Implementation of an Android Application to Help users Recover and/or Secure a Lost Phone

GRADUATE PROJECT REPORT

Submitted to the Faculty of
The School of Engineering & Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Haritha Devulapally
Summer 2014

Committee Members

Dr. Mario A. Garcia
Committee Chairperson

Dr. Scott A. King
Committee Member
ABSTRACT

Smart phones are mobile phones with several advanced features. They are equivalent to pocket computers. Smart phones carry private data like personal and business contacts, messages, pictures, banking information etc. There are several security issues related to smart phones. Due to the ubiquity of smart phones, providing means to enhance the security becomes a crucial factor. It is important to have control over the smart phone even when users are away from the phone and hence be able to access it remotely. This prevents the danger of misuse of the sensitive information stored on the phone by unauthorized individuals to a good extent.

As a part of controlling the phone remotely in cases where the phone is misplaced or lost, this application allows the user to lock the phone by sending a text message containing the secret code from another phone. The lock code of the phone can be changed and the logs can be erased. Also, the profile settings on the phone such as ring tone status can be changed from silent to vibrate or general and vice versa. This is useful in recovering a misplaced phone etc. When an unauthorized user tries to hack the secret code by sending many messages, the user will be blocked after a certain number of failed attempts and if the genuine owner of the phone forgets the code, he can recover it by sending a message.

The application allows only the authorized user of the phone to change the settings related to the secret codes and remote assist features. This can be ensured by making the settings screen viewable only when a predefined number is dialed on the phone.
TABLE OF CONTENTS

Abstract ................................................................................................................................. ii

Table of Contents .................................................................................................................. iii

List of Figures ........................................................................................................................ vi

List of Tables .......................................................................................................................... ix

1. Background and Rationale ................................................................................................. 1
   1.1 Introduction ....................................................................................................................... 2
   1.2 Prior Work ......................................................................................................................... 3
      1.2.1 Android Device Manager ......................................................................................... 4
      1.2.2 Locate my droid ........................................................................................................ 4
      1.2.3 TrackMe .................................................................................................................... 4
   1.3 Solution ............................................................................................................................. 6

2. Remote Assist App .............................................................................................................. 8
   2.1 Problem Statement ............................................................................................................ 8
   2.2 Motivation ......................................................................................................................... 8
   2.3 System Description ......................................................................................................... 9
   2.4 Project Scope ................................................................................................................... 9
      2.4.1 System Architecture ............................................................................................... 10
   2.5 User Interface ................................................................................................................ 11

3. System Design .................................................................................................................... 15
   3.1 Design Rationale ............................................................................................................ 15
   3.2 Object Oriented Analysis & Design .............................................................................. 17
      3.2.1 Use Case Diagrams ............................................................................................... 17
3.2.2 Sequence Diagram.................................................................25
3.2.3 Class Diagram.........................................................................26
4. Project Implementation........................................................................31
  4.1 System Flow...................................................................................31
  4.2 Software and Hardware requirements........................................33
  4.3 Functional Requirements..........................................................33
  4.4 Android Application.......................................................................34
    4.4.1 Application Connectivity....................................................34
    4.4.2 Launch the Application.......................................................34
    4.4.3 Enable Remote Assist and Device Admin.............................37
    4.4.4 Set Secret Code and Security Question.................................39
    4.4.5 Lock the Phone.......................................................................41
    4.4.6 Change Profile Settings.......................................................43
    4.4.7 Erase call logs.......................................................................47
    4.4.8 Block Sender..........................................................................48
    4.4.9 Recover Forgotten Secret Code..............................................50
    4.4.10 Delete Text Messages..........................................................52
    4.4.11 Retrieve GPS Co-ordinates..................................................53
    4.4.12 Delete Accounts...................................................................54
    4.4.13 Factory Reset.........................................................................55
    4.4.14 Master Command...................................................................55
5. Evaluation and Expected Results....................................................56
  5.1 Compatibility..................................................................................56
5.2 Security........................................................................................................56
5.3 Portability......................................................................................................56
5.4 Reliability.....................................................................................................56
5.5 Testing...........................................................................................................57
  5.5.1 Launch the App Phase...............................................................................57
  5.5.2 Remote Assist Phase................................................................................59
  5.5.3 Secret Code Phase....................................................................................60

6. Conclusion and Future Work.............................................................................62
  6.1 Conclusion....................................................................................................62
  6.2 Future Work..................................................................................................63

References...........................................................................................................64
LIST OF FIGURES

Figure 2.1 System Level Architecture .................................................................10
Figure 2.2 User Settings Activity ..........................................................................12
Figure 2.3 Overflow Menu ......................................................................................13
Figure 2.4 About Activity .......................................................................................13
Figure 2.5 Instruction Activity ................................................................................14
Figure 3.1 Programming Level Design .....................................................................15
Figure 3.2 Codes to Perform Certain Actions ..........................................................16
Figure 3.3 Use Case Diagram for RemoteAssist Application ....................................18
Figure 3.4 Use Case Diagram to Launch RemoteAssist app .....................................19
Figure 3.5 Use Case Diagram to Manage Settings ...................................................20
Figure 3.6 Use Case Diagram to Lock the Phone .....................................................21
Figure 3.7 Use Case Diagram to Change the Profile Settings ....................................22
Figure 3.8 Use Case Diagram to Erase Call Logs ....................................................23
Figure 3.9 Use Case Diagram to Block and Unblock Sender ....................................24
Figure 3.10 Use Case Diagram to Retrieve the Secret Code .....................................25
Figure 3.11 Sequence Diagram for RemoteAssist Application .................................26
Figure 3.12 Class Diagram for RemoteAssist Application .......................................27
Figure 4.1 System Flow Diagram ............................................................................32
Figure 4.2 Code Snippet for OutgoingCallReceiver ...............................................35
Figure 4.3 Code to Launch the Application ..............................................................36
Figure 4.4 Settings Screen .......................................................................................36
Figure 4.5 Code Snippet for Remote Assist and Device Admin .................................37
Figure 4.6 Enable Remote Assist..................................................................................38
Figure 4.7 Enable Device Admin..................................................................................38
Figure 4.8 Set Secret Code..........................................................................................39
Figure 4.9 Set Security Question..................................................................................40
Figure 4.10 Set Security Answer..................................................................................40
Figure 4.11 Code Snippet to Lock the Phone...............................................................41
Figure 4.12 SMS in Desired Format to Lock the Phone.................................................42
Figure 4.13 Android Phone Locked with Secret Code..................................................42
Figure 4.14 SMS in Desired Format to Change to Ring Mode......................................43
Figure 4.15 Code Snippet to Change the Phone to Ring Mode......................................44
Figure 4.16 SMS in Desired Format to Change to Vibrate Mode.................................45
Figure 4.17 Code Snippet to Change the Phone to Vibrate Mode...............................45
Figure 4.18 SMS in Desired Format to Change to Silent Mode....................................46
Figure 4.19 Code Snippet to Change the Phone Mode to Silent....................................47
Figure 4.20 Erase Call Logs........................................................................................47
Figure 4.21 Code Snippet to Clear Call Logs...............................................................48
Figure 4.22 Attempts to Access Phone with Invalid Codes.........................................48
Figure 4.23 Blocked Sender without any Acknowledgement.......................................49
Figure 4.24 Code Snippet to Block Sender.................................................................49
Figure 4.25 Recover Secret Code................................................................................50
Figure 4.26 Security Question and Answer.................................................................51
Figure 4.27 Code Snippet to Recover Secret Code......................................................52
Figure 4.28 Code Snippet to Delete Text Messages......................................................53
Figure 4.29 Code Snippet to Retrieve GPS Co-ordinates...................................................54
Figure 4.30 Code Snippet to Delete Accounts.................................................................54
Figure 4.31 Code Snippet to Perform Factory Reset..........................................................55
Figure 5.1(a) Successful Launch.......................................................................................58
Figure 5.1(b) Unsuccessful Launch....................................................................................58
Figure 5.2(a) Valid Code....................................................................................................60
Figure 5.2(b) Invalid Code................................................................................................60
Figure 5.3(a) Successful Recovery.....................................................................................61
Figure 5.3(b) Unsuccessful Recovery with Incorrect Command.......................................62
Figure 5.3(c) Unsuccessful Recovery with Incorrect Security Answer...............................62
LIST OF TABLES

