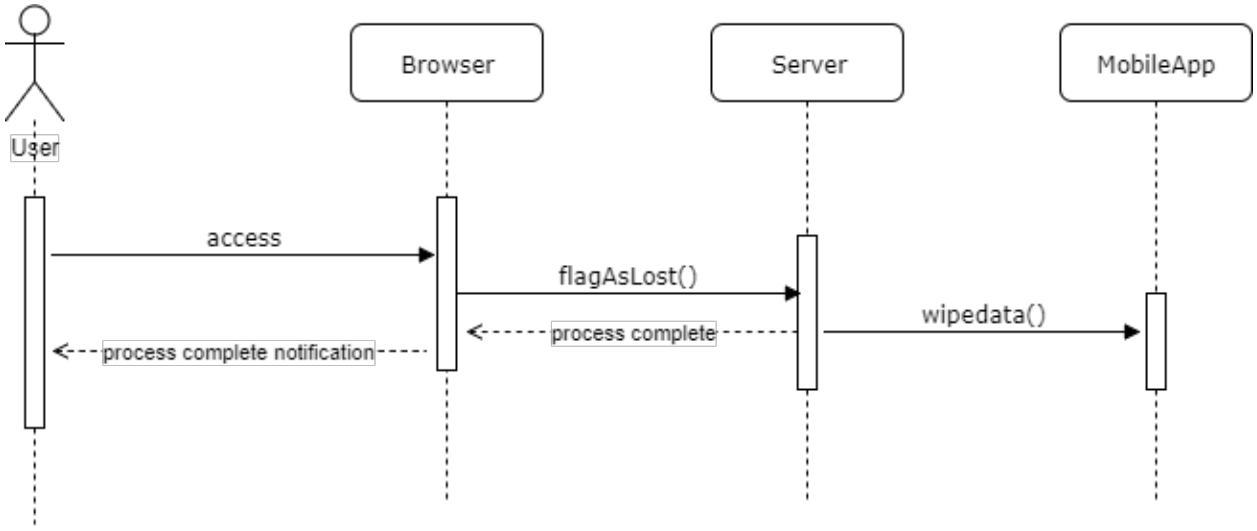

Figure 16 - Update Status Sequence Diagram

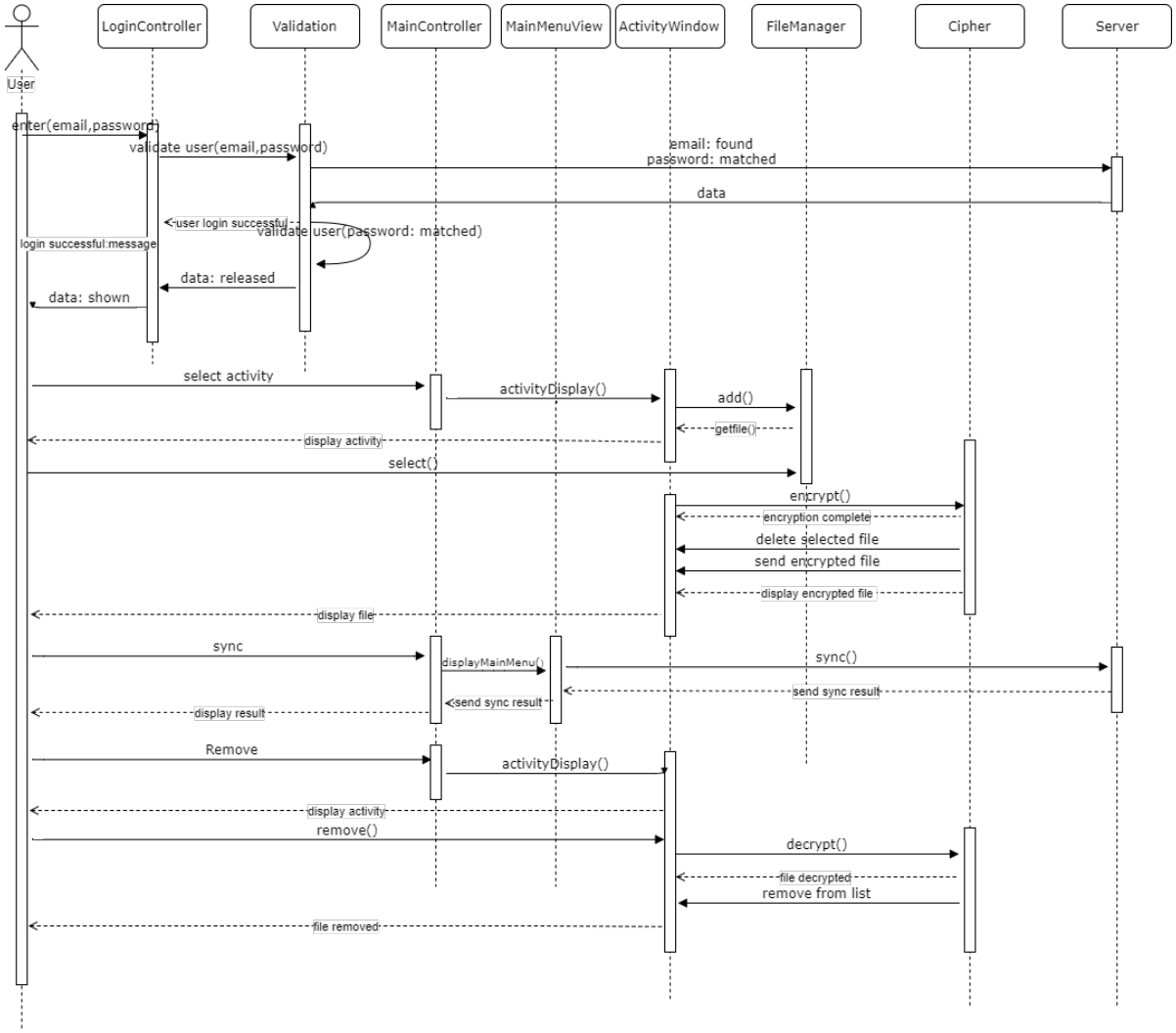

Figure 17 – Overall System Sequence Diagram

Class	Description
MainMenuView	This is main menu viewclass of the System. It creates the interface for the main menu which includes Messages Button, Contacts Button, Gallery Button, Sync Button, Files Button, Settings Button and Logout Button and Method sync()
MainController	It is the main Controller class. It constructs all the necessary elements for the application to run It invokes the events by making function calls to different methods FileManager, ActivityMenu and MainMenuView
ActivityWindow	This is the class for handling all activities displayed to user. It has four sub classes i.e. MessagesActivityWindow, FilesActivityWindow, GalleryActivityWindow, ContactsActivityWindow
SettingsActivityWindow	This class enables the user to configure settings i.e. Enter key for encryption, set passcode for the application to start and also specify which times can the application synchronize data with the server
Cipher	This class is responsible to generate key stream using a standard algorithm which is then used to encrypt files. It needs the key entered by the user to generate key stream
LoginController	This class is responsible for managing login activity and how the user is verified

LoginWindow	This class has all the necessary elements to make the interface for login window
SignupController	This class is used to create user accounts by creating instances of 'ApplicationUser' class. The data is sent to the server and saved in the database
SignupWindow	This class has all the necessary elements to make the interface for signup window
MessagesActivityWindow	This is a subclass of the class "ActivityWindow" and it handles messages activity. It is used to display messages view so that the user can add messages to it and secure it
GalleryAvtivityWindow	This is a subclass of the class "ActivityWindow" and it handles gallery activity. It is used to display gallery view so that the user can add images and videos to it and secure it
ContactsActivityWindow	This is a subclass of the class "ActivityWindow" and it handles contacts activity. It is used to display comtacts view so that the user can add contacts to it and secure it
FilesActivityWindow	This is a subclass of the class "ActivityWindow" and it handles files activity. It is used to display files view so that the user can add files to it and secure it
FileManager	This class handles all the activities which are concerned with getting the data (files) from mobile phone.
Validation	This class is used to validate users' accounts by checking the username and password combination
ApplicationUser	This class contains all the information about users that are registered i.e. username, password.

Logical View (State Transition Diagram)

The State Transitions occurring in the application are shown in **Fig. 4.3.2.8** below:

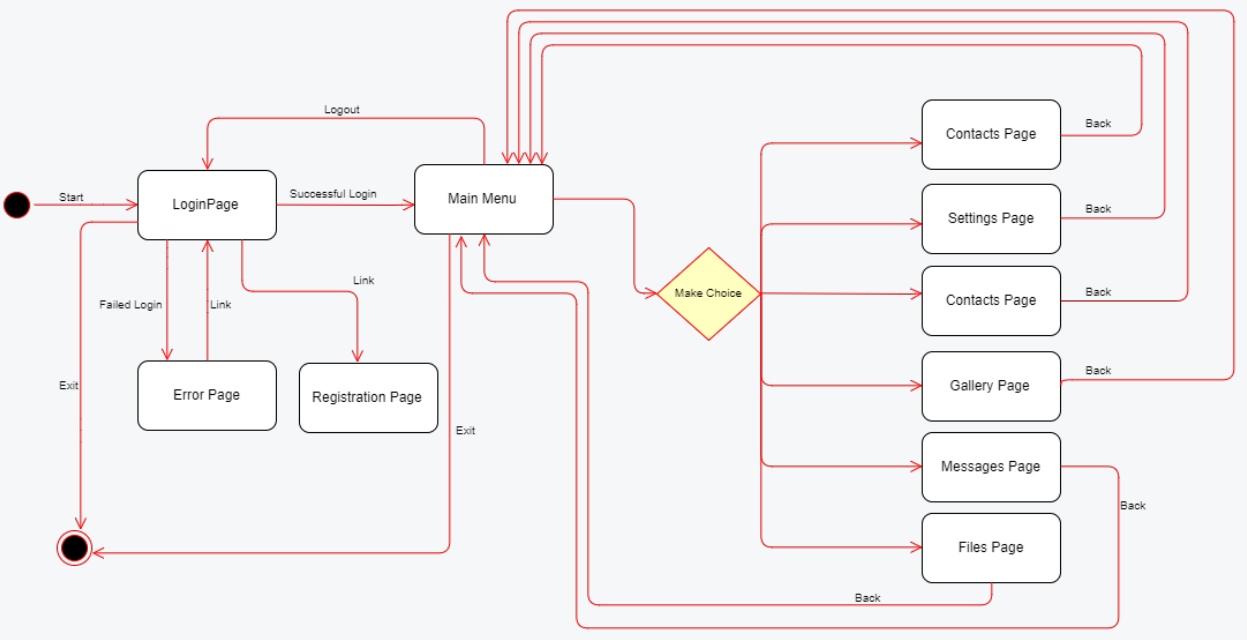

Fig. 4.3.2.8:*State diagram for Mobile Phone Self Encryption*

Dynamic view (Activity Diagram) In activity diagram, the dynamic view of the system is shown. All the activities are shown concurrently with their respective start and end states.

