COMP 4905 Honours Project Report

An SMS application to prevent Sender ID Spoofing

Author: Zhaohong Wan

Supervisor: Lianying Zhao

School of Computer Science

Carleton University

2020/08/14

Contents

Abstract

This document is a final stage report for a project titled “An SMS(Short Message Service) application to prevent Sender ID Spoofing.” Sender spoofing is a technique that can easily replace the Sender ID with alphanumeric text. This SMS application is developed by Android Studio, Firebase Realtime Database and Room database. It aims to identify the user of this application through the pre-established secret that is bound with the incoming messages. When a user sends an SMS message to a recipient, this SMS message will be sent through the SMS channel, but the secret for authentication will be stored in an online Database. After the authentication, the authentication results will be stored in the recipient’s mobile device, and the result will be displayed to the recipient. Through this SMS application, the user can quickly check if the incoming message is from the user or not.

Acknowledgement

1. Introduction

1.1 Problem

Short Message Service(SMS), also called text messaging, is a worldwide text messaging service component installed in most mobile devices. SMS uses standardized communication protocols to make sure every mobile device can send or receive text messages. However, until today, the safety of standardized communication protocols is no longer secured anymore, and people discovered many techniques to manipulate SMS. Sender ID spoofing is one of the most favourite techniques that is being used by many scammers.

Sender ID spoofing is a technology that can manipulate the originating phone number(also called Sender ID). Hackers can use this technology to change the originating phone number of sent messages, pretend as the sender they want, and then use some scams to spoof the recipients. However, it is hard to say Sender ID spoofing is totally illegal since there are some legitimate uses of Sender ID spoofing. For example, a company might send more than a thousand SMS messages to its customers. This process usually is done through a computer network, so the company has to use Sender ID spoofing to create a Sender ID to let its customer identify itself[sectigostore, 2020].

1.2 Motivation

Since Sender ID spoofing has become a common issue that happens in people's lives, this project aims to design and implement an SMS Android application that can provide several different authentication methods for the user to verify if the application registered the incoming message's phone number.

1.3 Goals

The primary goal of this SMS application is to help the user to authenticate incoming SMS messages efficiently. Below are the goals that I wanted to be achieved in this project.

1. The SMS application requires proper authentication to quickly tell if the SMS message is from the registered phone number.
2. Only the registered sender can send the SMS message that can pass the authentication.
3. To guarantee the incoming SMS message is from the registered phone number, the sender also needs to install the SMS application, so he can send the message that includes the information for authentication.
4. The authentication process should be fast and automatic since this is an SMS application. The user should not spend any time except texting, and the user should get the result of the authentication process directly.
5. The application should be able to receive the incoming SMS message, even if it is working on the background.

1.4 Objectives

Here will list the objectives that are needed to achieve in this project.

- The SMS application will provide two authentication methods for its user: HMAC-based One Time Password (HOTP) and Digital Signature.
- To prevent the Middle-Person attack, the HOTP authentication process has to achieve data integrity.
- For the situation that a user will send the same message to thousands of recipients, the Digital Signature will be used in this situation to make sure every recipient can verify this message successfully.

- Since the SMS channel might not be secure for hackers, it is not safe to place all the information that the authentication process needs in the SMS message. So the information will be transferred through an online database and SMS message separately.
- For the security of the authentication, the application will need the user to use a received OTP password to verify himself. So he can become a registered user.
- The HOTP and Digital Signature will be generated and used automatically in the application.

1.5 Report outline

The remainder of this report will be listed below:

- Section 2: In this section, it will present the event that was caused by Sender ID spoofing, and the reason it needs to be solved. Then, it will include a short discussion of the reason why this project tries to solve this problem using the Short Message Service (SMS) application, but not the Instant Messaging (IM) application. In the end, it will present the two critical components that are being used in this project: Firebase Realtime Database and Room Database.
- Section 3: This section will present the design choice that was made during this project.
- Section 4: In this section, it will present the development plan that was used for the project, and the design plan to show how the application works in general.
- Section 5: This section will present all of the implementations for each part of the project. Finally, it will provide the workflow diagram of the authentication and explain its process.
- Section 6: Conclusion.