Table 1.1 Types of Data and their Sensitivity Value from User Perspective.......................5
Table 3.1 Use Case description for RemoteAssist application........................................18
Table 3.2 Use Case description to Launch RemoteAssist app.......................................19
Table 3.3 Use Case description to Manage Settings.....................................................20
Table 3.4 Use Case description to Lock the Phone.......................................................21
Table 3.5 Use Case description to Change the Profile Settings.......................................22
Table 3.6 Use Case description to Erase Call Logs..........................................................23
Table 3.7 Use Case description to Block and Unblock Senders.......................................24
Table 3.8 Use Case description to Retrieve the Secret Code...........................................25
1. BACKGROUND AND RATIONALE

Smart phones have become an integral part of our lives. A smart phone is a handheld mobile device that not only allows you to make phone calls like a standard mobile phone but also performs many added functions similar to those of Personal Digital Assistants and computers such as sending and receiving emails, capturing and storing images, editing documents in Office, GPS navigation and also browsing the internet, etc. The history of smart phones dates back to the 1990's when the first smart phone was introduced by IBM in 1992 called Simon (ITLog 2014). In 1993, it was released to the public and was marketed by Bell South. Some of its features were an inbuilt calendar, calculator, email, world clock, address book and games and also included a QWERTY keyboard. Ericsson introduced a smart phone called R380 in 2000 (ITLog 2014), but it could not run any native 3rd party applications. HTC introduced its first smart phone in 2002 along with Palm Treo. Sony Ericsson introduced the P800 in 2002 that contained an MP3 player, touch screen and a built-in camera. Apple introduced the first smart phone in 2007 with HD touch screen, fast browsing internet on the go and the ability to download numerous apps.

According to a study conducted by eMarketer, the number of smart phone users all over the world surpassed 1 billion in 2012 and is expected to come to 1.75 billion by the end of 2014 and also continue its fast-paced trajectory growth through 2017 (eMarketer 2014). According to the research forecast, almost one quarter of the entire population in the world will use a smart phone at least on a monthly basis. According to a study conducted by Pew Research Center’s Internet & American Life Project, about 56% of Americans use a smart phone (Smith 2013). The percentage of people using smart phones for shopping and retail purposes increased to a great extent. Four out of five people who use smart phones use it for shopping (Rao 2012). A study conducted by TNS shows that among the 6 billion mobile users around the world, 19% of them
use location based services on the mobile and 62% of them aspire to do so (Walsh 2012). Navigation via GPS and maps, finding entertainment and dining venues and sharing location on applications like Facebook and Foursquare were some of the main uses of location based services.

There are six major smart phone operating systems, namely, iOS, Android, BlackBerry OS, Windows phone, Symbian and web OS. Mobile operating systems had to grow in sophistication to accommodate the features of smart phones such as large storage space, high performance CPUs and GPUs, high resolution screens and multitasking capabilities. The need to allow external developers to write software for mobile devices is another important factor.

Android is one of the leading developers of smart phones and was introduced as a competitor to the Apple iOS. Android uses the Linux kernel. Android has many advantages such as multitasking, ease of notification, easy access to numerous apps on the Google Play, diverse phone options, easy installation of modified ROM, widgets etc. Android is open source which makes it the most developer friendly platform to work on especially for beginners.

1.1 Introduction

Smart phone usage has increased significantly over time and they have become an important aspect of our lives. This brings about the concern of security on these devices. Most of the smart phones contain private data such as personal images and videos, business contacts and calendars regarding important time and schedule of activities, banking information such as saved credit card numbers, user names and passwords, personal text or instant messages etc. The data on these devices needs to be secured at all times or else it will lead to violation of privacy and
misuse of information that might cause severe damage to an individual or in some cases even a group or a company.

Since smart phones are carried around everywhere, it is quite common for them to be lost or stolen. According to Becher, M. and Freiling, F.C "The forensic analysis of mobile devices is an attack vector targeting the confidentiality of the stored data". They also mentioned that "It is an unexpected attack vector and it is only valid in the case of an attacker getting physical access to the device" (Michael Becher 2011). This type of attack is possible in two situations- when the phone is taken for a limited period of time from the owner without notice, that is, the phone is lost or stolen or when there is a legitimate change of ownership.

According to a study conducted by Symantec, out of all the people who found lost Android phones, 89% of them tried to access sensitive information (Templet 2012). There is a need to provide security to data on the phones so that it cannot be misused by unauthorized individuals. Therefore a security application which allows the user to lock the phone remotely helps in providing security to some extent.

1.2 Prior work

Many applications are being developed in order to protect the data on the smart phones as a result of increase in the number of lost or misplaced phones every year. A study indicates that among the total number of smart phone users, about 54% of them do not use passwords or lock codes to protect their phones. Smart phone users need to be cautious and should have proper means to recover or at least secure the data if the phone is lost. Some of the applications that serve this purpose are Android device manager, Android Lost, Find my phone, etc. (GooglePlay, https://play.google.com/store/apps/details?id=com.locatemy&hl=en 2012)
1.2.1 Android device manager

This application allows the user to locate an android device if it is lost or misplaced. It allows to reset screen lock pin and also erase the data on the phone. But in order to use this application, the user needs to have an active Google account associated with a Gmail account. Also, the application has to be installed on the device from which you are searching the lost phone. The major problem with this application is that when the phone is stolen, the bad guy can access the app and deactivate the Google account to which it is being synced (GooglePlay, https://play.google.com/store/apps/details?id=com.google.android.apps.adm&hl=en 2014).

1.2.2 Locate my Droid

This application enables the user to locate a lost or stolen phone and it also gives an alert tone. The user has to activate the application on the phone after installing it. The phone can be tracked on a map by logging on to the website www.locatemydroid.com. The drawback of this application is the use of a third party web server (GooglePlay, https://play.google.com/store/apps/details?id=com.locatemy&hl=en 2012)

1.2.3 TrackMe

This application allows the user to remotely access the phone when it is lost or stolen. The owner of the phone can track the GPS co-ordinates of the phone by sending a text message to the phone from any other phone. It is also possible to retrieve the most recent call logs and text messages on the phone. With the help of this application, the user can delete the call logs and also, when the sim card is changed, an alert message is sent to the owner of the phone (Adusumalli 2014).
According to a study conducted by Muslukhov, I. and Boshmaf, Y., data is divided mainly into two classes based on protection requirements. They are: - valuable data and sensitive data. The data on smart phones is further categorized into 11 types as shown in Table 1.1 (Y. B. Ildar Muslukhov 2012). The fully filled circles represent the data that was considered sensitive or valuable most of the time, the partially filled circles represent the data considered sensitive or valuable by a minority of people whereas the empty circles represent the data that was not considered sensitive or valuable by all. The purpose of this study was to approach the smart phone's data security problem in a user centric way and analyze the requirements of data protection systems from user's point of view.