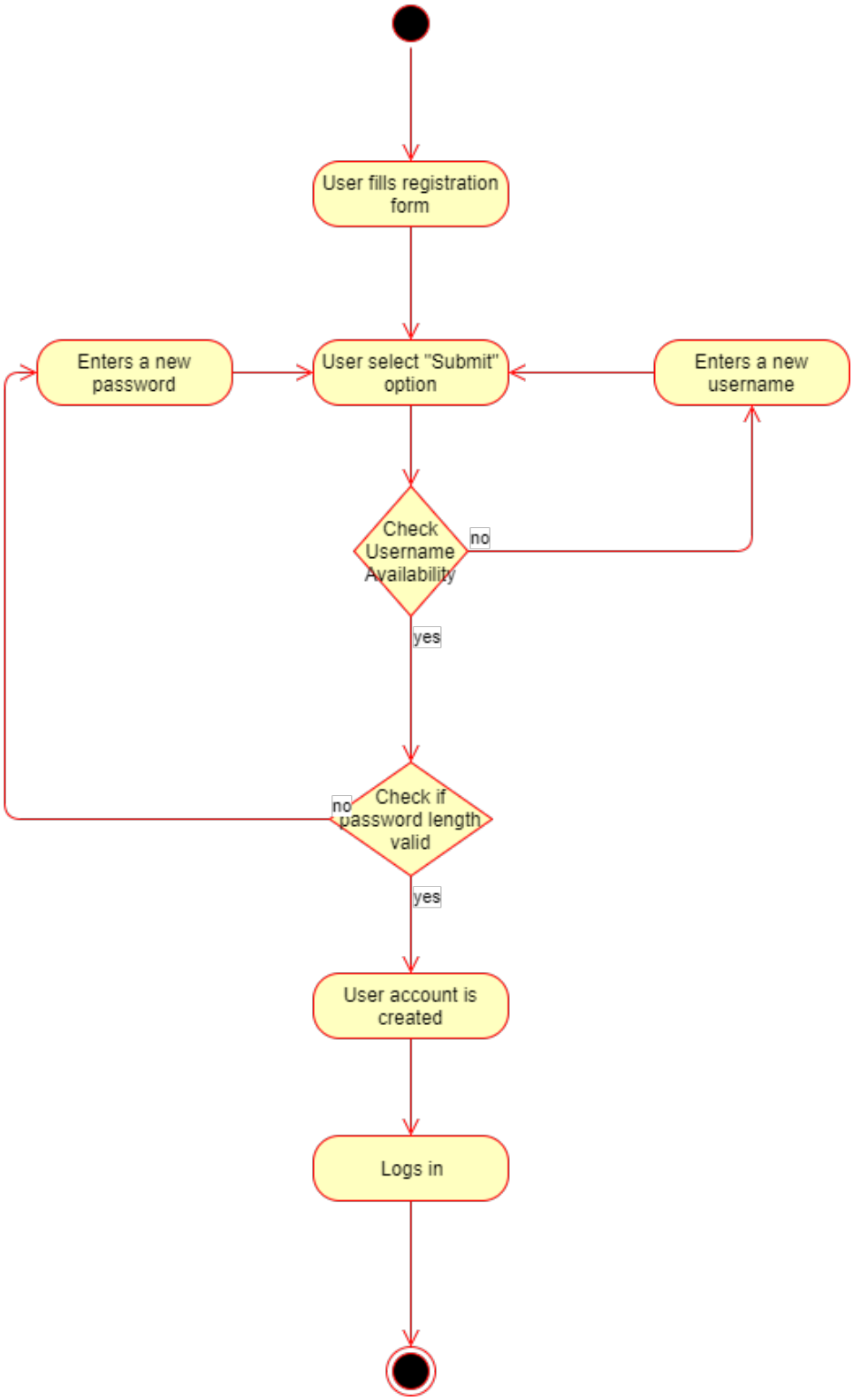

Figure 18 – Signup Activity

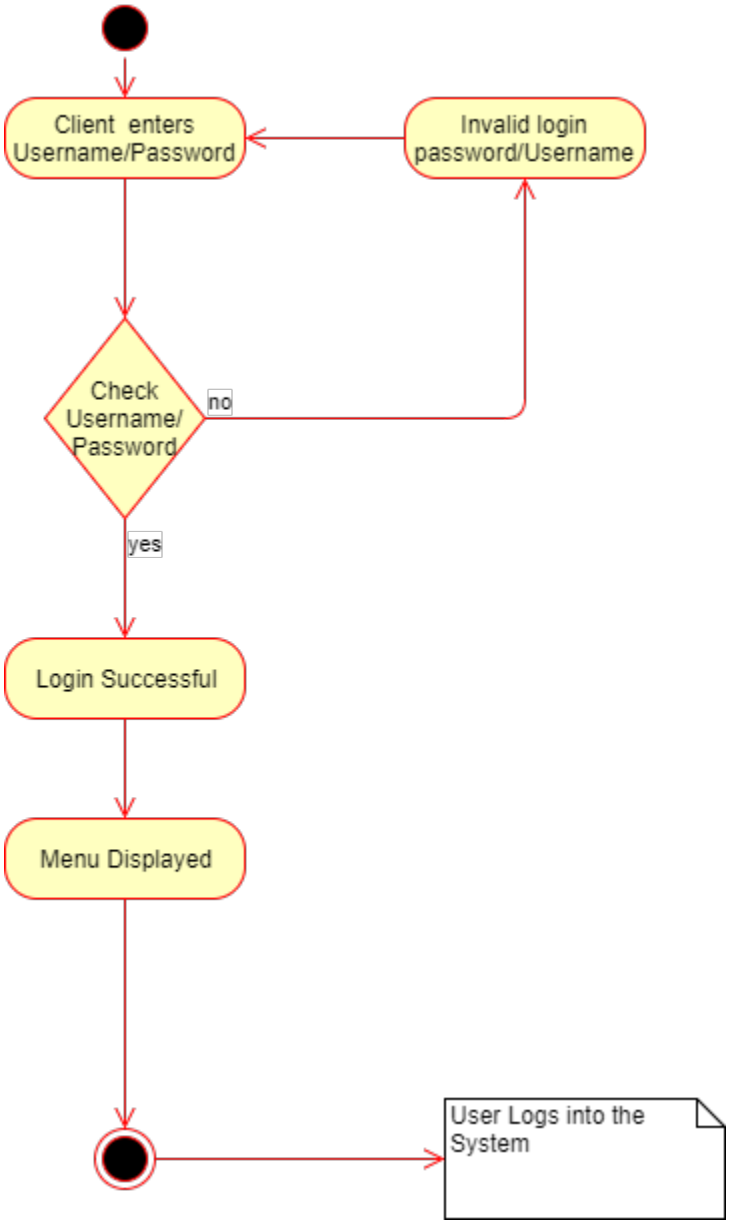

Figure 19 – Login Activity

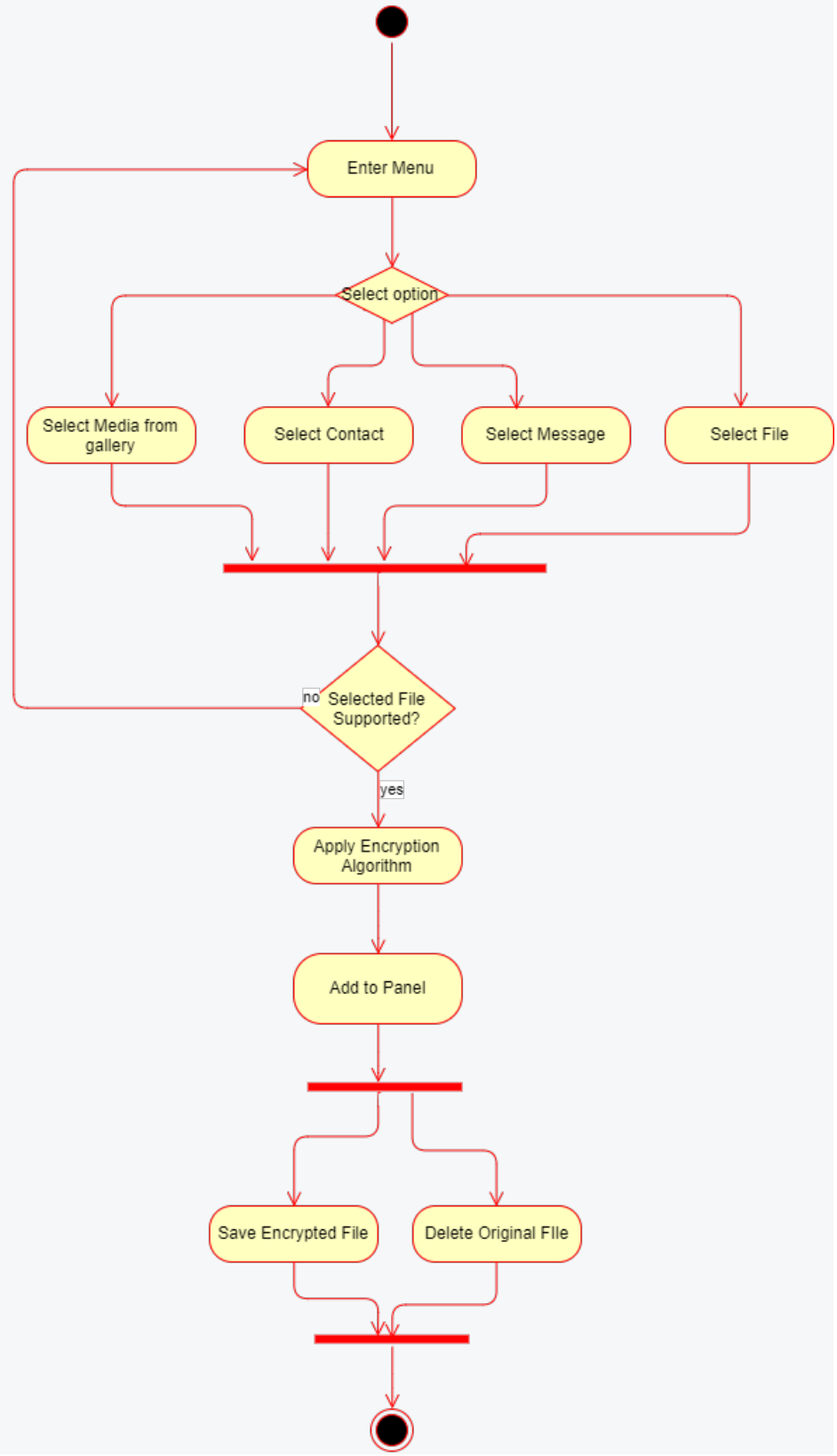

Figure 20 – Add Activity

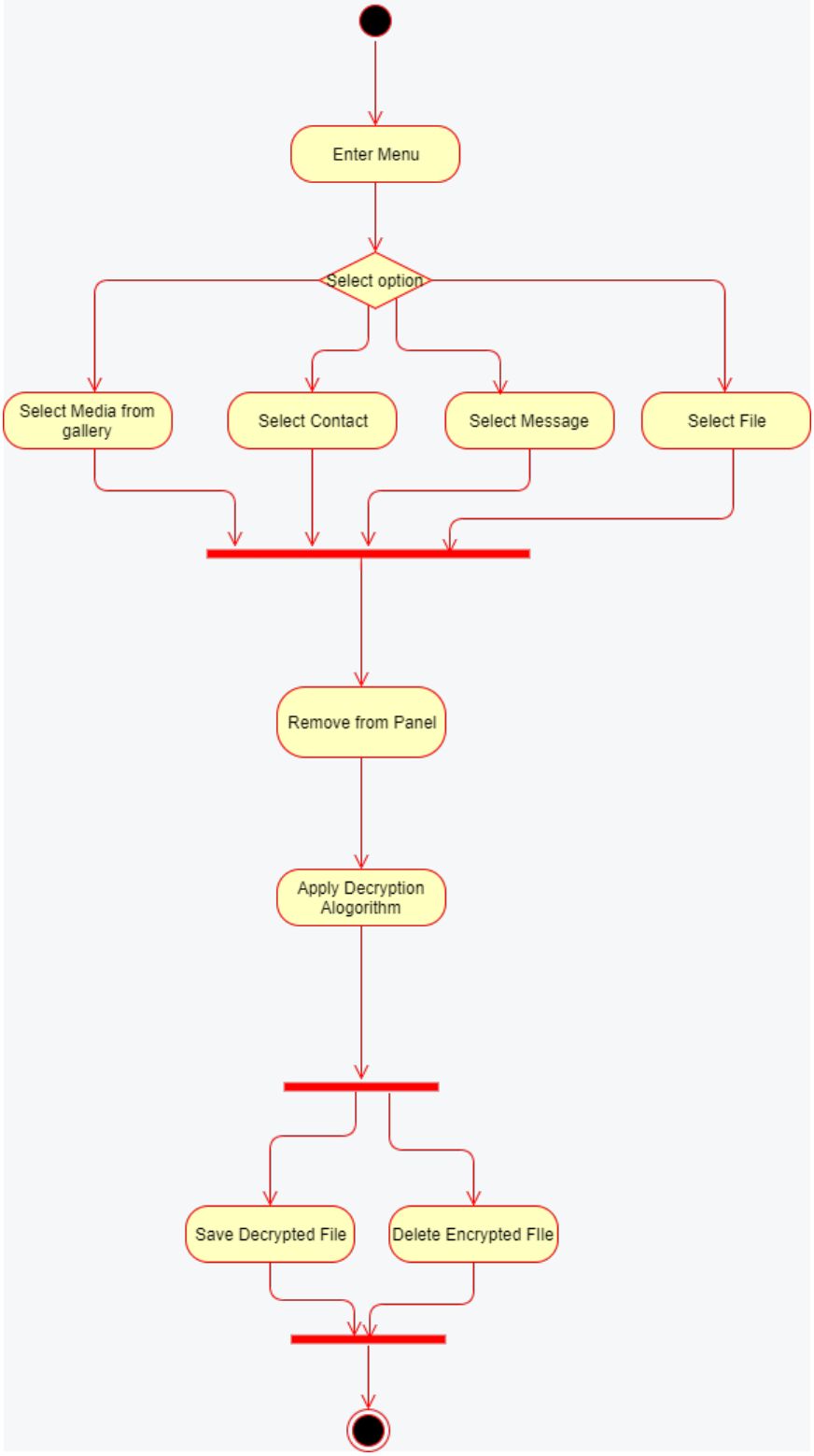

Figure 21 – Remove Activity

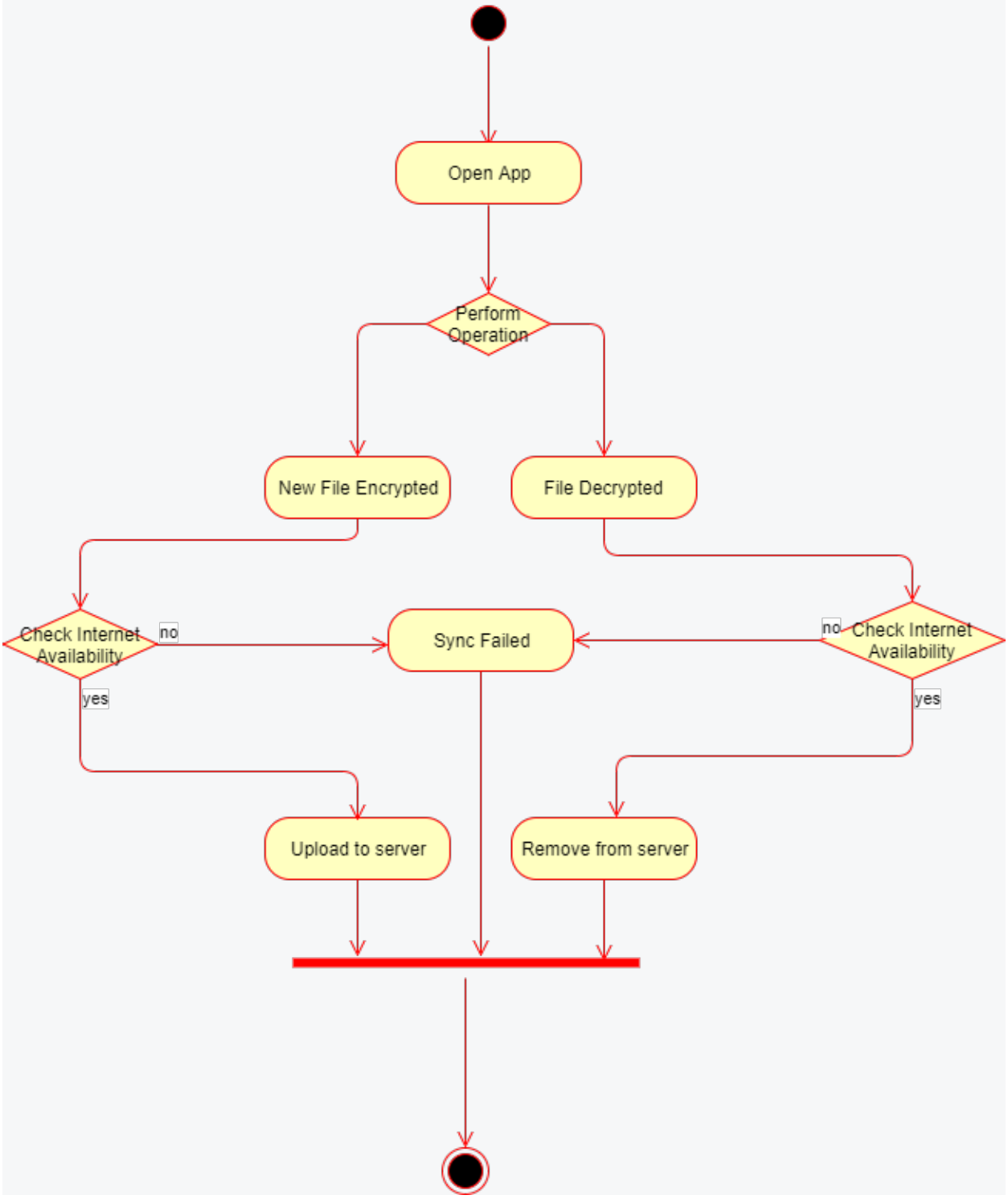

Figure 22 – Sync Activity

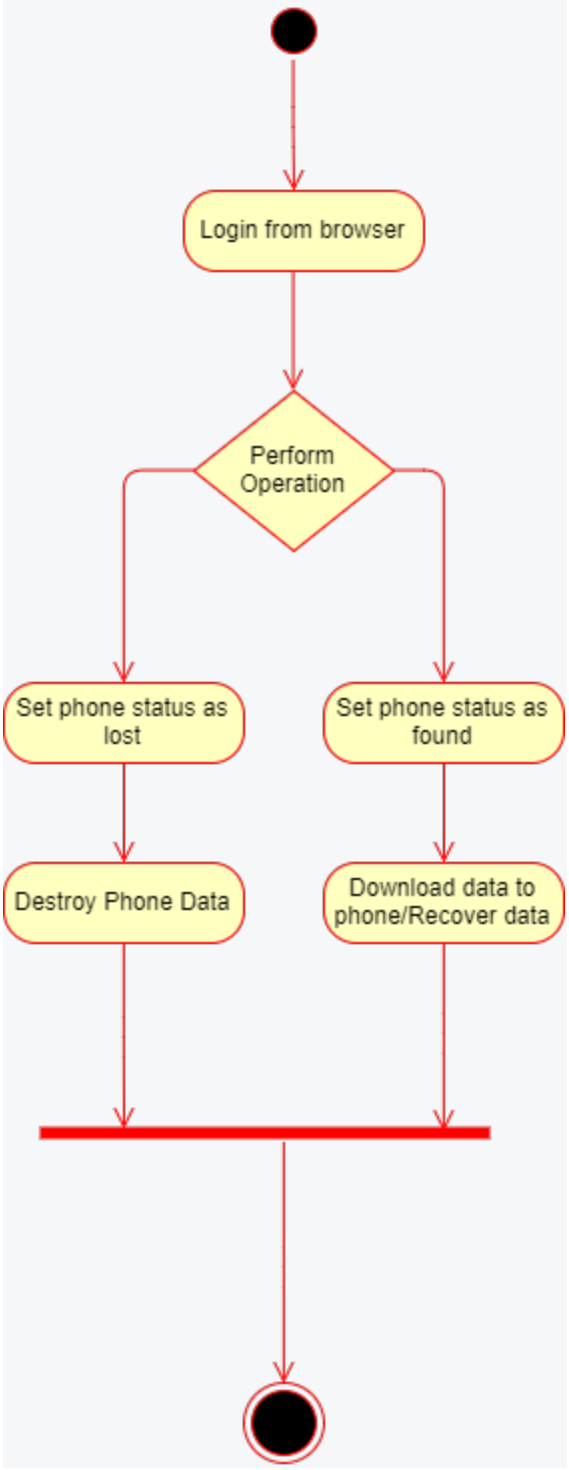

Figure 23 – Update Status Activity

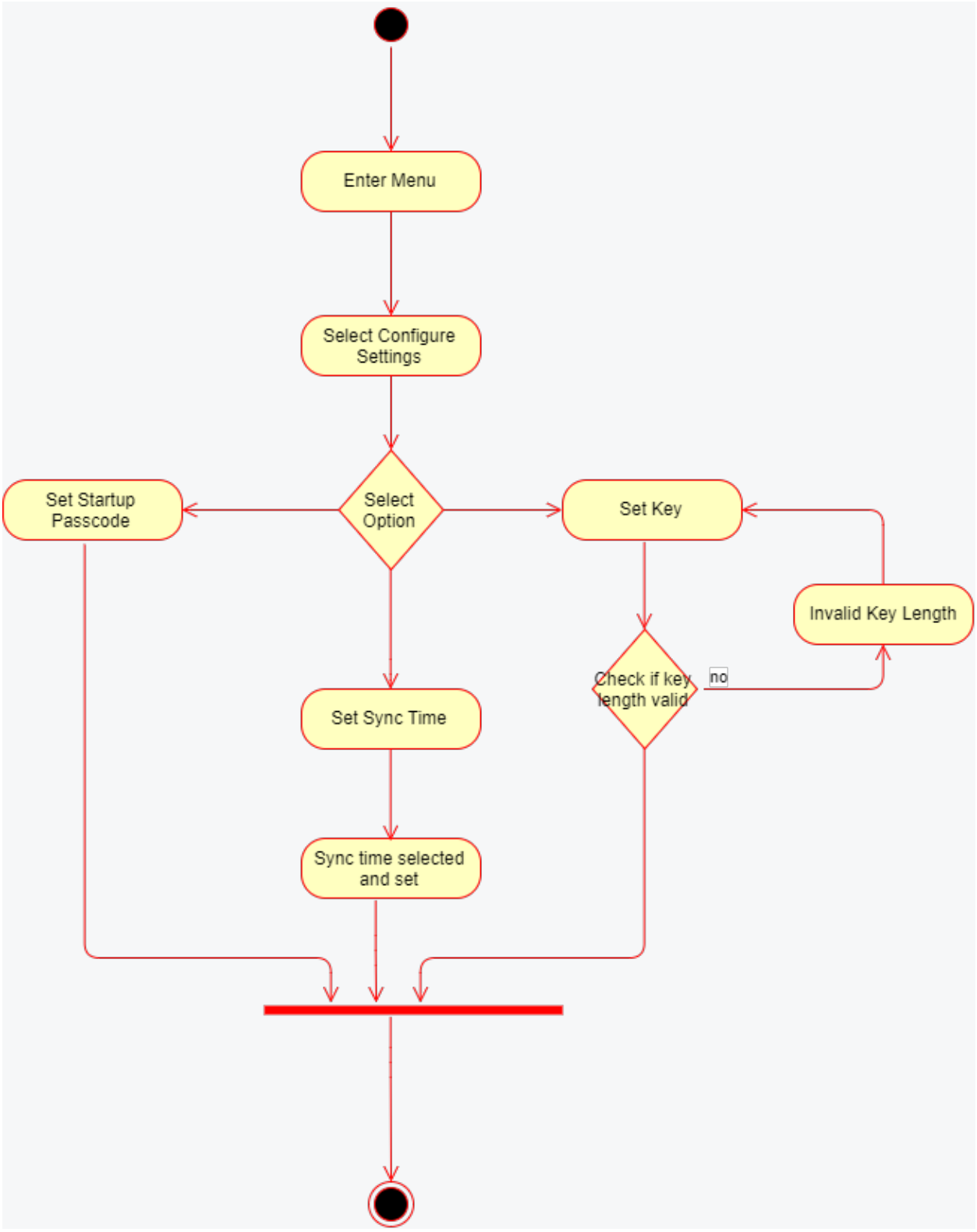

Figure 24 – Configure Settings Activity

Class Diagram

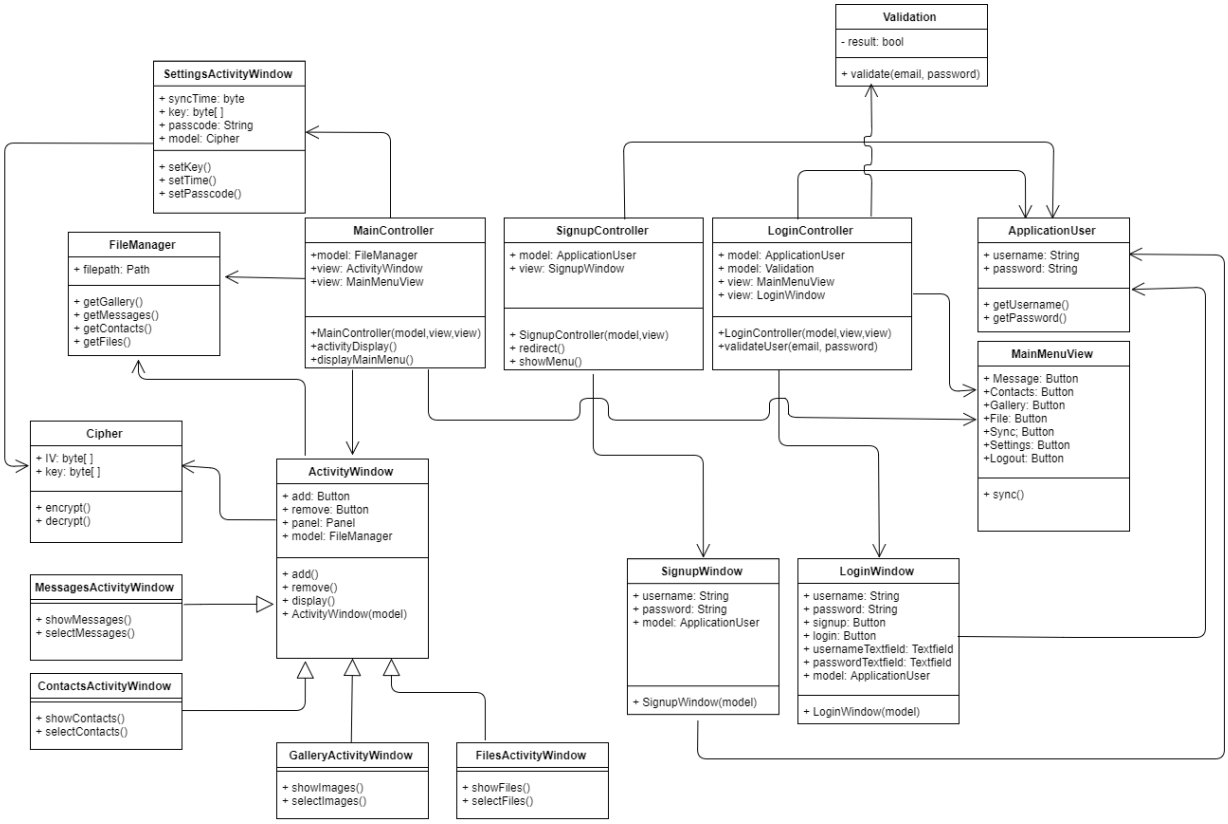

Fig. 4.3.2.10:*Class diagram used Mobile Phone Self Encryption*

User Interface

Main Screen Activity would be displayed to the user. Here the user can choose from 4 different options. He can**Login**. Can access **Offline Mode**. Can **Signup** and can select **Lock Apps** feature

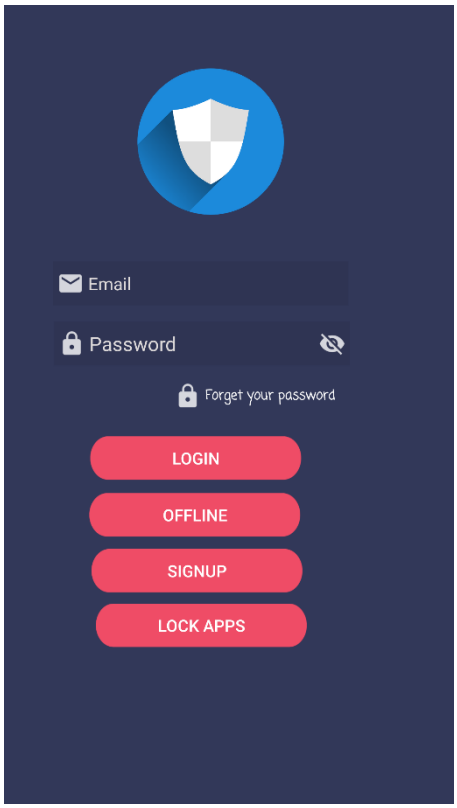

Fig. 4.3.2.15:*Main Screen*

By Clicking on the **Login**, the user would be asked for credentials (Email and Password). After successful login, Menu will be displayed.