Table 1.1 Types of Data and their Sensitivity Value from User's Perspective (Y. B. Ildar Muslukhov 2012)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Sensitive</th>
<th>Valuable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS Messages</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Photos/Videos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voice Recordings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contacts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Music</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Passwords</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Emails</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documents</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Events in Calendar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recorded GPS Tracks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Therefore, smart phones when lost or stolen pose a serious threat to the sensitive or valuable data stored on them. According to Landman, M., CIO.com reported 31,544 smart phones that have been left in New York City taxis in June 2010. London taxi drivers reported over 60,000 mobile phones along with 5,500 PDAs and 4,500 laptops were left in their cabs over a six month period. This shows that due to the small size of these smart phone devices, they are prone to be misplaced easily (Landman 2010).
1.3 Solution

The purpose of this project is to provide a suitable solution to some of the problems related to mobile data security such as locking the phone, changing profile settings and clearing call logs in a single android application. The user can remotely access and control a lost or stolen phone. The features of this application are:-

- When the phone is lost, the owner of the phone can lock the phone remotely in case they forget to do so.
- The call logs on the phone can be erased.
- The profile settings of the phone can be changed to silent, vibrate or general ring mode remotely.
- After a certain number of failed attempts, the phone number of the malicious user is blocked.
- The secret code can be recovered in case the owner of the phone forgets it.
- The owner of the phone can change the settings on the phone to activate all the above remote assist features on a settings screen which can be accessible only to him by dialing a secret code on the keypad. This code activates and launches the settings screen.

A customized service is built using android framework that runs in the background without any UI. This service listens to the incoming message notifications from the Broadcast receivers. The SMS is then parsed and a check for authentication and syntax is done and after verification, the necessary action such as lock, hide, and ring is taken.
2. Remote Assist App

2.1 Problem Statement

Due to the rapid progress in technology, smart phones have become an important part of our day to day activities. A huge amount of private and personal data is being saved on the smart phones such as business contacts or business calendar containing important scheduled meetings etc., private photos, text message and instant message conversations, banking information such as account numbers or saved passwords etc. Therefore, it may result in breach of confidentiality of the data in case the phone is lost or stolen. There is a need to be able to ensure the security of the data on the phone and also a means for the owner to be able to have access to his phone even if it is lost or misplaced. The user needs to be able to remotely control his phone without the use of any third party web servers.

2.2 Motivation

A study (Sullivan 2012) that was conducted by Symantec corp., a computer security firm revealed that 89% of the people who found the lost phone tried to gain access to unauthorized data on lost mobile phones. As part of the study, Symantec deliberately lost 50 android phones loaded with tracking and logging software and placed them in locations where they could be easily seen by people. The employees at the firm could track the phone and the activities being performed on the phone. Results showed that 43% of the people clicked on "online banking", 53% clicked on "HR Salaries", 57% tried to open a folder named "saved passwords", 60% tried to access the social networking apps and personal e-mail, 72% of the people tried to view a folder named "private photos". This study shows that it is necessary to be able to secure the data
on the phone so that it cannot be misused by unauthorized people. The user should have control over his phone even in cases where the phone is lost or stolen.

### 2.3 System Description

Based on the above studies, it is important to have an application that allows the user to have control over his phone even after losing or misplacing it. Therefore the intention is to develop an application integrating the features such locking the phone, changing the lock code of the phone, erasing the logs and also changing the profile settings of the phone. All these actions can be performed by the user remotely by sending text message to the phone along with the secret code. The secret code can be set by the user in the settings screen of the application. In order to view the settings screen of the application, the user has to dial a secret number on the keypad which launches the application and makes it visible to the user.

### 2.4 Project Scope

This project consists of an Android application that makes use of some of the built in components of android such as services, broadcast receivers, telephony manager API and content providers shown in the Figure 2.1 (TutorialsPoint 2014). The application is first launched by dialing 00000 on the keypad. Once the application is launched, the user can set his preferences for the application and also the secret code. The secret code is an alpha numeric code which can be varied. This application runs as a service and monitors the incoming text message for the commands. Each command from the sender's device has to include the secret code which is used for authentication. After verifying that the secret code is correct, the application checks for the
operation code (Lock, Ring, etc.) The correct operation is performed and a message is sent back to the sender's mobile as acknowledgement.

2.4.1 System Architecture

![System level architecture diagram](https://www.tutorialspoint.com/android/images/android_systemArchitecture.png)

**Figure 2.1. System level architecture diagram.** (TutorialsPoint 2014)

Figure 2.1 (TutorialsPoint 2014) represents the android system architecture. Some of the important components of android used in this project are

- **Telephony Manager** - With the help of this module, the user can launch the application by dialing the secret number on the keypad. This module also performs the task of checking the status of the phone whether it is in ring mode or silent etc.

- **Broadcast receivers** - An Outgoing Call Broadcast Receiver and an Incoming SMS Broadcast Receiver are used in the application. The outgoing call broadcast receiver listens to all the outgoing calls. For example, when the number 00000 is dialed on the keypad, the application is launched with the help of the telephony manager API. The
incoming SMS broadcast receiver listens to all the messages coming to the device. The SMS sender details like number, message and body are sent to the customized service

- Service- A customized service is built which runs in the background without any UI. This service listens to the incoming message notifications from broadcast receivers. The message is then parsed and a check for authentication and syntax is done. After verifying if the code is correct, the appropriate action (lock, vibrate etc.) is performed and acknowledgement is sent.

2.5 User Interface

The app has three activities: User settings activity, About activity and Instructions activity. The activities are the screens with which the user can interact. The user settings activity is the main activity where the user can apply the settings. It extends PreferenceActivity in order to monitor the changes made by the user. In the user settings activity, we use a shared preferences file which can be later accessed from other classes for the purpose of verifying the secret code, secret answer and sending a secret question to the user. There is an onClickListener set for the unblock all senders button which listens for the user response and calls the utils.unblockAllSenders() method. Figure 2.2 shows the user settings activity.

The action bar has an overflow menu with two menu items: About and Help. Figure 2.3 shows the screen shot with the overflow menu.

The About activity contains just an image of the app icon. This is the screen that the user will be able to view when he opens the app without using the launch code 00000. The user can also go to this activity by selecting the About option in the overflow menu. Figure 2.4 shows the About activity screen.
The Instructions activity is the screen that contains the instructions for the user on how to use the app, that is, the correct format for sending the secret code and command separated by the command separator ##. The Instructions activity class is called when the user clicks on the help option in the overflow menu. Figure 2.5 shows the Instructions activity screen.

Figure 2.2 User Settings Activity

Figure 2.3 Overflow Menu
Figure 2.4 About Activity

Figure 2.5 Instruction Activity

INSTRUCTIONS:
Send a Message to your phone with the following text:

1. Lock phone: CODE##LOCK
2. Ring Mode: CODE##RING
3. Vibrate Mode: CODE##VIBRATE
4. Silent Mode: CODE##SILENT
5. Clear call logs: CODE##CLR_LOGS
6. Recover Forgotten Code: CODE##SECRET_CODE##FORGOT
3. SYSTEM DESIGN

In this chapter, the architecture and overall design of the system is discussed, and the use cases of the system are analyzed.

The system level design and architecture is discussed briefly in the following section.

Figure 3.1 shows the programming level design of the system.

3.1 Design rationale

![Diagram of Programming Level Design]

**Figure 3.1: Programming Level Design**

When the dialer button is clicked, a call is sent to the operating system and the OutgoingCallReceiver verifies the dialing code and starts the activity. Any changes made in the
settings activity are made to the preferences file, and device admin permissions are acquired from the settings activity. When an incoming message is sent, the IncomingSMSReceiver reads the message and verifies the secret code and format. The request is processed and sent to SmsManager; which then sends an acknowledgement message to the user. In the case of locking the phone and resetting the password, the Device admin permissions are obtained and then the request is processed.

Figure 3.2 shows the codes in correct format to perform certain actions.

- Code to lock phone: <SECRET CODE>##LOCK
- Code to change to ring mode: <SECRET CODE>##RING
- Code to change to vibrate mode: <SECRET CODE>##VIBRATE
- Code to change to silent mode: <SECRET CODE>##SILENT
- Code to erase call log: <SECRET CODE>##CLR_LOGS
- Code to recover forgotten code: SECRET_CODE##FORGOT
- Code to answer security question: SECRET_CODE##<ANSWER>

**Figure 3.2: Codes to perform certain actions**

When the SMS with the secret code is received, the app verifies the secret code and, if it is valid, it looks further for the commands (such as LOCK, RING, CLR_LOGS), and those operations are carried out. If the secret code received is not valid, it will send message announcing that the code is invalid. After five successive attempts to access the phone with an invalid code, the sender will be blocked from accessing the phone. For example, if "LOCK" command is received in the SMS, it uses Telephony manager API to perform the lock operation.
3.2 Object oriented analysis and design

The following sub sections give a description of the use cases, sequence diagrams and class diagrams.

3.2.1 Use-Case Diagrams

The use-case diagram consists of a set of use-cases and a set of actors that are involved with at least one use case. It graphically represents the interactions amongst the different elements of the system, by describing who uses the system and what can be done with it.

Figure 3.3 shows the use case diagram for the application and Table 3.1 gives its description.

Use-Case for RemoteAssist Application:

![Use Case Diagram for RemoteAssist Application](image)

Figure 3.3: Use Case Diagram for RemoteAssist Application
### Table 3.1: Use Case Description for RemoteAssist Application

<table>
<thead>
<tr>
<th><strong>TITLE:</strong></th>
<th>RemoteAssist Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary:</strong></td>
<td>This use case allows the user to Launch the app and set the secret code which is later on used to lock the phone, change the lock, change the profile settings, erase the call logs and block any malicious users. The secret code can also be retrieved if forgotten.</td>
</tr>
<tr>
<td><strong>Actors:</strong></td>
<td>Application User</td>
</tr>
<tr>
<td><strong>Precondition:</strong></td>
<td>User must initialize and run the application.</td>
</tr>
<tr>
<td><strong>Trigger:</strong></td>
<td>User dials 00000 on the keypad to launch the RemoteAssist app.</td>
</tr>
</tbody>
</table>
| **Main Successful Scenario:** | The app allows actions such as:  
1. Set 5-digit secret code.  
2. Lock the phone.  
3. Change the phone lock password.  
5. Erase call logs  
6. Block senders after 5 invalid attempts.  
7. Retrieve secret code if forgotten by owner. |

**Use Case to Launch RemoteAssist App:**

Figure 3.4 shows the use case diagram to launch the application and Table 3.2 gives its description.

![Use Case Diagram](image)

**Figure 3.4: Use Case diagram to launch RemoteAssist App**
Table 3.2: Use Case Description to launch RemoteAssist App

<table>
<thead>
<tr>
<th>TITLE:</th>
<th>Launch the RemoteAssist Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary:</td>
<td>This use case allows the user to Launch the app and display the settings screen.</td>
</tr>
<tr>
<td>Actors:</td>
<td>Application User, RemoteAssist App</td>
</tr>
<tr>
<td>Precondition:</td>
<td>User must initialize and run the application.</td>
</tr>
<tr>
<td>Trigger:</td>
<td>User dials 00000 on the keypad to launch the RemoteAssist app.</td>
</tr>
<tr>
<td>Main Successful Scenario:</td>
<td>The app displays information such as:</td>
</tr>
<tr>
<td></td>
<td>1. Settings screen</td>
</tr>
</tbody>
</table>

Use Case to Manage Settings:

Figure 3.5 shows the use case diagram to manage settings, and Table 3.3 gives its description.

![Use Case Diagram to Manage Settings](image)

**Figure 3.5: Use Case Diagram to Manage Settings**

Table 3.3: Use Case Description to Manage Settings

<table>
<thead>
<tr>
<th>TITLE:</th>
<th>Manage the Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary:</td>
<td>This use case allows the user to manage the user settings and also set the 5-digit secret code.</td>
</tr>
<tr>
<td>Actors:</td>
<td>Application User</td>
</tr>
</tbody>
</table>
### Use Case to Lock the Phone:

Figure 3.6 shows the use case diagram to lock the phone, and Table 3.4 gives its description.

**Figure 3.6: Use Case Diagram to Lock the Phone**

**Table 3.4: Use Case Description to Lock the Phone**

<table>
<thead>
<tr>
<th>TITLE:</th>
<th>Lock the Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary:</td>
<td>This use case allows the user to send a 5-digit secret code along with the LOCK command. The application identifies the code and locks the phone. It also sends an acknowledgement message to the user.</td>
</tr>
<tr>
<td>Actors:</td>
<td>Application User, RemoteAssist App</td>
</tr>
</tbody>
</table>
Precondition: User must initialize and run the application. Also, the 5-digit secret code needs to be set.

Trigger: User sends 5-digit code and command in SMS format.

Main Successful Scenario: The app performs actions such as:
  1. Lock the phone.
  2. Send success message as Acknowledgement.

Use Case to Change Profile Settings:

Figure 3.7 shows the use case diagram to change the profile settings, and Table 3.5 gives its description.

![Use Case Diagram to Change the Profile Settings](image)

**Figure 3.7: Use Case Diagram to Change the Profile Settings**

**Table 3.5: Use Case Description to Change the Profile Settings**

<table>
<thead>
<tr>
<th>TITLE:</th>
<th>Change the Profile Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary:</td>
<td>This use case allows the user to send a 5-digit secret code along with the command to change the profile setting to either ring, vibrate or silent. The application identifies the code and changes settings. It also sends an acknowledgement message to the user.</td>
</tr>
<tr>
<td>Actors:</td>
<td>Application User, RemoteAssist App</td>
</tr>
</tbody>
</table>
Precondition: User must initialize and run the application. Also, the 5-digit secret code needs to be set.

Trigger: User sends 5-digit code and command in SMS format.

Main Successful Scenario: The app performs actions such as:
1. Change the profile to ring/vibrate/silent mode.
2. Send success message as Acknowledgement

Use Case to Erase Call Logs:

Figure 3.8 shows the use case diagram to erase call logs, and Table 3.6 gives its description.

![Use Case Diagram to Erase Call Logs](image)

**Figure 3.8: Use Case Diagram to Erase Call Logs**

<table>
<thead>
<tr>
<th>TITLE:</th>
<th>Erase Call Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary:</td>
<td>This use case allows the user to send a 5-digit secret code along with the command to erase the call logs. The application identifies the code and deletes the recent calls in the log. It also sends an acknowledgement message to the user.</td>
</tr>
<tr>
<td>Actors:</td>
<td>Application User, RemoteAssist App</td>
</tr>
<tr>
<td>Precondition:</td>
<td>User must initialize and run the application. Also, the 5-digit secret code needs to be set.</td>
</tr>
<tr>
<td>Trigger:</td>
<td>User sends 5-digit code and command in SMS format.</td>
</tr>
</tbody>
</table>
Main Successful Scenario:
The app performs actions such as:
1. Erase call logs
2. Send success message as Acknowledgement

Use Case to Block and Unblock senders:

Figure 3.9 shows the use case diagram to block and unblock senders, and Table 3.7 gives its description.

![Use Case Diagram to Block and Unblock sender](image)

**Figure 3.9: Use Case Diagram to Block and Unblock sender**

**Table 3.7: Use Case Description to Block and Unblock sender**

<table>
<thead>
<tr>
<th>TITLE:</th>
<th>Block and Unblock Senders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary:</td>
<td>This use case allows the app to block the unauthorized user after he/she attempts to access the phone 5 times with the invalid code. An invalid code message is sent to the sender. The user can unblock all senders by clicking on the button /.</td>
</tr>
<tr>
<td>Actors:</td>
<td>Application User, RemoteAssist App</td>
</tr>
<tr>
<td>Precondition:</td>
<td>User must initialize and run the application. Also, the 5-digit secret code needs to be set.</td>
</tr>
<tr>
<td>Trigger:</td>
<td>User sends invalid 5-digit code and command in SMS format five times.</td>
</tr>
</tbody>
</table>
Main Successful Scenario:

The app performs actions such as:
1. Send Invalid Code message to sender.
2. Blocks the sender after 5 failed attempts.
3. Unblocks all senders when the correct button is clicked.