Fig. 4.3.2.16:*Menu*

User can select Messages, Contacts, Files(Images and Videos), Settings and Logout

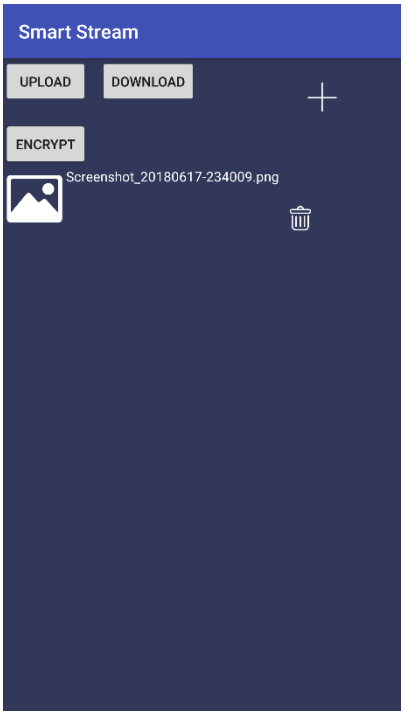

Fig. 4.3.2.17:*Encrypt/Dectypt Images Screen*

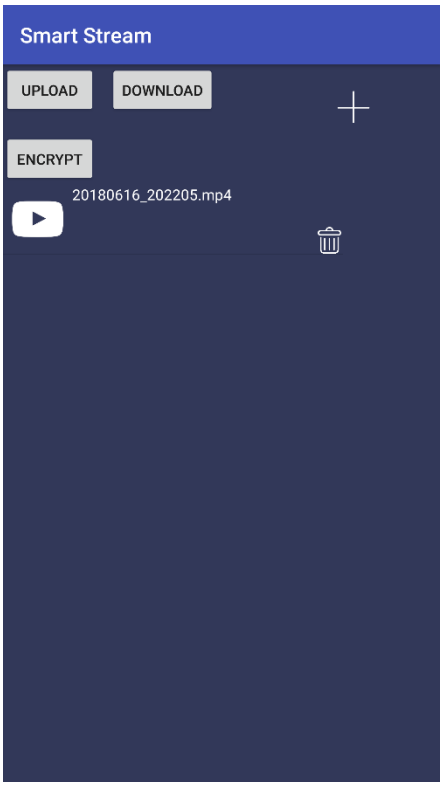

Fig. 4.3.2.18:*Encrypt/Decrypt Videos Screen*

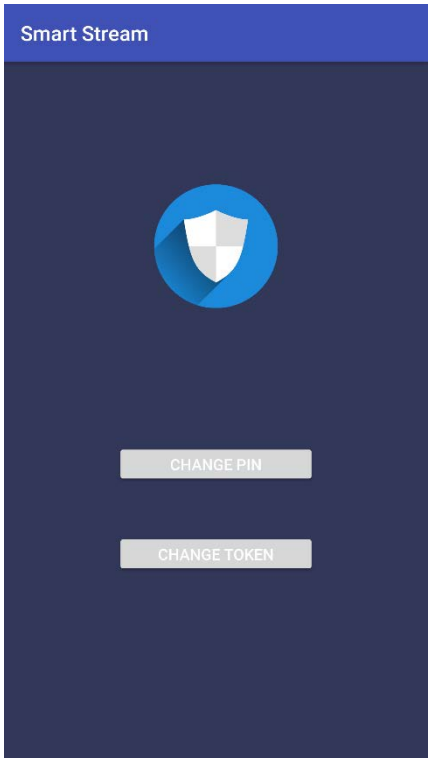

Fig. 4.3.2.19:*Settings Screen*

The user can change offline pin and also change token which is used to wipe data in future.

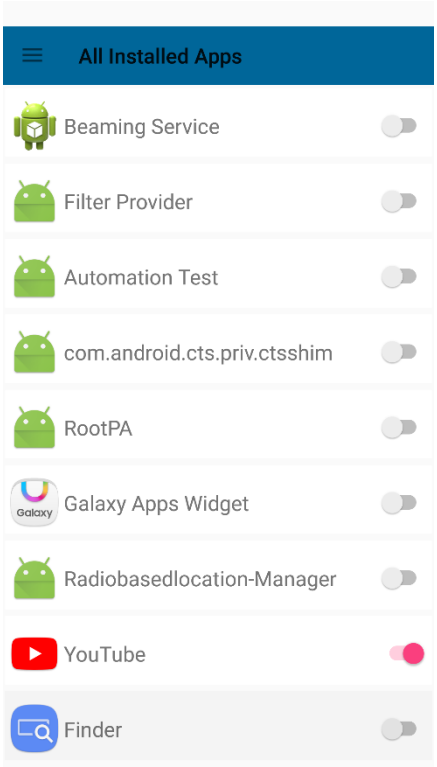

Fig. 4.3.2.20:*Lock Apps Screen*

User can turn on the lock to lock any app.

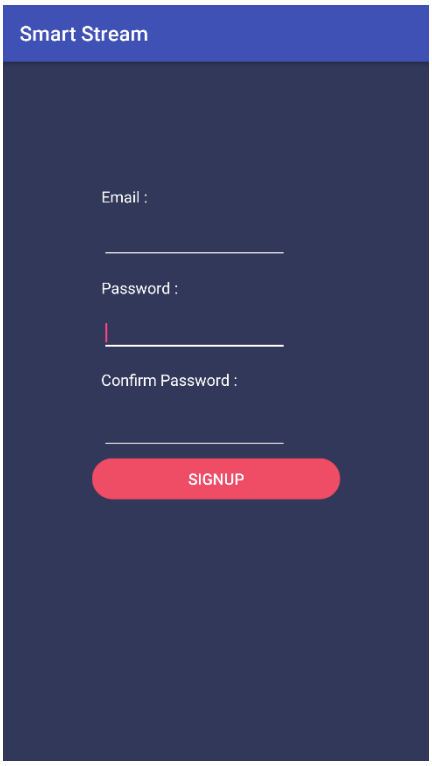

Fig. 4.3.2.21: Signup Screen

4.3.2 Detailed Description of Components

This section describes in detail all the modules of Mobile Phone Self-encryption. These modules have been assigned responsibilities. Modules are further sub classified into components.

4.4 Add Module:

This module performs the encryption for Mobile Self Encryption Application i.e. it encrypts messages, contacts, Gallery (images) and Files (Documents, videos, notes etc.). This module provides the main functionality of the project.

Identification	**Name:** Add
Type	Component
Purpose	This component fulfils following requirement from Software Requirements Specification Document: **Add** **Requirement** The system shall be able to encrypt Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) and save it in encrypted form on Mobile Self Encryption Application respectively. **Description** This feature enables the user to encrypt their Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) and save them on Mobile Application and sync with server.
Function	This component of system gets Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) from Mobile phones Messages, contacts, Gallery (images) and Files (Documents, videos, notes etc.) for encryption.
Subordinates	It has no subordinates
Dependencies	This component is independent.
Interfaces	N/A
Resources	**Software:** Open source android studio libraries, Android SDK, Java libraries.
Processing	Component would get Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) from mobile phone's Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) and encrypt it.
Data	This component uses following information of the application

	Messages, Contacts, Images, Videos, Files etc.

4.5 Remove Module:

This module performs the decryption for Mobile Self Encryption Application i.e. it decrypts messages, contacts, Gallery (images) and Files (Documents, videos, notes etc.). This module provides the main functionality of the project.

Identification	**Name:** Remove
Type	Component
Purpose	This component fulfils following requirement from Software Requirements Specification Document: **Remove** **Requirement** The system shall be able to decrypt Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) and save it in decrypted form on Mobile Phone. **Description** This feature enables the user to decrypt the encrypted Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) and then save these decrypted Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) on Mobile Phone.
Function	This component of system gets encrypted Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) from applications Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) for decryption and decrypt it.
Subordinates	It has no subordinates

Dependencies	This component is independent.
Interfaces	N/A
Resources	**Software:** Open source android studio libraries, Android SDK, Java libraries.
Processing	Component would get encrypted Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) from application Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) and decrypt encrypted message.
Data	This component uses following information of the application Messages, Contacts, Images, Videos, Files etc.

4.6 Sync Module: This module allows users to sync files with server.

Identification	**Name:** Sync
Type	Component
Purpose	This component fulfils following requirement from Software Requirements Specification Document: **Add** **Requirement** The system shall be able to sync encrypted Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) saved on Mobile Phone with server. **Description** This feature enables the user to create back up of their encrypted Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) on

	server.
Function	This component of system gets encrypted Messages, Contacts, Gallery (images) and Files (Documents, videos, notes etc.) from Mobile Applications and sync it with server
Subordinates	It has no subordinates
Dependencies	This component is dependent on add and remove.
Interfaces	N/A
Resources	**Software:** Open source android studio libraries, Android SDK, Java libraries and Firebase
Processing	Component would sync messages, contacts, images, videos, files etc.
Data	This component uses following information of the application Messages, Contacts, Images, Videos, Files etc.

4.7 Settings Module: This module allows users to set key for encryption/decryption and set time for synchronization.

Identification	**Name:** Settings
Type	Component
Purpose	This component fulfils following requirement from Software Requirements Specification Document: **Add** **Requirement** The system shall be able to get key from user and set sync time **Description**

	This feature enables the user to create key for encryption/decryption and enables them to set time for automatic sync
Function	This component of system gets key to encrypt/decrypt data and gets sync time from user so that it can synchronize with server at that specific time period
Subordinates	It has no subordinates
Dependencies	This component is independent
Interfaces	N/A
Resources	**Software:** Open source android studio libraries, Android SDK, Java libraries
Processing	Component would get sync time and key from user
Data	This component uses following information of the application Key, sync time

4.8 Login Module

Identification	**Name:** Login
Type	Component
Purpose	This component fulfils following requirement from Software Requirements Specification Document: **Add** **Requirement** The system shall be able authenticate users and log them in **Description** This feature enables users to enter credentials and log in to their accounts

Function	This component of system authenticates users and does the login process
Subordinates	It has no subordinates
Dependencies	This component is independent
Interfaces	N/A
Resources	**Software:** Open source android studio libraries, Android SDK, Java libraries
Processing	Component would get username and password from user for authentication
Data	This component uses following information of the application Username, password

4.9 Registration Module

Identification	**Name:** Registration
Type	Component
Purpose	This component fulfils following requirement from Software Requirements Specification Document: **Add** **Requirement** The system shall be able to register users **Description** This feature enables the user to create accounts so to login and use those accounts to secure their data on the database of the server
Function	This component of system username, password from user
Subordinates	It has no subordinates
Dependencies	This component is independent

Interfaces	N/A
Resources	**Software:** Open source android studio libraries, Android SDK, Java libraries
Processing	Component would get username and password from user
Data	This component uses following information of the application Username, password

4.4 Reuse and Relationship to other products

Mobile Phone Self-encryption is a new product, Security enhancement is done on already existing operating system of Android, and therefore, android OS is reused. Mobile Phone Self-encryption has the potential to have more features to it. The system is being designed signed in modular fashion.

4.5 Design and Tradeoffs

Mobile Phone Self-encryption is based on client-server architecture that is Mobile phone act as client and Firebase act as server. They are connected through the internet.

The constraints of our project are:

- The user must decrypt all the files before decrypting the key.
- If the user changes the key he cannot retrieve the data on server because it’s with old key so it cannot be decrypted.

Chapter 5: Testing and Evaluation

5.1 Introduction

This test plan chapter describes the appropriate strategies, process and methodologies used to plan, execute and manage testing of the Mobile Self Encryption Android application project. The test plan will ensure that the application meets the customer requirements at an accredited level.

Manual Testing will be followed which includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. Each Unit will be tested separately and then will be integrated with other units, therefore Unit Testing and Integration testing will be followed. For each unit Black box Testing is done and for combined units Acceptance Testing is done.

The test scope includes the Testing of all functional, application performance and use cases requirements listed in the *requirement document*

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed.

This document includes the plan, scope, approach and procedure of Mobile Self Encryption Application test. The pass/fail criteria of the test items are also defined. The Test Plan document documents and tracks the necessary information required to effectively define the approach to be used in the testing of the product.