Use Case to Retrieve Secret Code:

Figure 3.10 shows the use case diagram to retrieve the secret code, and Table 3.8 gives its description.

![Use Case Diagram to Retrieve the Secret Code](image)

**Figure 3.10: Use Case Diagram to Retrieve the Secret Code**

**Table 3.8: Use Case Description to Retrieve the Secret Code**

<table>
<thead>
<tr>
<th>TITLE:</th>
<th>Retrieve Secret Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary:</td>
<td>This use case allows the user to send a message with a command to retrieve the secret code in case the user forgets it. The user is then asked a security question. The security answer should be sent as an SMS. It is then verified by the application and the secret code is sent, again via SMS.</td>
</tr>
<tr>
<td>Actors:</td>
<td>Application User, RemoteAssist App</td>
</tr>
</tbody>
</table>
**Precondition:** User must initialize and run the application. Also, the 5-digit secret code, Security question and security answer needs to be set.

**Trigger:** User sends command for ‘forgot secret code’.

**Main Successful Scenario:** The app performs actions such as:
1. Retrieve the secret code through SMS.

### 3.2.2 Sequence Diagram

Figure 3.11 shows the sequence diagram for the application.

![Sequence Diagram](image)

**Figure 3.11:** Sequence diagram for RemoteAssist application

### 3.2.3 Class diagram

Figure 3.12 shows the Class diagram of the application.
Figure 3.12: Class diagram for RemoteAssist app
Some of the important classes used in this application are:-

**OutgoingCallReceiver** - The methods that are used in this class are OutgoingCallReceiver(), onReceive(). When the user dials 00000 on the keypad, it verifies the number entered with the dial code already set in the preferences. The outgoing call is then redirected to the RemoteAssist Service and the app is launched. The apps should first prevent the call from being placed by setting resultData to null.

**UserSettingsActivity** - In this class, the user settings layout is displayed and all the settings are applied, such as security question, secret code, security answer. If the device admin is enabled, the activation page is launched. An intent is used to call the DevicePolicyManager. When the RemoteAssist is enabled, the IncomingSMSReceiver class is called.

**AboutActivity** - The image for the about screen is displayed.

**InstructionsActivity** - The instructions for the user on how to access the phone are given, that is, the correct format and command to perform each operation.

**MyPreferences** - Using the key name and the default value, the preferences are updated by calling the getSharedPreferences() method. The PreferenceChangeListener class listens for any changes in the preferences.

**MyDeviceAdminReceiver** - Whenever the administrative operations such as phone lock are performed, this class is called.

**RemoteAssistApplication** - Initializes all the MyPreferences.

**Utils** - Using this class, the sender is blocked after five invalid attempts by adding his number to the BlockedSenderList, and the senders can also be unblocked. The failure count is maintained. Any message to the sender is sent from this class. If the device admin is enabled, then the administrator permissions are activated.
**IncomingSMSReceiver** - The methods used in this class are IncomingSMSReceiver(), log(String), onReceive(Context, Intent), parseCommandForAction(Context, String, String). With the help of these methods and the Telephony manager API, the message sent by the user is received in a bundle, read and parsed to check if the sender is not blocked and if the format matches and then it is sent to the RemoteAssistService and the service is started.

**RemoteAssistService** - The important methods used in this class are:-

- parseMessageForCommandCode(String) - This method parses the message for command code and compares the specified string with the given string and returns true if they are equal, else false.

- processForgotSecretCodeCommand() - This method processes the forgot secret code command.

- processSecretCodeRecoveryQuestionCommand() - With the help of this method, the security question is sent and the security answer is verified with the answer that is already saved in the array.

- processPhoneLockCommand() - The device can be locked with a password and others cannot change it until the existing password is entered.

- processClearLogsCommand() - The recent sent and received calls from the call log database are queried and then deleted.

- processPhoneRingerCommand() - This method makes use of an android component called Audio manager to get access to volume and ringer mode control. RINGER_MODE_NORMAL is set which enables the phone to be audible.

- processPhoneSilentCommand() - As above, uses Audio manager to get access to volume and ringer mode control. RINGER_MODE_SILENT is set which enables the phone to be silent and not vibrate.
-processPhoneVibrateCommand() - Also uses Audio manager. RINGER_MODE_VIBRATE is set which enables the phone to be silent and vibrate.
4. PROJECT IMPLEMENTATION

4.1 System Flow:

Figure 4.1 represents the flow of the system. When the phone is lost or misplaced, the owner of the phone can use \textit{any other device} to send a text message containing the secret code and the operation to be performed to the lost phone. The broadcast receiver monitors the incoming text message and sends it to the RemoteAssistService. The message is then parsed and is verified for authenticity. If the secret code is correct, then the required operation is performed and an acknowledgement message is sent back to the phone from which the message is sent. If the secret code is wrong, a message saying that the code is invalid is sent to the sender. After 5 successive failed attempts, the sender is blocked.

When the user sends the command to recover the secret code, it is verified. If it is correct, then a security question is sent to the user and if it is wrong, it sends a message back saying that the command could not be processed. The user has to reply with the security answer. If the security answer is correct, then the secret code is sent to the user and if it is wrong, a message saying that the command cannot be processed is sent to the user.
Figure 4.1: System Flow Diagram
4.2 Software and Hardware Requirements

- Android SDK

This provides the necessary tools and API libraries to build Android applications. It also provides tools to test and debug android applications.

- Android device

An android device is used in this project to install and run the RemoteAssist application and also for testing the functionalities and working of the application.

- Android OS version Gingerbread to KitKat

This application can be run on android devices with android OS versions greater than 2.3.

4.3 Functional Requirements

The project can be divided into different modules and some sub modules. They are as follows:-

1. Launch the application

2. Enable Remote Assist and Device Admin.

3. Set secret code and security question.

4. Lock phone.

5. Change Profile Settings.
   - Change to Ring mode.
   - Change to Vibrate mode.
   - Change to Silent mode.

6. Erase call logs.

7. Block senders.

8. Recover forgotten secret code.
9. Delete text messages.
10. Retrieve Location.
11. Delete Accounts.
12. Factory Reset.

4.4 Android application

4.4.1 Application connectivity

A connection needs to be established between the Android phone and the application in order for the application to run. After connection establishment, the app should be run by enabling the remote assist features and giving administrative permissions by activating it. As soon as this is done, the Broadcast Receivers start listening for the occurrence of any event, such as outgoing calls or incoming messages. The messages are then sent to the customized service, which parses and verifies the secret code and the command, and performs the operations.

4.4.2 Launch the application:

When the user enters the dialing code and the dialer key is clicked, the OutgoingCallReceiver notifies the BroadcastReceiver and the onReceive() method is called. It verifies the dialing code by checking it with the default dialing code and if it is correct, then it calls the UserSettingsActivity class. Figure 4.2 shows the code snippet for OutgoingCallReceiver.

```java
@Override
public void onReceive(Context context, Intent intent) {
    final String actionName = intent.getAction();
    log(actionName);
    if (Intent.ACTION_NEW_OUTGOING_CALL.equals(actionName)) {
        final String dialingCode = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);
        log("Dialling Code " + dialingCode);
        if (TextUtils.equals(IConstants.USER_SETTINGS_DIALING_CODE, dialingCode)) {
```
setResultData(null);
abortBroadcast();
if (!Utils.isDeviceAdminEnabled(context)) {
    MyPreferences.saveIsPrefDeviceAdminEnabled(false);
}
final Intent userSettingIntent = new Intent(context, UserSettingsActivity.class);
userSettingIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startActivity(userSettingIntent);

**Figure 4.2 : Code snippet for OutgoingCallReceiver**

The user will be able to launch and run the application by dialing 00000 on the keypad. Until and unless this is done, the application will not run. Figure 4.3 shows the screenshot.