5.2 Test Items

Based on the Mobile Self Encryption Application requirements and design description, application modules of mobile Android application and non-functional scenario will be tested. The Requirements Defined in Software Requirements Specification and the Design entities as explained in Software Design Document will be tested.

5.3 Features tested

Following Features are Tested:

- Ability to Register new user into System.
- Ability to Log in into the System.
- Ability to encrypt Images from application.
- Ability to encrypt Videos from application.
- Ability to encrypt Files from application.
- Ability to encrypt Messages from application.
- Ability to encrypt Contacts from application.
- Ability to decrypt Images from application.
- Ability to decrypt Videos from application.
- Ability to decrypt Files from application.
- Ability to decrypt Messages from application.
- Ability to decrypt Contacts from application.
- Ability to Upload unsynced Images from application to server.
- Ability to Upload unsynced Videos from application to server.
- Ability to Upload unsynced Files from application to server.
- Ability to Upload unsynced Messages from application to server.
- Ability to Upload unsynced Conctacts from application to server.
- Ability to download unsynced Images from server to application.
- Ability to download unsynced Videos from server to application.
- Ability to download unsynced Files from server to application.
- Ability to download unsynced Messages from server to application.
- Ability to download unsynced Contacts from server to application.
- Feature to set pincode.
- Feature to use offline mode.

5.4 Approach

Acceptance test will be executed based on this acceptance test plan. And after all test cases are executed, a test report will be summarized to show the quality of Mobile Self Encryption Application. Following test approaches will be used in test execution:

- **Unit test**. Developers are responsible for unit test as white-box testing. The implementation of each module and individual component will be verified separately.
- **Integration test**. After the unit test is passed above the defined quality threshold, testers will execute the integration test cases. After all the modules are integrated, it's crucial to test the product as a black-box. End-to-end scenarios will be tested to ensure the communication functionality.
- **Regression test**. After developers fix the bug in one feature, regression test will be executed by testers to ensure that the other functions are not affected.
- **Field test**. Firstly, untrained end users recreate one or more existing (but narrow) mass observation events in the Self Encryption Android Application. A number of observers will be invited to help with evaluation. After that, post event questionnaires will be used to collect quantitative usage data as well as qualitative data and further improvement will be taken into consideration.
- **Positive and negative testing design technique**. This approach will be combined with unit test and integration test. Test cases are designed in obvious scenarios, which ensure that all functional requirements are satisfied. What's more, different test cases will also be covered to show how the system reacts with invalid operations.

5.5 Item Pass/Fail Criteria

Details of the test cases are specified in section Test Deliverables. Following the principles outlined below, a test item would be judged as pass or fail.

- Preconditions are met
- Inputs are carried out as specified
- The result works as what specified in output => Pass

- The system doesn't work or not the same as output specification => Fail

5.6 Suspension Criteria and Resumption Requirements

Any bugs found can be fixed by developers quickly and no need to start the testing process from the beginning. However, when major bugs will block the some test cases as they are interdependent and the testing has to be paused. The test will restart from the very beginning until the major error is solved.

5.7 Test Deliverables

Following are the Test Cases:

Test Case Name	Registration(with valid data)
Test Case No	1
Description	Testing feature to register a new user into system
Preconditions	The user must have installed Mobile Self Encryption Application in android operating system
Input Values	Enter username and password Click sign up button
Valid Inputs	Enter valid username and password and click on sign up button
Steps	First select the Self Encryption android application installed in Android Operating System then enter username and password in the respective fields on the registration page
Expected Output	Registration successful, log in user into the system
Actual Output	Registration successful, log in user into the system

Test Case Name	Registration(with invalid data)
Test Case No	2
Description	Testing feature to register a new user into system

Preconditions	The user must have installed Mobile Self Encryption Application in android operating system
Input Values	Enter username and password Click sign up button
Valid Inputs	Enter invalid username and password and click on sign up button
Steps	First select the Self Encryption android application installed in Android Operating System then enter username and password in the respective fields on the registration page
Expected Output	Registration failed, shows invalid data pop up message
Actual Output	Registration failed, shows invalid data pop up message

Test Case Name	Login(with correct username and password)
Test Case No	3
Description	Testing feature to login into the system with correct data
Testing Technique Used	Unit Testing
Preconditions	The user must have installed Mobile Self Encryption Application in android operating system
Input Values	Enter username and password Click sign in button
Valid Inputs	Enter correct username and password and click on sign in button
Steps	First select the Self Encryption android application installed in Android Operating System then enter username and password in the respective fields on the login page
Expected Output	Login successful application takes user to main menu
Actual Output	Login successful application takes user to main menu

Test Case Name	Login(with incorrect username and password)

Test Case No	4
Description	Testing feature to login into the system with false data
Testing Technique Used	Unit Testing
Preconditions	The user must have installed Mobile Self Encryption Application in android operating system
Input Values	Enter incorrect username and password Click sign in button
Valid Inputs	Enter incorrect username and password and click on sign in button
Steps	First select the Self Encryption android application installed in Android Operating System then enter username and password in the respective fields on the login page
Expected Output	Login failed application stays on login page
Actual Output	Login failed application stays on login page

Test Case Name	Feature Choice image option
Test Case No	5
Description	Testing Feature Choose images
Testing Technique Used	Unit Testing
Preconditions	Application should be installed in Android Operating System
Input Values	Select image button on the main menu
Valid Inputs	Select image button on the main menu
Steps	Select the Mobile Self Encryption application installed in Android Operating System Log in into application Select Files option Select image option

Expected Output	Mobile Self Encryption android application displays Image page to the user
Actual Output	Mobile Self Encryption android application displays Image page to the user

Test Case Name	Feature Choice Video option
Test Case No	6
Description	Testing Feature Choose Videos
Testing Technique Used	Unit Testing
Preconditions	Application should be installed in Android Operating System
Input Values	Select video button on the main menu
Valid Inputs	Select video button on the main menu
Steps	Select the Mobile Self Encryption application installed in Android Operating System Log in into application Select Files option Select video option
Expected Output	Mobile Self Encryption android application displays video page to the user
Actual Output	Mobile Self Encryption android application displays video page to the user

Test Case Name	Feature Choice select file option
Test Case No	7
Description	Testing Feature Choose Files
Testing Technique Used	Unit Testing

Preconditions	Application should be installed in Android Operating System
Input Values	Select File button on the main menu
Valid Inputs	Select File button on the main menu
Steps	Select the Mobile Self Encryption application installed in Android Operating System Log in into application Select File option
Expected Output	Mobile Self Encryption android application displays Files page to the user
Actual Output	Mobile Self Encryption android application displays Files page to the user

Test Case Name	Feature Choice Messages option
Test Case No	8
Description	Testing Feature Choose Messages
Testing Technique Used	Unit Testing
Preconditions	Application should be installed in Android Operating System
Input Values	Select Messages button on the main menu
Valid Inputs	Select Messages button on the main menu
Steps	Select the Mobile Self Encryption application installed in Android Operating System Log in into application Select Messages option
Expected Output	Mobile Self Encryption android application displays Messages page to the user
Actual Output	Mobile Self Encryption android application displays Messages page to the user

Test Case Name	Feature Choice Contacts option
Test Case No	9
Description	Testing Feature Choose Contacts
Testing Technique Used	Unit Testing
Preconditions	Application should be installed in Android Operating System
Input Values	Select Contacts button on the main menu
Valid Inputs	Select Contacts button on the main menu
Steps	Select the Mobile Self Encryption application installed in Android Operating System Log in into application Select Contacts option
Expected Output	Mobile Self Encryption android application displays Contacts page to the user
Actual Output	Mobile Self Encryption android application displays Contacts page to the user

Test Case Name	Feature Choice Settings option
Test Case No	10
Description	Testing Feature Choose Settings
Testing Technique Used	Unit Testing
Preconditions	Application should be installed in Android Operating System
Input Values	Select Settings button on the main menu
Valid Inputs	Select Settings button on the main menu
Steps	Select the Mobile Self Encryption application installed in Android Operating System

	Log in into application Select Settings option
Expected Output	Mobile Self Encryption android application displays Settings page to the user
Actual Output	Mobile Self Encryption android application displays Settings page to the user

Test Case Name	Encrypt Image from Gallery
Test Case No	11
Description	Testing Select add/encrypt Image from Gallery Feature for Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Images option from application Choice Menu
Input Values	Choose images from Gallery
Valid Inputs	Choose Select Image form Gallery option by clicking on the add images button Press encrypt
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select image feature which is displayed on Feature Choice Menu choose the add image button displaying on the top of screen select images press encrypt button
Expected Output	The Gallery of the android mobile phone should be accessed by Self Encryption Application and should enable user to select desired image from Gallery to encrypt

Actual Output	The Gallery of the android mobile phone is accessed by Self Encryption Application enabling user to select desired image from Gallery to encrypt

Test Case Name	Encrypt Videos from Gallery
Test Case No	12
Description	Testing Select add/encrypt Video from Gallery Feature for Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Video option from application Choice Menu
Input Values	Choose Video from Gallery
Valid Inputs	Choose Select Video form Gallery option by clicking on the add Video button Press encrypt button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select Video feature which is displayed on Feature Choice Menu choose the add Video button displaying on the top of screen select videos press encrypt
Expected Output	The Gallery of the android mobile phone should be accessed by Self Encryption Application and should enable user to select desired Video from Gallery to encrypt
Actual Output	The Gallery of the android mobile phone is accessed by Self Encryption Application enabling user to select desired Video from Gallery to encrypt

Test Case Name	Encrypt Files from File manager
Test Case No	13
Description	Testing add/encrypt Files from File manager Feature for Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Files option from application Choice Menu
Input Values	Choose add/encrypt File from file manager
Valid Inputs	Choose add/encrypt files form File Manager option by clicking on the add Files button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Files feature which is displayed on Feature Choice Menu choose the add files button displaying on the top of screen select files press encrypt
Expected Output	The File manager of the android mobile phone should be accessed by Self Encryption Application and should enable user to select desired File from Memory to encrypt
Actual Output	The File manager of the android mobile phone is accessed by Self Encryption Application enabling user to select desired File from memory to encrypt

Test Case Name	Encrypt Messages from System
Test Case No	14
Description	Testing add/encrypt Messages from memory Feature for

	Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Messages option from application Choice Menu
Input Values	Choose add/encrypt Messages from System
Valid Inputs	Choose add/encrypt Messages form System option by clicking on the add Messages button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select Messages feature which is displayed on Feature Choice Menu choose the add Messages button displaying on the top of screen select meassages press encrypt
Expected Output	The memory of the android mobile phone should be accessed by Self Encryption Application and should enable user to select desired Message from Memory to encrypt
Actual Output	The memory of the android mobile phone is accessed by Self Encryption Application enabling user to select desired Message from memory to encrypt

Test Case Name	Encrypt Contacts from System memory
Test Case No	15
Description	Testing add/encrypt Contacts from memory Feature for Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Contacts option from application Choice Menu

Input Values	Choose add/encrypt Contacts from System
Valid Inputs	Choose add/encrypt Contacts form System option by clicking on the add Contacts button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select Contacts feature which is displayed on Feature Choice Menu choose the add Contacts button displaying on the top of screen select contacts press encrypt
Expected Output	The memory of the android mobile phone should be accessed by Self Encryption Application and should enable user to select desired Contact from Memory
Actual Output	The memory of the android mobile phone is accessed by Self Encryption Application enabling user to select desired Contact from memory

Test Case Name	Decrypt Image
Test Case No	16
Description	Testing Decrypt Image from Application Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Images option from application Choice Menu
Input Values	Decrypt images from application images vault
Valid Inputs	Choose decrypt Image form application images vault by clicking on the delete button
Steps	First select the Self Encryption android application installed in Android operating system

	Log in into system/use offline mode choose the select image feature which is displayed on Feature Choice Menu choose the delete image button displaying in front of selected image
Expected Output	Image decrypted and removed from Application vault moves back to original location
Actual Output	Image decrypted and removed from Application vault moves back to original location

Test Case Name	Decrypt Video
Test Case No	17
Description	Testing Decrypt Video from Application Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Video option from application Choice Menu
Input Values	Decrypt Video from application Video vault
Valid Inputs	Choose decrypt Video form application Video vault by clicking on the delete button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select Video feature which is displayed on Feature Choice Menu choose the delete Video button displaying in front of selected Video
Expected Output	Video decrypted and removed from Application vault moves back to original location

Actual Output	Video decrypted and removed from Application vault moves back to original location

Test Case Name	Decrypt File
Test Case No	18
Description	Testing Decrypt File from Application Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected File option from application Choice Menu
Input Values	Decrypt File from application File vault
Valid Inputs	Choose decrypt File form application File vault by clicking on the delete button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select File feature which is displayed on Feature Choice Menu choose the delete File button displaying in front of selected File
Expected Output	File decrypted and removed from Application vault moves back to original location
Actual Output	File decrypted and removed from Application vault moves back to original location

Test Case Name	Decrypt Message
Test Case No	19
Description	Testing Decrypt Message from Application Feature for Self

	Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Message option from application Choice Menu
Input Values	Decrypt Message from application Message vault
Valid Inputs	Choose decrypt Message form application Message vault by clicking on the delete button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select Message feature which is displayed on Feature Choice Menu choose the delete Message button displaying in front of selected Message
Expected Output	Message decrypted and removed from Application vault moves back to original location
Actual Output	Message decrypted and removed from Application vault moves back to original location

Test Case Name	Decrypt Contact
Test Case No	20
Description	Testing Decrypt Contact from Application Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Contact option from application Choice Menu
Input Values	Decrypt Contact from application Contact vault
Valid Inputs	Choose decrypt Contact form application Contact vault by

	clicking on the delete button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select Contact feature which is displayed on Feature Choice Menu choose the delete Contact button displaying in front of selected Contact
Expected Output	Contact decrypted and removed from Application vault moves back to original location
Actual Output	Contact decrypted and removed from Application vault moves back to original location

Test Case Name	Upload Images
Test Case No	21
Description	Testing Upload Images from Application to server a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Image option from application Choice Menu
Input Values	Upload Image from application Images vault
Valid Inputs	Choose upload image form application image vault by clicking on the upload button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Image feature which is displayed on Feature Choice Menu choose the upload image button displaying on the top of

	screen
Expected Output	Image successfully uploaded on the server
Actual Output	Image successfully uploaded on the server

Test Case Name	Upload Videos from application
Test Case No	22
Description	Testing Upload Videos from Application to server a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Video option from application Choice Menu
Input Values	Upload Video from application Video vault
Valid Inputs	Choose upload Video form application Video vault by clicking on the upload button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Video feature which is displayed on Feature Choice Menu choose the upload Video button displaying on the top of screen
Expected Output	Video successfully uploaded on the server
Actual Output	Video successfully uploaded on the server

Test Case Name	Upload File from application

Test Case No	23
Description	Testing Upload Files from Application to server a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected File option from application Choice Menu
Input Values	Upload File from application File vault
Valid Inputs	Choose upload File form application File vault by clicking on the upload button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select File feature which is displayed on Feature Choice Menu choose the upload File button displaying on the top of screen
Expected Output	File successfully uploaded on the server
Actual Output	File successfully uploaded on the server

Test Case Name	Upload Message from application
Test Case No	24
Description	Testing Upload Message from Application to server a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Message option from application Choice Menu
Input Values	Upload Message from application Message vault

Valid Inputs	Choose upload Message form application Message vault by clicking on the upload button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Message feature which is displayed on Feature Choice Menu choose the upload Message button displaying on the top of screen
Expected Output	Message successfully uploaded on the server
Actual Output	Message successfully uploaded on the server

Test Case Name	Upload Contact from application
Test Case No	25
Description	Testing Upload Contact from Application to server a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Contact option from application Choice Menu
Input Values	Upload Contact from application Contact vault
Valid Inputs	Choose upload Contact form application Contact vault by clicking on the upload button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Contact feature which is displayed on Feature Choice Menu choose the upload Contact button displaying on the top of

	screen
Expected Output	Contact successfully uploaded on the server
Actual Output	Contact successfully uploaded on the server

Test Case Name	Download Images from server to mobile
Test Case No	26
Description	Testing Download Images from Server to Application a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Image option from application Choice Menu
Input Values	Download Image from server to application vault
Valid Inputs	Choose Download image form server to application vault by clicking on the download button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Image feature which is displayed on Feature Choice Menu choose the download image button displaying on the top of screen
Expected Output	Image successfully downloaded from server to application
Actual Output	Image successfully downloaded from server to application

Test Case Name	Download Video from server to mobile

Test Case No	27
Description	Testing Download Video from Server to Application a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Video option from application Choice Menu
Input Values	Download Video from server to application vault
Valid Inputs	Choose Download Video form server to application vault by clicking on the download button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Video feature which is displayed on Feature Choice Menu choose the download Video button displaying on the top of screen
Expected Output	Video successfully downloaded from server to application
Actual Output	Video successfully downloaded from server to application

Test Case Name	Download File from server to mobile
Test Case No	28
Description	Testing Download File from Server to Application a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected File option from application Choice Menu
Input Values	Download File from server to application vault

Valid Inputs	Choose Download File form server to application vault by clicking on the download button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select File feature which is displayed on Feature Choice Menu choose the download File button displaying on the top of screen
Expected Output	File successfully downloaded from server to application
Actual Output	File successfully downloaded from server to application

Test Case Name	Download Message from server to mobile
Test Case No	29
Description	Testing Download Message from Server to Application a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Message option from application Choice Menu
Input Values	Download Message from server to application vault
Valid Inputs	Choose Download Message form server to application vault by clicking on the download button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Message feature which is displayed on Feature Choice Menu choose the download Message button displaying on the top of

	screen
Expected Output	Message successfully downloaded from server to application
Actual Output	Message successfully downloaded from server to application

Test Case Name	Download Contact from server to mobile
Test Case No	30
Description	Testing Download Contacts from Server to Application a Feature for Self Encryption Application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Contact option from application Choice Menu
Input Values	Download Contact from server to application vault
Valid Inputs	Choose Download Contact form server to application vault by clicking on the download button
Steps	First select the Self Encryption android application installed in Android operating system Log in into system choose the select Contact feature which is displayed on Feature Choice Menu choose the download Contact button displaying on the top of screen
Expected Output	Contact successfully downloaded from server to application
Actual Output	Contact successfully downloaded from server to application

Test Case Name	Login into Offline mode(with correct pin code)

Test Case No	31
Description	Testing feature to login into the system when offline
Testing Technique Used	Unit Testing
Preconditions	The user must have installed Mobile Self Encryption Application in android operating system
Input Values	Click Offline mode button Enter pin code
Valid Inputs	Click on Offline mode button Enter pin code
Steps	First select the Self Encryption android application installed in Android Operating System click on Offline mode button Enter pin code
Expected Output	Login successful application takes user to main menu
Actual Output	Login successful application takes user to main menu

Test Case Name	Login into Offline mode(with incorrect pin code)
Test Case No	32
Description	Testing feature to login into the system when offline
Testing Technique Used	Unit Testing
Preconditions	The user must have installed Mobile Self Encryption Application in android operating system
Input Values	Click Offline mode button Enter incorrect pin code
Valid Inputs	Click on Offline mode button

	Enter incorrect pin code
Steps	First select the Self Encryption android application installed in Android Operating System click on Offline mode button Enter incorrect pin code
Expected Output	Login Failed, pop up message invalid pin code, user stays on same page
Actual Output	Login Failed, pop up message invalid pin code, user stays on same page

Test Case Name	Feature set pin code
Test Case No	33
Description	Testing Feature Setting pin code
Testing Technique Used	Unit Testing
Preconditions	Application should be installed in Android Operating System User must select Setting option
Input Values	Press set pin code, enter pin code
Valid Inputs	Press set pin code, enter pin code
Steps	Select the Mobile Self Encryption application installed in Android Operating System Log in into application/use offline mode Select Settings option Select set pin option Enter pin code
Expected Output	Pin code successfully updated
Actual Output	Pin code successfully updated

Test Case Name	Feature to view Image in application
Test Case No	34
Description	Testing feature to view Image in the application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Image option from application Choice Menu
Input Values	Click on image icon in image vault
Valid Inputs	Click on image icon in image vault
Steps	First select the Self Encryption android application installed in Android operating system Log in into system/use offline mode choose the select Image feature which is displayed on Feature Choice Menu click on image icon
Expected Output	The original image is shown
Actual Output	The original image is shown

Test Case Name	Feature to view Video in application
Test Case No	34
Description	Testing feature to view Video in the application
Testing Technique Used	Unit Testing
Preconditions	The user must have selected Video option from application Choice Menu
Input Values	Click on Video icon in Video vault
Valid Inputs	Click on Video icon in Video vault
Steps	First select the Self Encryption android application installed in

	Android operating system Log in into system/use offline mode choose the Video feature which is displayed on Feature Choice Menu click on Video icon
Expected Output	The original Video is shown
Actual Output	The original Video is shown

5.8 Environmental Needs

Hardware

- Mobile with Android platform

Software

- Mobile Platform: Android 3.0/3.1 or later (Eclair Based on Linux Kernel 2.6.29 or later
- Eclipse 3.4 (Ganymede) or 3.5 (Galileo) with ADT Plugin

5.9 Responsibilities, Staffing and Training Needs

Responsibilities

- Yawar Khan is responsible for Acceptance Testing
- Hamza Yousaf is responsible for Integration Testing
- Ali Anwar is responsible for testing each separate unit that is Unit Testing.

Skills

- Skills needed to test Mobile Self Encryption Application using Mobile/Android Application

5.10 Risks and contingencies

We have tried to test on various android platforms as much as possible, but it's impossible to test for all android platforms. What's more, mobile Android application is tested on Android devices and is tested on limited mobiles, thus we cannot predict the system behavior on the other mobile platforms (e.g. iPhone, Blackberry, Symbian platform etc.). Further investigation is required to verify and improve Self Encryption Application.

Chapter 6: Future Work

This project will be a benchmark for future encryption project. For now we worked on a single stream cipher. In the future we will:

- We will work on different stream ciphers and users will choose a cipher according to his/her requirements i.e. if they want security over speed they will choose different cipher, vice versa.
- We will also work on block cipher to look at its effect on functionality of our project.
- Investigate potential problems and benefits when merging different applications together such as merge of encryption algorithms and compression algorithms

Chapter 7: Conclusion

Lack of effective protection of sensitive data in mobile devices is a major concern that prevents the mobile devices from being used. The proposedsystem will remove the barrier and enable employees to enjoy the high efficiency and convenience brought by mobile devices. It will provide user to store data in encrypted form so it cannot be used falsely and also provide backup in case phone is lost.

Keeping our private data secure, it means only user can access and use the data. Encryption is used for securing files from thefts. AES algorithm is mostly used for encryption scheme. Approaches to encryption are endpoint encryption, file and folder encryption.

The purpose of this project is to allow users to store sensitive data on their mobile phones without having to worry about its confidentiality even if the mobile phone is lost. This system is developed so that employees and other mobile users can store and operate on sensitive data on their mobile phones without having to worry

of it being leaked. Security is the main purpose of the project and security provided using encryption scheme.

Bibliography

Similar Projects at MCS

1. BitVise XTS-AES Based Disk Encryption Software
   by Myra Khalid, Laraib Zahid and Usama Ahmad

2. A similar approach was made by Amit Banerjee, Muhamadul Hassan, MD. Auhidur Rahman and Rajesh Chapagain, Department of Computer Science, South Asian University, New Delhi 110021, India.

   CLOAK: A Stream Cipher Based Encryption Protocol for Mobile Cloud Computing.

Link: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8016572

Appendix (Proposal)

Description:

This project allows users to store sensitive data on their mobile phones without having to worry about its confidentiality even if the mobile phone is lost. This system is developed so that employees and other mobile users can store and operate on sensitive data on their mobile phones without having to worry of it being leaked. This software project concentrates on securing data on mobile phones by storing it in an encrypted form. This data is encrypted with a stream cipher whose key is stored on a trusted server. When the mobile device is lost, it sends a report to the server and the server then destroys the respective key so that the data on the mobile can never be decrypted and remains confidential.

Scope of Work:

Suggested and sponsored by **NESCOM**. Can be utilized in android devices.

Academic Objective:

Studying the different security techniques and the applying our own knowledge to overcome such vulnerabilities. To utilize the knowledge of all the software engineering related subjects which we have studied during our tenure.

End Goal Objective:

To build a software/application, that prevents confidential information being lost or misused.