The settings screen will be displayed, where the user can manage all the required settings such as enabling/disabling remote assist, set secret code, security question and unblock all senders. Figure 4.4 shows the settings screen.

**Figure 4.3: Code to launch the application.**
4.4.3 Enable Remote Assist and Device Admin:

Whenever the Remote Assist or Device Admin in checked on, the DevicePolicyManager, ACTION_ADD_DEVICE_ADMIN, is called through an intent. Figure 4.5 shows the code snippet for this action.

The user can turn on the remote assist button in order to activate the remote assist feature. This feature allows the user to access the phone remotely and perform certain operations. The Remote Assist and Device Admin are synchronized. Enabling one of them also enables the other. When the device admin option is enabled, it asks the user permission to confirm the activation. By activating this feature, the app gets permissions to perform actions such as erasing the data, change unlock password. Figure 4.6 and Figure 4.7 shows the screen shots.
if (!isAdminEnabled) {
    Utils.removeAsDeviceAdmin(this);
    isDeviceAdminConfigShown = false;
}

if (!isDeviceAdminEnabled && isAdminEnabled && !isDeviceAdminConfigShown) {
    isDeviceAdminConfigShown = true;
    final Intent intent = new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
    intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, mComponentName);
    intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION, getString(R.string.device_admin_message));
    startActivityForResult(intent, ADMIN_INTENT_REQUEST_CODE);
}

Figure 4.5: Code Snippet for Remote Assist and Device Admin

Figure 4.6: Enable Remote Assist
4.4.4 Set secret code and security question:

a. **Secret Code:**

The user can set the 5-digit secret code, which is later on used to perform certain operations such as locking the phone, changing the profile settings etc. It is an editTextPreference. Figure 4.8 shows the screenshot.

b. **Security Question:**

The user can set the security question by selecting from the list of questions available. When the user tries to recover the secret code, this question will be sent as an SMS and the user should be able to reply with the correct answer in order to recover the secret code. It is a listPreference. Figure 4.9 shows the screenshot.
c. **Set Security Answer:**

The user can set the answer to a particular security question that has been selected. This answer has to be sent as an SMS when the security question is asked. This allows the user to retrieve the secret code. It is an editTextPreference. Figure 4.10 shows the screenshot.

![Figure 4.8: Set Secret Code](image)

![Figure 4.9: Set Security Question](image)
4.4.5 Lock the phone:

The user should send an SMS containing the secret code and LOCK command in the pre-defined format. The app parses the message, recognizes the secret code and locks the phone with the secret code as the lock password. An acknowledgement message is sent to the user, confirming the successful completion of the command. Figure 4.12 shows the screen shot of the SMS sent in desired format.

The device administrator rights are obtained if it is checked on in the app. It gets the secret code from the shared preferences file by calling myPreferences.getPrefSecretCode(), which returns the secret code based on the key and default value. It checks whether the secret code and message format is correct, and then it process the request using the DevicePolicyManager.lockNow() and DevicePolicyManager.resetPassword(). Figure 4.11 shows the implementation of the phone lock command.
if (isAdmin) {
    final String password = MyPreferences.getPrefSecretCode();
    if (!TextUtils.isEmpty(password)) {
        mDevicePolicyManager.resetPassword(password, DevicePolicyManager.RESET_PASSWORD_REQUIRE_ENTRY);
    }
    mDevicePolicyManager.lockNow();

    sb.append(getString(R.string.cmd_success)).append(NEW_LINE_TOKEN).append(getString(R.string.phone_locked_Success));
} else {

    sb.append(getString(R.string.cmd_failure)).append(NEW_LINE_TOKEN).append(getString(R.string.phone_locked_failure));
}
final String message = sb.toString();
log(message);

..........................
Utils.sendMessage(mMessageSender, message);

Figure 4.11: Code snippet to lock the phone

Figure 4.12: SMS in desired format to Lock the Phone

Figure 4.13 shows the android phone locked with the secret code after the operation has been performed.
4.4.6 Change Profile Settings:

This has three parts. They are:-

a. Change to ring mode

b. Change to vibrate mode

c. Change to silent mode

a. Change to ring mode:

The user sends the 5-digit secret code along with the ring command in a pre-defined format as an SMS to the phone. The app parses the SMS, recognizes the secret code and using the AudioManager it changes the phone's setting to ring mode. It also sends an acknowledgement
message confirming the success of the operation. Figure 4.14 shows a screen shot of the SMS sent in desired format.

![Figure 4.14: SMS in desired format to change to ring mode](image)

The processPhoneRingerCommand() uses AudioManager to set the phone mode to normal and then it calls the utils.sendMessage() method for sending the response to the user. Figure 4.15 shows the code snippet to show how the phone mode is changed to ringer mode.

```java
private void processPhoneRingerCommand() {
    final AudioManager audioManager = (AudioManager)
    getSystemService(AUDIO_SERVICE);
    audioManager.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
    final StringBuilder sb = new StringBuilder(getString(R.string.cmd_success));
    sb.append(NEW_LINE_TOKEN).append(getString(R.string.ringer_mode_message)).append(SPACE_TOKEN).append(getString(R.string.ringer_mode_normal));
    final String message = sb.toString();
    log(message);
    Utils.sendMessage(mMessageSender, message);
}
```

![Figure 4.15: Code snippet to change phone to ring mode](image)
b. **Change to vibrate mode:**

The user sends the 5-digit secret code along with the vibrate command in a pre-defined format as an SMS to the phone. The app parses the SMS, recognizes the secret code and using the Audio manager, it changes the phone's setting to vibrate mode. It also sends an acknowledgement message confirming the success of the operation. Figure 4.16 shows a screen shot of the SMS sent in desired format.

![Figure 4.16: SMS in desired format to change to vibrate mode](image)

The processPhoneVibrateCommand() uses AudioManager to set the phone mode to vibrate and then it calls the utils.sendMessage() method for sending the response to the user. Figure 4.17 shows the code snippet to show how the phone mode is changed to vibrate mode.

```java
private void processPhoneVibrateCommand() {
    AudioManager audioManager = (AudioManager)
        getSystemService(AUDIO_SERVICE);
    audioManager.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
    final StringBuilder sb = new StringBuilder(getString(R.string.cmd_success));
    sb.append(NEW_LINE_TOKEN).append(getString(R.string.ringer_mode_message)).append(SPACES_TOKEN).append(getString(R.string.ringer_mode_vibrate));
    final String message = sb.toString();
    log(message);
}```
c. **Change to silent mode:**

The user sends the 5-digit secret code along with the silent command in a pre-defined format as an SMS to the phone. The app parses the SMS, recognizes the secret code and using the AudioManager, it changes the phone's setting to silent mode. It also sends an acknowledgement message confirming the success of the operation. Figure 4.18 shows a screen shot of the SMS in the desired format.

![Figure 4.18: SMS in desired format to change to silent mode](image)

The processPhoneSilentCommand() uses AudioManager to set the phone mode to silent and then it calls the utils.sendMessage() method for sending the response to the user. Figure 4.19 shows the code snippet to show how the phone mode is changed to silent mode.

```java
private void processPhoneSilentCommand() {
    // Code snippet
}
```
```java
final AudioManager audioManager = (AudioManager) getSystemService(AUDIO_SERVICE);
audioManager.setRingerMode(AudioManager.RINGER_MODE_SILENT);
final StringBuilder sb = new StringBuilder(getString(R.string.cmd_success));

sb.append(NEW_LINE_TOKEN).append(getString(R.string.ringer_mode_message)).append(SPACE_TOKEN).append(getString(R.string.ringer_mode_silent));
final String message = sb.toString();
log(message);
Utils.sendMessage(mMessageSender, message);
```

Figure 4.19: Code snippet to change phone mode to silent

4.4.7 Erase call logs

The user sends the 5-digit secret code along with the command to erase the call logs in a predefined format as an SMS to the phone. The app parses the SMS, recognizes the secret code, accesses the call log database and clears all the call logs on the phone completely. It also sends an acknowledgement message confirming the success of the operation. Figure 4.20 shows a screen shot of the SMS sent in desired format.

![Figure 4.20: Erase Call Logs](image)
The method uses getContentResolver().delete(CallLog.Calls.CONTENT_URI) to delete the call logs and then it calls the utils.sendMessage() method for sending the response to the user. Figure 4.21 shows the implementation of processClearLogsCommand() method.

```java
private void processClearLogsCommand() {
    getContentResolver().delete(CallLog.Calls.CONTENT_URI, null, null);
    final StringBuilder sb = new StringBuilder(getString(R.string.cmd_success));
    sb.append(NEW_LINE_TOKEN).append(getString(R.string.call_logs_cleared))
    final String message = sb.toString();
    log(message);
    Utils.sendMessage(mMessageSender, message);
}
```

**Figure 4.21: Code snippet to clear call logs**

### 4.4.8 Block sender

An unauthorized user might try to hack the secret code by bombarding the phone with many messages containing various combinations of codes. In order to avoid this threat, the app blocks the sender from accessing the phone after 5 invalid attempts. No acknowledgement message is sent after 5 failed attempts. Figure 4.22 and Figure 4.23 show the attempts to access the phone with invalid codes.
A counter is set in order to count the number of times a wrong code is sent by user which is saved in the shared preferences file. Each time a wrong code is sent, it sends a response by calling the utils.sendMessage() method and when the failure count is equal to 5, it removes the user phone number from myPreferences file. The user phone number is also added to the list using...
utils.addToBlockedSenderList() so that the app does not send any response to the user. Figure 4.24 shows the code snippet.

```java
public static void blockSenderIfPossible (String sender) {
    int failureCount = MyPreferences.getFailureCountForSender(sender);
    failureCount++;
    log("Failed attempt count : " + failureCount);
    MyPreferences.updateFailureCountForSender(sender, failureCount);
    if (failureCount >= MAX_FAILED_ATTEMPT) {
        MyPreferences.removeSender(sender);
        Utils.addToBlockedSenderList(sender);
    } else {
        Utils.sendMessage(sender, "Invalid Code. Please Try Again.");
    }
}
```

**Figure 4.24: Code snippet to block sender**

4.4.9 Recover forgotten secret code

case the owner of the phone forgets the secret code, he can retrieve it in an SMS by sending a message to the phone with a predefined command in a specific format. The command is then verified by the app and a security question is sent out as an SMS. In order to retrieve the secret code, the correct answer to the security question should be sent as a reply in the desired format. Figure 4.25 shows the screen shot of the message sent in desired format with the command in order to retrieve the secret code. Figure 4.26 shows the security question, the security answer and the secret code that has been retrieved.
Figure 4.25: Recover Secret Code

Figure 4.26 Security Question and Answer
The method `processSecretCodeRecoveryQuestionCommand()` is called when the user requests the secret code. The security question is sent to the user. When the user answers the security question correctly, the `processForgotSecretCodeCommand()` method sends the secret code to the user. Figure 4.27 shows the code snippet.

```java
private void processForgotSecretCodeCommand() {
    final String defaultSecretCode = MyPreferences.getPrefSecretCode();
    final StringBuilder sb = new StringBuilder(getString(R.string.cmd_success));
    sb.append(NEW_LINE_TOKEN).append(getString(R.string.settings_secret_code_summary)).append(NEW_LINE_TOKEN).append(defaultSecretCode);
    final String message = sb.toString();
    log(message);
    Utils.sendMessage(mMessageSender, message);
}
```

```java
private void processSecretCodeRecoveryQuestionCommand() {
    final int questionIndex = Integer.parseInt(MyPreferences.getPrefSecurityQuestion());
    final String[] questionsArray = getResources().getStringArray(R.array.array_key_security_questions);
    final String selectedSecurityQuestion = questionsArray[questionIndex];
    final StringBuilder sb = new StringBuilder(getString(R.string.security_question_message));
    sb.append(NEW_LINE_TOKEN).append(selectedSecurityQuestion);
    final String message = sb.toString();
    log(message);
    Utils.sendMessage(mMessageSender, message);
}
```

**Figure 4.27**: Code snippet to recover secret code

### 4.4.10 Delete Text Messages

The user sends the 5-digit secret code along with the command to delete the text messages on the phone in a pre-defined format as an SMS to the phone. The app parses the SMS, recognizes the
secret code, accesses the SMS database and clears all the messages on the phone. It also sends an
acknowledgement message confirming the success of the operation.

A cursor is used to provide read-write access to the result set returned by the SMS database
query. All the SMS messages are saved in a table in the database. Each SMS message
corresponds to a row id. moveToNext() method is used to move the cursor to the next row in the
table. getContentResolver().delete(Uri.parse("content://sms/" + id), null, null) method is used to
delete all the SMS messages based on the row id. The URI “content://sms/” should be provided.
All the messages will be deleted except the current message sent. Figure 4.28 shows the code
snippet to delete text messages.

```java
Cursor c = getApplicationContext().getContentResolver().query(Uri.parse("content://sms/"), null, null, null, null);
while (c.moveToNext()) {
    System.out.println("While loop in clear messages");
    int id = c.getInt(0);
    getApplicationContext().getContentResolver().delete(Uri.parse("content://sms/" + id), null, null);
} c.close();
```

**Figure 4.28: Code Snippet to Delete Text Messages**

4.4.11 Retrieve GPS Co-ordinates:

The user sends the 5-digit secret code along with the command as an SMS message in a pre-
deﬁned format, to retrieve the GPS co-ordinates of the phone’s location. The app parses the
SMS, recognizes the secret code and returns the latitude and longitude co-ordinates of the
phone’s location. It also sends an acknowledgement message confirming the success of the operation.

Location class in android is used to represent a geographic location. The minimum distance to update the location is set to 10 meters and the minimum time for each update is set to 1 minute. LOCATION_SERVICE is used with getSystemService() method to retrieve a location manager for controlling location updates. The last known location is obtained using the NETWORK_PROVIDER, which determines the location based on the availability of a cell tower. Similarly, GPS_PROVIDER is used to determine the location using satellites. Figure 4.29 shows the code snippet for retrieving the GPS co-ordinates.

```java
locationManager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER,
    MIN_TIME_BW_UPDATES,
    MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
Log.d("GPS Enabled", "GPS Enabled");
if (locationManager != null) {
    location = locationManager
        .getLastKnownLocation(LocationManager.GPS_PROVIDER);
    if (location != null) {
        latitude = location.getLatitude();
        longitude = location.getLongitude();
    }
}
```

**Figure 4.29: Code Snippet to Retrieve GPS Co-ordinates**

### 4.4.12 Delete Accounts:

The user sends the 5-digit secret code along with the command as an SMS message in a pre-defined format, to delete the accounts on the phone in the account manager. The app parses the SMS, recognizes the secret code and deletes the accounts. It also sends an acknowledgement message confirming the success of the operation.
An account manager variable is created and all the accounts on the device are obtained using the getAccounts() method and stored in an array. This method lists all the accounts of any type registered on the device. Based on the index number and the length of the account, the accounts are deleted using removeAccount() method. In order to perform these operations, GET_ACCOUNTS and MANAGE_ACCOUNTS permissions should be added to the AndroidManifest.xml file. Figure 4.30 shows the code snippet.