Pseudo code for components

Video Activity.java

```
public class VideoActivity extends AppCompatActivity {

    String realPath[];
    Button uploadVideo, downloadVideo;
    DataInputStream reader;
    DataOutputStream writer;
    String[] filePathStrings;
    String[] fileNameStrings;

    ListView lv;
    MyListAdapterVideos listAdapter;

    File file;
    File[] listFile;

private String[] videoListServer;
private Uri[] fileUriArray;

    StorageReference storageReference;
    DatabaseReference databaseReference;
final String databasePathVideos="Encrypted_Videos";
    ProgressDialog progressDialog;


@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_video);

storageReference = FirebaseStorage.getInstance().getReference();
databaseReference= FirebaseDatabase.getInstance().getReference(databasePathVideos);
progressDialog=new ProgressDialog(VideoActivity.this);

        Button add = (Button) findViewById(R.id.addVideo);
        add.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
                Intent i = new Intent(Intent.ACTION_PICK);
                i.setType("video/*");
                i.putExtra(Intent.EXTRA_ALLOW_MULTIPLE, true);
//i.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(i, 1);

            }
        });

uploadVideo = (Button) findViewById(R.id.uploadVideos);
uploadVideo.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
                getImageNamesFromDatabase(1);
final ProgressDialog progressDialog=new ProgressDialog(VideoActivity.this);
                progressDialog.setTitle("Uploading..");
                progressDialog.setMessage("Please wait");
                progressDialog.show();
```

```
final Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
@Override
public void run() {
progressDialog.hide();
                        Intent intent=new
Intent(VideoActivity.this,VideoActivity.class);
                        startActivity(intent);
//    Toast.makeText(VideoActivity.this, "Please Wait File is Being Encrypted",
Toast.LENGTH_SHORT).show();
}
                },3000);


            }
        });
downloadVideo = (Button)findViewById(R.id.downloadVideos);
downloadVideo.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
                getImageNamesFromDatabase(2);
            }
        });



file = new File(Environment.getExternalStorageDirectory() + File.separator +
"EncryptedVideos");
file.mkdirs();
if (file.isDirectory()) {
listFile = file.listFiles();


if(listFile == null){
filePathStrings = new String[0];
fileNameStrings = new String[0];
fileUriArray = new Uri[0];

for (int i = 0; i <0; i++) {
filePathStrings[i] = listFile[i].getAbsolutePath();
fileNameStrings[i] = listFile[i].getName();
fileUriArray[i] = Uri.fromFile(listFile[i]);
                }
            }
else {
filePathStrings = new String[listFile.length];
fileNameStrings = new String[listFile.length];
fileUriArray = new Uri[listFile.length];

for (int i = 0; i <listFile.length; i++) {
filePathStrings[i] = listFile[i].getAbsolutePath();
fileNameStrings[i] = listFile[i].getName();
fileUriArray[i] = Uri.fromFile(listFile[i]);
                }
            }
        }

        Button encrypt = (Button) findViewById(R.id.encVideo);
        encrypt.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
if(realPath[0] == null){
                    Toast.makeText(VideoActivity.this, "please choose first",
```

```
Toast.LENGTH_SHORT).show();
                }
else{
                    enc3(realPath);
                    File a = new File(realPath[0]);
                    a.delete();
                    Intent scanIntent = new
Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
                    scanIntent.setData(Uri.fromFile(a));
                    sendBroadcast(scanIntent);

final ProgressDialog progressDialog=new ProgressDialog(VideoActivity.this);
                    progressDialog.setTitle("Encrypting..");
                    progressDialog.setMessage("Please wait");
                    progressDialog.show();


final Handler handler = new Handler();
                    handler.postDelayed(new Runnable() {
@Override
public void run() {
progressDialog.hide();
                            Intent intent=new
Intent(VideoActivity.this,VideoActivity.class);
                            startActivity(intent);
//    Toast.makeText(VideoActivity.this, "Please Wait File is Being Encrypted",
Toast.LENGTH_SHORT).show();
}
                    },3000);

                }

            }
        });
        ui();
    }

public boolean checkNamesOnServer(String name){
boolean flag = false;
for(int i = 0; i <videoListServer.length; i++){
//System.out.println("FileName: "+name);
            //System.out.println("FileNameServer: "+imageListServer[i]);
if(name.equals(videoListServer[i])){
                flag = true;
            }
        }
        System.out.println(flag);
return flag;
    }
public boolean checkNamesOnPhone(String name){
boolean flag = false;
for(int i = 0; i <fileNameStrings.length; i++){
//System.out.println("FileName: "+name);
            // System.out.println("FileNameServer: "+fileNameStrings[i]);
if(name.equals(fileNameStrings[i])){
                flag = true;
            }
        }

return flag;
    }

public void getImageNamesFromDatabase(final int choice){
```

```
databaseReference.addListenerForSingleValueEvent(new ValueEventListener() {
@Override
public void onDataChange(DataSnapshot dataSnapshot) {
videoListServer = new String[(int)dataSnapshot.getChildrenCount()];
int i = 0;
for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
//ImageUpload class require default constructor
VideoUpload vid = snapshot.getValue(VideoUpload.class);
videoListServer[i] = vid.getName();
//Toast.makeText(GalleryActivity.this,
imageListServer[i]+"Length"+imageListServer.length, Toast.LENGTH_SHORT).show();
i++;
                }
                i = 0;
                System.out.println("GOGOGO");
if(choice == 1){
for(int j = 0; j <fileUriArray.length; j++){

boolean isPresentOnServer = checkNamesOnServer(fileNameStrings[j]);
if(isPresentOnServer == false){
                            System.out.println("Not Present: "+fileNameStrings[j]);
                            uploadVideo(fileNameStrings[j], fileUriArray[j]);
                        }
                    }
                }
if(choice == 2){
for(int j = 0; j <videoListServer.length; j++){

boolean isPresentOnPhone = checkNamesOnPhone(videoListServer[j]);
if(isPresentOnPhone == false){
                            System.out.println("Not Present: "+videoListServer[j]);
try {
                                downloadVideo(videoListServer[j]);
                            } catch (IOException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }
            }

@Override
public void onCancelled(DatabaseError databaseError) {

            }
        });


    }

private void uploadVideo(final String fileName, final Uri filePath) {
if(filePath != null)
        {

progressDialog.setTitle("Uploading...");
progressDialog.show();

            StorageReference ref = storageReference.child("Encrypted_Videos/"+
fileName);
            ref.putFile(filePath)
                    .addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
@Override
```

```
public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                            VideoUpload videoUpload = new VideoUpload(fileName,
filePath.toString());
                            String videoId=databaseReference.push().getKey();
databaseReference.child(videoId).setValue(videoUpload);

progressDialog.dismiss();
                            Toast.makeText(VideoActivity.this, "Uploaded",
Toast.LENGTH_SHORT).show();
                        }
                    })
                    .addOnFailureListener(new OnFailureListener() {
@Override
public void onFailure(@NonNull Exception e) {
progressDialog.dismiss();
                            Toast.makeText(VideoActivity.this, "Failed
"+e.getMessage(), Toast.LENGTH_SHORT).show();
                        }
                    })
                    .addOnProgressListener(new
OnProgressListener<UploadTask.TaskSnapshot>() {
@Override
public void onProgress(UploadTask.TaskSnapshot taskSnapshot) {
double progress = (100.0*taskSnapshot.getBytesTransferred()/taskSnapshot
                                    .getTotalByteCount());
progressDialog.setMessage("Uploaded "+(int)progress+"%");
                        }
                    });
        }
    }
private void downloadVideo(final String fileName) throws IOException {
progressDialog.setTitle("Downloading...");
progressDialog.show();
        StorageReference ref = storageReference.child("Encrypted_Videos/"+ fileName);
final File storagePath = new File("/storage/emulated/0/EncryptedVideos/"+fileName);
//final File localFile = File.createTempFile("sam", null , storagePath);
ref.getFile(storagePath).addOnSuccessListener(new
OnSuccessListener<FileDownloadTask.TaskSnapshot>() {
@Override
public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot) {
progressDialog.dismiss();
                Toast.makeText(VideoActivity.this, "Downloaded: "+fileName,
Toast.LENGTH_SHORT).show();
            }
        }).addOnFailureListener(new OnFailureListener() {
@Override
public void onFailure(@NonNull Exception e) {
progressDialog.dismiss();
                Toast.makeText(VideoActivity.this, "Failed "+e.getMessage(),
Toast.LENGTH_SHORT).show();
            }
        }).addOnProgressListener(new
OnProgressListener<FileDownloadTask.TaskSnapshot>() {
@Override
public void onProgress(FileDownloadTask.TaskSnapshot taskSnapshot) {
double progress = (100.0*taskSnapshot.getBytesTransferred()/taskSnapshot
                        .getTotalByteCount());
progressDialog.setMessage("Downloaded "+(int)progress+"%");
            }
        });
    }
```

```
public void ui(){
new Thread(new Runnable() {
@Override
public void run() {
                runOnUiThread(new Runnable() {
@Override
public void run() {
lv = (ListView)findViewById(R.id.listView2);
listAdapter = new MyListAdapterVideos(VideoActivity.this, filePathStrings,
fileNameStrings);
lv.setAdapter(listAdapter);
                    }
                });
            }
        }).start();

    }

@Override
protected void onActivityResult(int reqCode, int resCode, Intent data) {

if(resCode == Activity.RESULT_OK && data != null){
if(data.getClipData() == null){
realPath = new String[1];
// SDK < API11
if (Build.VERSION.SDK_INT <11)
realPath[0] = RealPathUtil.getRealPathFromURI_BelowAPI11(this, data.getData());

// SDK >= 11 && SDK < 19
else if (Build.VERSION.SDK_INT <19)
realPath[0] = RealPathUtil.getRealPathFromURI_API11to18(this, data.getData());

// SDK > 19 (Android 4.4)
else
realPath[0] = RealPathUtil.getRealPathFromURI_API19(this, data.getData());
            }
else{
realPath = new String[data.getClipData().getItemCount()];
for(int i = 0; i < data.getClipData().getItemCount(); i++) {
// SDK < API11
if (Build.VERSION.SDK_INT <11)
realPath[i] = RealPathUtil.getRealPathFromURI_BelowAPI11(this,
data.getClipData().getItemAt(i).getUri());

// SDK >= 11 && SDK < 19
else if (Build.VERSION.SDK_INT <19)
realPath[i] = RealPathUtil.getRealPathFromURI_API11to18(this,
data.getClipData().getItemAt(i).getUri());

// SDK > 19 (Android 4.4)
else
realPath[i] = RealPathUtil.getRealPathFromURI_API19(this,
data.getClipData().getItemAt(i).getUri());
                }
            }



//setTextViews(Build.VERSION.SDK_INT, data.getData().getPath(),realPath);
            //enc();

}
```

```
    }
public void enc3(String[] pathIn){
for(int i=0; i<pathIn.length;i++){
            File file1 = new File(pathIn[i]);
            String filename = file1.getName();
try{
                initialize(pathIn[i],
"/storage/emulated/0/EncryptedVideos/"+filename);
                encrypt();
//initialize("/storage/emulated/0/Encrypted/abcde.jpg",
"/storage/emulated/0/Decrypted/de.jpg");
                //encrypt();
}
catch(FileNotFoundException e){
                System.out.println(e.getMessage()+"NOTFOUNFEX");

            }
catch(IOException e){
                System.out.println(e.getMessage()+"IOEX");
            }
        }

    }
public void initialize(String inputFilePath, String outputFilePath) throws
FileNotFoundException{
reader = new DataInputStream(new FileInputStream(new File(inputFilePath)));
writer = new DataOutputStream(new FileOutputStream(new File(outputFilePath)));
    }
public void encrypt() throws IOException {
int readBytes = 0;
        RC4Cipher rc4Encryption = new RC4Cipher("123abcde");
byte[] buffer = new byte[2048];
do {
            readBytes = reader.read(buffer, 0, 2048);
            buffer = rc4Encryption.rc4(buffer);
if (readBytes >0) {
writer.write(buffer, 0, readBytes);
writer.flush();
            }
//System.out.println(k++);

} while (readBytes >0);
    }

}
```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {


    FirebaseAuth auth;
    EditText email;
    EditText password;
    Button signup,login,offline,other,lockapps;
    String pin;
    TextView forgetpassword;

@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
        getWindow().requestFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);

        SharedPreferences settings = getSharedPreferences("PREFS", 0);
pin = settings.getString("pin", "");

signup = (Button)findViewById(R.id.Signup);
lockapps = (Button)findViewById(R.id.lockapps);
forgetpassword=(TextView)findViewById(R.id.forgetpassword);
signup.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
                Intent i = new Intent(MainActivity.this, SignupActivity.class);
                startActivity(i);
            }
        });
lockapps.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
                Intent i = new Intent(MainActivity.this, LockSplashActivity.class);
                startActivity(i);
            }
        });
forgetpassword.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
                Intent intentforget=new
Intent(MainActivity.this,ForgetPassword.class);
                startActivity(intentforget);

            }
        });

email = (EditText)findViewById(R.id.Email);
password = (EditText)findViewById(R.id.Password);
auth = FirebaseAuth.getInstance();


//final LottieAnimationView animationView =
(LottieAnimationView)findViewById(R.id.animation_view);
        /*animationView.addAnimatorListener(new Animator.AnimatorListener() {
            @Override
            public void onAnimationStart(Animator animator) {

            }

            @Override
            public void onAnimationEnd(Animator animator) {
                Intent i = new Intent(SecondMainActivity.this, MenuActivity.class);
                startActivity(i);
            }
            @Override
            public void onAnimationCancel(Animator animator) {

            }
            @Override
            public void onAnimationRepeat(Animator animator) {

            }
        });*/
```

```
offline = (Button)findViewById(R.id.OfflineButton);
offline.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
if(pin.equals("")){
                    Intent i = new Intent(MainActivity.this, CreatePinActivity.class);
                    startActivity(i);
                    finish();
                }
else{
                    Intent i = new Intent(MainActivity.this, EnterPinActivity.class);
                    startActivity(i);
                    finish();
                }
            }
        });
login=(Button)findViewById(R.id.Login);
login.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view){


                String em = email.getText().toString();
final String pass = password.getText().toString();

if (TextUtils.isEmpty(em)) {
                    Toast.makeText(getApplicationContext(), "Enter email address!",
Toast.LENGTH_SHORT).show();
return;
                }

if (TextUtils.isEmpty(pass)) {
                    Toast.makeText(getApplicationContext(), "Enter password!",
Toast.LENGTH_SHORT).show();
return;
                }


auth.signInWithEmailAndPassword(em, pass).addOnCompleteListener(MainActivity.this, new
OnCompleteListener<AuthResult>() {
@Override
public void onComplete(@NonNull Task<AuthResult> task) {
if(!task.isSuccessful()){
if (password.length() <6) {
password.setError("Short Password");
                            } else {
                                Toast.makeText(MainActivity.this, "Authorization
Failed", Toast.LENGTH_LONG).show();
                            }
                        }
else{
                            Intent i = new Intent(MainActivity.this,
MenuActivity.class);
                            startActivity(i);
                            finish();
                        }
                    }
                });