```java
AccountManager accountManager = AccountManager.getInstance(this);
Account[] accounts = accountManager.getAccounts();
for (int index = 0; index < accounts.length; index++) {
    System.out.println("deleting each account");
    accountManager.removeAccount(accounts[index], null, null);
}
```

**Figure 4.30: Code Snippet to Delete Accounts**

### 4.4.13 Factory Reset:

The user sends the 5-digit secret code along with the command as an SMS message in a predefined format, to perform a factory reset on the device. An acknowledgement message is sent to the user first and the operation is performed. All the data on the phone is wiped out and the phone is rebooted.

Using DEVICE POLICY SERVICE with getSystemService() method, a Device Policy Manager component is retrieved. wipeData() method is used to reset the device to factory settings. The <wipe-data> tag needs to be included in the admin.xml file in order to perform these operations and BIND_DEVICE_ADMIN permission needs to be included in the AndroidManifest.xml file. Figure 4.31 shows the code snippet.

```java
DevicePolicyManager mDPM =
(DevicePolicyManager) getSystemService(Context.DEVICE_POLICY_SERVICE);
```
mDPM.wipeData(0);

**Figure 4.31: Code Snippet to Perform Factory Reset**

4.4.14 **Master Command:**

The user sends the 5-digit secret code along with the Master Command as an SMS in a pre-defined format. The app parses the SMS, recognizes the secret code and performs the operations. It also sends an acknowledgement message confirming the success of the operation. The master command is set to perform various operations such as lock the screen, erase call logs and text messages, retrieve GPS co-ordinates and change the phone to silent mode, all at once.
5. EVALUATION AND EXPECTED RESULTS

5.1 Compatibility

Android SDK provides a virtual mobile device emulator that can be run on the computer. This emulator was used to develop and test the application in the initial development stages. After the application was developed, it was tested on live android mobile devices Xperia C and LG G2, running on android versions Jelly Bean (4.1-4.3.1) and KitKat (4.4-4.4.4). It was found to be compatible with these versions.

5.2 Security

The application provides security to the phone by locking it and by prompting the user to enter the security answer correctly in order to retrieve the secret code.

5.3 Portability

The application is built on JAVA and can be run on any device using Android operating system version greater than 2.3.

5.4 Reliability

The main purpose of this application is to allow the user to have control over his/her phone even if it is lost or stolen, and offers some means to secure the personal data on the phone. This can be achieved by using not only an android or smart phone, but also any other normal phone irrespective of whether that phone has an internet connection or not. The user might not have access to the web or a laptop to perform these operations, which is why the SMS method is used for convenience and ease of availability. As mentioned earlier, the app runs a customized service written to parse the commands and send SMS.
5.5 Testing

The proposed project was tested for the following phases:-

1. Launch the app phase
   - Successful Launch
   - Unsuccessful Launch

2. Remote Assist phase
   - Valid Code
   - Invalid Code

3. Recover Secret Code phase
   - Successful recovery
   - Unsuccessful recovery

5.5.1 Launch the app phase

In this phase, the user would be able to launch the app by dialing 00000 on the keypad. The two scenarios for successful launch and unsuccessful launch are presented below.

Successful Launch

The successful launch of the app displays the settings page, where the user can manage the necessary settings such as enable Remote Assist, enable Device Admin, set the secret code and unblock all senders. Figure 5.1(a) shows the screen shot after successful launch.

Unsuccessful Launch:
If the launch of the app is unsuccessful, it does not display the settings page. Instead it gives an error message. Figure 5.1(b) shows the screen shot of one such error screen.

![Figure 5.1(a): Successful launch](image1) ![Figure 5.1(b): Unsuccessful launch](image2)

### 5.5.2 Remote Assist phase

The operations that can be performed on the phone remotely such as lock, change to ring mode, vibrate mode, silent mode, erase call logs etc. come under the Remote Assist phase. Two common scenarios seen in this phase are listed below.

**Valid Code:**

When an authorized user tries to access the phone, the app parses the SMS and recognizes the secret code. If the secret code is valid, it performs the necessary action and sends an
acknowledgement message. The Figure 5.2(a) shows the screen shot of an operation performed, and acknowledgement received for a valid code.

Invalid Code:

When an unauthorized user tries to access the phone, the app will again parse the SMS and check the secret code. If the code is not valid, it does the necessary action and sends an error message saying. The Figure 5.2(b) shows the screen shot for invalid code.

![Figure 5.2(a): Valid Code](image)

![Figure 5.2(b): Invalid code](image)

5.5.3 Secret Code phase

If the owner of the phone forgets the secret code, they would be prevented by this system from accessing their own phone. To combat this, the secret code can be retrieved in an SMS when the user sends the predefined command to recover it. The two scenarios are shown below.
Successful Recovery:
When the owner of the phone sends the correct command in the desired predefined format, the application sends them a security question, which is already set during the installation. The user has to reply with the correct security answer in order to retrieve the secret code via an SMS. The screen shot for this scenario is given below in Figure 5.3(a).

![Successful Recovery](image)

Figure 5.3(a): Successful Recovery

Unsuccessful Recovery:
When the user sends an incorrect command to retrieve the secret code, an error message is displayed saying that the command is not processed, and the secret code is not retrieved. When the user sends the correct command but gives the wrong answer for the security question, the same results apply - the command will not be processed and the secret code will not be retrieved. The screen shot for unsuccessful recovery is shown in Figure 5.3(b) and Figure 5.3(c).
Figure 5.3(b): Unsuccessful Recovery with incorrect command

Figure 5.3(c) Unsuccessful Recovery with Incorrect Security Answer
6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The RemoteAssist application allows the user to have access to the phone even after it is lost or misplaced, providing that the phone is turned on. The owner of the phone can lock it remotely and erase the call logs. In cases where the phone is temporarily misplaced or lost, the user can change the profile settings of the phone from silent to ring or vibrate and vice versa. The user can also retrieve the GPS co-ordinates of the phone's location when the GPS is enabled. The accounts in the account manager can be removed and also, text messages on the phone can be deleted. An option to perform factory reset of the phone is also available. All these operations can be performed by sending a text message to the phone specifying the necessary operation along with the secret code known only to the user.

The user can change the secret code and choose whether to turn on/off the remote assist feature on the settings screen. The application will not be visible to anyone unless the owner of the phone dials a secret number on the keypad. This launches the application and allows the user to make the necessary changes. This feature is added as an extra layer of security to the app.

If an unauthorized user tries to hack the secret code by bombarding the phone with many messages containing combinations of secret codes, the app blocks the sender after 5 successive failed attempts. There is also a feature to unblock all senders. If the owner of the phone forgets the secret code, sending a specific command in the desired format can retrieve it. A security question is sent to the user and the user is required to reply with the pre-specified security answer.
This application does not require internet connection or the use of a third party website to access the phone. All these features can be achieved just by using the messaging services.

6.2 Future work

In future, this application could include a variety of other features such as allowing the user to set the dialing code in the user settings activity. This would be similar to how the user can set the secret code, with an editTextPreference field given, to allow the user to change the dialing code after he/she installs the application.

Another feature would be to include an alternate number in the user settings, so that when the SIM card is changed, an alert message is sent to the alternate contact number along with the new SIM card number.

One more useful feature that could be added is to block the simcard when malicious users try to replace the SIM card with their own. This would render the phone useless to the malicious user in terms of making calls and sending messages.
References


GooglePlay.