            }
        });
```

```
    }
}
```

Gallery Activity.java

```
public class GalleryActivity extends AppCompatActivity {
    String realPath[];
    TextView txtSDK;
    Button uploadImage, downloadImage;
    TextView txtUriPath,txtRealPath;
    ImageView imageView;
    DataInputStream reader;
    DataOutputStream writer;
    String path;
private Uri[] fileUriArray;
    String filePathUriToString;
    ProgressDialog progressDialog;

    ListView lv;
    MyListAdapter listAdapter;


private String[] imageListServer;


    File file;
    File[] listFile;
    String[] filePathStrings;
    String[] fileNameStrings;
    StorageReference storageReference;
    DatabaseReference databaseReference;
final String databasePathImages="Encrypted_Images";
    String[] countryNames = {"Greece","Spain"};
int[] countryFlags = {R.drawable.flag_1, R.drawable.flag_2};
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_gallery);

        StrictMode.VmPolicy.Builder builder = new StrictMode.VmPolicy.Builder();
        StrictMode.setVmPolicy(builder.build());
storageReference = FirebaseStorage.getInstance().getReference();
databaseReference= FirebaseDatabase.getInstance().getReference(databasePathImages);
progressDialog=new ProgressDialog(GalleryActivity.this);


        Button add = (Button) findViewById(R.id.add);
        add.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
                Intent i = new Intent(Intent.ACTION_PICK);
                i.setType("image/*");
                i.putExtra(Intent.EXTRA_ALLOW_MULTIPLE, true);
//i.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(i, 1);

            }
        });
uploadImage = (Button) findViewById(R.id.uploadImages);
uploadImage.setOnClickListener(new View.OnClickListener() {
```

```
@Override
public void onClick(View view) {
                getImageNamesFromDatabase(1);


            }
        });
downloadImage = (Button)findViewById(R.id.downloadImages);
downloadImage.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
                getImageNamesFromDatabase(2);
            }
        });

file = new File(Environment.getExternalStorageDirectory() + File.separator +
"EncryptedImages");
file.mkdirs();
if (file.isDirectory()) {
listFile = file.listFiles();

if(listFile == null){
filePathStrings = new String[0];
fileNameStrings = new String[0];
fileUriArray = new Uri[0];

for (int i = 0; i <0; i++) {
filePathStrings[i] = listFile[i].getAbsolutePath();
fileNameStrings[i] = listFile[i].getName();
fileUriArray[i] = Uri.fromFile(listFile[i]);
                }
            }
else {
filePathStrings = new String[listFile.length];
fileNameStrings = new String[listFile.length];
fileUriArray = new Uri[listFile.length];

for (int i = 0; i <listFile.length; i++) {
filePathStrings[i] = listFile[i].getAbsolutePath();
fileNameStrings[i] = listFile[i].getName();
fileUriArray[i] = Uri.fromFile(listFile[i]);
                }
            }
        }


//String[] empty1 = {};
        //myTask mt = new myTask();
        //mt.execute();

        //lv = (ListView)findViewById(R.id.listView1);
        //listAdapter = new MyListAdapter(GalleryActivity.this, filePathStrings,
fileNameStrings);
        //lv.setAdapter(listAdapter);
        //generateListContent();
        //lv.setAdapter(new MyListAdapter(this, R.layout.listview_items, data));

Button encrypt = (Button) findViewById(R.id.enc);
        encrypt.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
if(realPath[0] == null){
                    Toast.makeText(GalleryActivity.this, "please choose first",
```

```
Toast.LENGTH_SHORT).show();
                }
else{
                    enc3(realPath);
//Toast.makeText(GalleryActivity.this, realPath[0]+"", Toast.LENGTH_SHORT).show();
File a = new File(realPath[0]);
                    a.delete();
                    Intent scanIntent = new
Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
                    scanIntent.setData(Uri.fromFile(a));
                    sendBroadcast(scanIntent);

final ProgressDialog progressDialog=new ProgressDialog(GalleryActivity.this);
                    progressDialog.setTitle("Encrypting..");
                    progressDialog.setMessage("Please wait");
                    progressDialog.show();

final Handler handler = new Handler();
                    handler.postDelayed(new Runnable() {
@Override
public void run() {
progressDialog.hide();
                            Intent intent=new
Intent(GalleryActivity.this,GalleryActivity.class);
                            startActivity(intent);
                                                    }
                    },3000);
                }
            }
        });
        ui();
    }

public boolean checkNamesOnServer(String name){
boolean flag = false;
for(int i = 0; i <imageListServer.length; i++){
//System.out.println("FileName: "+name);
            //System.out.println("FileNameServer: "+imageListServer[i]);
if(name.equals(imageListServer[i])){
                flag = true;
            }
        }
        System.out.println(flag);
return flag;
    }
public boolean checkNamesOnPhone(String name){
boolean flag = false;
for(int i = 0; i <fileNameStrings.length; i++){
//System.out.println("FileName: "+name);
           // System.out.println("FileNameServer: "+fileNameStrings[i]);
if(name.equals(fileNameStrings[i])){
                flag = true;
            }
        }

return flag;
    }

public void getImageNamesFromDatabase(final int choice){
databaseReference.addListenerForSingleValueEvent(new ValueEventListener() {
@Override
public void onDataChange(DataSnapshot dataSnapshot) {
imageListServer = new String[(int)dataSnapshot.getChildrenCount()];
```

```
int i = 0;
for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
//ImageUpload class require default constructor
ImageUpload img = snapshot.getValue(ImageUpload.class);
imageListServer[i] = img.getName();
//Toast.makeText(GalleryActivity.this,
imageListServer[i]+"Length"+imageListServer.length, Toast.LENGTH_SHORT).show();
i++;
                }
                i = 0;
                System.out.println("GOGOGO");
if(choice == 1){
for(int j = 0; j <fileUriArray.length; j++){

boolean isPresentOnServer = checkNamesOnServer(fileNameStrings[j]);
if(isPresentOnServer == false){
                            System.out.println("Not Present: "+fileNameStrings[j]);
                            uploadImage(fileNameStrings[j], fileUriArray[j]);
                        }
                    }
                }
if(choice == 2){
for(int j = 0; j <imageListServer.length; j++){

boolean isPresentOnPhone = checkNamesOnPhone(imageListServer[j]);
if(isPresentOnPhone == false){
                            System.out.println("Not Present: "+imageListServer[j]);
try {
                                downloadImage(imageListServer[j]);
                            } catch (IOException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }
            }

@Override
public void onCancelled(DatabaseError databaseError) {

            }
        });


    }

public void ui(){
new Thread(new Runnable() {
@Override
public void run() {
                runOnUiThread(new Runnable() {
@Override
public void run() {
lv = (ListView)findViewById(R.id.listView1);
listAdapter = new MyListAdapter(GalleryActivity.this, filePathStrings,
fileNameStrings);
lv.setAdapter(listAdapter);
                    }
                });
            }
        }).start();

    }
```

```
public void generateListContent(){
for(int i=0; i<55; i++){
//data.add("This is row number "+i);
}
    }
/*@Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode == RESULT_OK) {
            Uri photoUri = data.getData();
            if (photoUri != null) {
                path = photoUri.toString();
            }
        }
    }*/
@Override
protected void onActivityResult(int reqCode, int resCode, Intent data) {

if(resCode == Activity.RESULT_OK && data != null){
if(data.getClipData() == null){
realPath = new String[1];
// SDK < API11
if (Build.VERSION.SDK_INT <11)
realPath[0] = RealPathUtil.getRealPathFromURI_BelowAPI11(this, data.getData());

// SDK >= 11 && SDK < 19
else if (Build.VERSION.SDK_INT <19)
realPath[0] = RealPathUtil.getRealPathFromURI_API11to18(this, data.getData());

// SDK > 19 (Android 4.4)
else
realPath[0] = RealPathUtil.getRealPathFromURI_API19(this, data.getData());
            }
else{
realPath = new String[data.getClipData().getItemCount()];
for(int i = 0; i < data.getClipData().getItemCount(); i++) {
// SDK < API11
if (Build.VERSION.SDK_INT <11)
realPath[i] = RealPathUtil.getRealPathFromURI_BelowAPI11(this,
data.getClipData().getItemAt(i).getUri());

// SDK >= 11 && SDK < 19
else if (Build.VERSION.SDK_INT <19)
realPath[i] = RealPathUtil.getRealPathFromURI_API11to18(this,
data.getClipData().getItemAt(i).getUri());

// SDK > 19 (Android 4.4)
else
realPath[i] = RealPathUtil.getRealPathFromURI_API19(this,
data.getClipData().getItemAt(i).getUri());
                }
            }

//setTextViews(Build.VERSION.SDK_INT, data.getData().getPath(),realPath);
            //enc();

}

    }
private void uploadImage(final String fileName, final Uri filePath) {
if(filePath != null)
```

```
        {

progressDialog.setTitle("Uploading...");
progressDialog.show();

            StorageReference ref = storageReference.child("Encrypted_Images/"+
fileName);
            ref.putFile(filePath)
                    .addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
@Override
public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                            ImageUpload imageUpload = new ImageUpload(fileName,
filePath.toString());
                            String imageId=databaseReference.push().getKey();
databaseReference.child(imageId).setValue(imageUpload);

progressDialog.dismiss();
                            Toast.makeText(GalleryActivity.this, "Uploaded",
Toast.LENGTH_SHORT).show();
                        }
                    })
                    .addOnFailureListener(new OnFailureListener() {
@Override
public void onFailure(@NonNull Exception e) {
progressDialog.dismiss();
                            Toast.makeText(GalleryActivity.this, "Failed
"+e.getMessage(), Toast.LENGTH_SHORT).show();
                        }
                    })
                    .addOnProgressListener(new
OnProgressListener<UploadTask.TaskSnapshot>() {
@Override
public void onProgress(UploadTask.TaskSnapshot taskSnapshot) {
double progress = (100.0*taskSnapshot.getBytesTransferred()/taskSnapshot
                                    .getTotalByteCount());
progressDialog.setMessage("Uploaded "+(int)progress+"%");
                        }
                    });
        }
    }
private void downloadImage(final String fileName) throws IOException {
progressDialog.setTitle("Downloading...");
progressDialog.show();
        StorageReference ref = storageReference.child("Encrypted_Images/"+ fileName);
final File storagePath = new File("/storage/emulated/0/EncryptedImages/"+fileName);
//final File localFile = File.createTempFile("sam", null , storagePath);
ref.getFile(storagePath).addOnSuccessListener(new
OnSuccessListener<FileDownloadTask.TaskSnapshot>() {
@Override
public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot) {
progressDialog.dismiss();
                Toast.makeText(GalleryActivity.this, "Downloaded: "+fileName,
Toast.LENGTH_SHORT).show();
            }
        }).addOnFailureListener(new OnFailureListener() {
@Override
public void onFailure(@NonNull Exception e) {
progressDialog.dismiss();
                Toast.makeText(GalleryActivity.this, "Failed "+e.getMessage(),
Toast.LENGTH_SHORT).show();
            }
        }).addOnProgressListener(new
```

```
OnProgressListener<FileDownloadTask.TaskSnapshot>() {
@Override
public void onProgress(FileDownloadTask.TaskSnapshot taskSnapshot) {
double progress = (100.0*taskSnapshot.getBytesTransferred()/taskSnapshot
                        .getTotalByteCount());
progressDialog.setMessage("Downloaded "+(int)progress+"%");
            }
        });
    }

public void enc3(String[] pathIn){
for(int i=0; i<pathIn.length;i++){
            File file1 = new File(pathIn[i]);
            String filename = file1.getName();
try{
                initialize(pathIn[i],
"/storage/emulated/0/EncryptedImages/"+filename);
                encrypt();
//initialize("/storage/emulated/0/Encrypted/abcde.jpg",
"/storage/emulated/0/Decrypted/de.jpg");
                //encrypt();
}
catch(FileNotFoundException e){
                System.out.println(e.getMessage()+"NOTFOUNFEX");

            }
catch(IOException e){
                System.out.println(e.getMessage()+"IOEX");
            }
        }

    }
public void initialize(String inputFilePath, String outputFilePath) throws
FileNotFoundException{
reader = new DataInputStream(new FileInputStream(new File(inputFilePath)));
writer = new DataOutputStream(new FileOutputStream(new File(outputFilePath)));
    }
public void encrypt() throws IOException {
int readBytes = 0;
        RC4Cipher rc4Encryption = new RC4Cipher("123abcde");
byte[] buffer = new byte[2048];
do {
            readBytes = reader.read(buffer, 0, 2048);
            buffer = rc4Encryption.rc4(buffer);
if (readBytes >0) {
writer.write(buffer, 0, readBytes);
writer.flush();
            }
//System.out.println(k++);

} while (readBytes >0);
    }
public void abctask(){
lv = (ListView)findViewById(R.id.listView1);
listAdapter = new MyListAdapter(GalleryActivity.this, filePathStrings,
fileNameStrings);
lv.setAdapter(listAdapter);
    }


}
```