2.Background

This section will introduce the background knowledge of Sender ID spoofing and why Sender ID spoofing is a vital threat, so it requires this project to solve it. Then, it will include a short discussion of the reason why this project tries to solve this problem using the Short Message Service (SMS) application, but not the Instant Messaging (IM) application. In the end, it will present the two critical components that are being used in this project: Firebase Realtime Database and Room Database.

2.1 Background of Sender ID spoofing

In the 21st Century, Sender ID spoofing is one of the problems that happened commonly in Modern SMS communication. Because of COVID-19, many people have to work from home. Under this kind of situation, people are supposed to help each other in this difficult time. Unfortunately, Sender ID spoofing becomes a convenient tool in the hacker or criminal's hand. Recently, the Australian Cyber Security Centre had published a social media warning about the SMS scam that happened in Australia[Message media, 2020]. The SMS scam victims received an SMS message, and this message claimed it was sent from the government service. This message encourages the user to click the website link that contained in the message and told the victims that the website would help them to understand better about COVID-19. It will install a malicious application into the user's phone secretly, and this application will send the information(which is stored in the phone) to the criminal. From this example, it is not hard to see how much the problem that Sender ID spoofing can cost.

2.2 The reason it is needed to solve

Sender ID spoofing is a vital threat because the user of this technology can simply pretend as anyone by changing his Sender ID (phone number), and this process will not cost him anything. The most important thing is the hacker might use a real person's phone number as the Sender ID, and the real owner can not know that.

Even though Sender ID spoofing might cause many problems, it is hard to say that Sender ID spoofing is an illegal technology that needs to be banned. Because there are many legitimate usages of Sender ID spoofing[SendSMS, 2019], for example, the public organization will require the recipient to quickly figure out the identity of them without spending time to verify the phone number is correct.

Because of Sender ID spoofing has both the illegal usage and the legitimate usage, it can not be stopped entirely, but the user of SMS should have a solution to help them to identify the Sender ID, so they will not be spoofed by the hacker and can still trust the innoxious message that sends from the public organization or other people.

2.3 The reason to use this SMS application

The reason that Sender ID spoofing happens very commonly is the protocol of Short Message Service (SMS) is no longer secure to the hacker. Every hacker can easily manipulate the sent message's originating address to achieve Sender ID spoofing. One of the methods to prevent the Sender ID spoofing is to use TCP (Transmission Control Protocol) / IP (Internet Protocol) chat application to replace all of the SMS applications. But that is not a suitable choice since all of the messages that are sent by public organizations are through SMS messages and all of the organizations required to use the conventional phone for backward compatibility. Because of those reasons, it is hard to ignore the requirement of those public organizations or service numbers, so an SMS application is developed for this project. For the safety of this SMS application, it will ask the user to use the received OTP for registration, then the user can be trusted by other users. But this process is not necessary, since the user can still verify other users without being a registered user.

2.4 Firebase Realtime Database

Before introducing the Firebase Realtime Database, it is necessary to know what Firebase is. Firebase is a powerful platform that can handle the backend online element for the developer's mobile or web application. This platform makes the developer focus more on the front-end, and without worry about the back-end server. Firebase provides many different features for the developer, and Firebase Realtime Database is one of them. Firebase Realtime Database is a cloud-hosted database, and it uses the document-oriented database as its database model, instead of the relational database. This database will store every uploaded data as JSON and synced data to every connected application client in real time[Firebase, 2020]. As long as the applications are connected to the Firebase Realtime Database, all of the clients share one Realtime Database instance and receive updates asynchronously. Except that, the Firebase Realtime Database also provides Android, IOS and javascript SDK for the developer to implement a cross-platform application.

2.5 Room Database

Room persistence library is an Object Relational Mapping (ORM) library that can map the database object to Java object easily[developer, 2020]. It provides an abstraction layer over SQLite to use all of the functionality of SQLite. Meanwhile, it has optimized the SQLite database, so the developer can easily find out the problem when they did something wrong on the SQL queries.

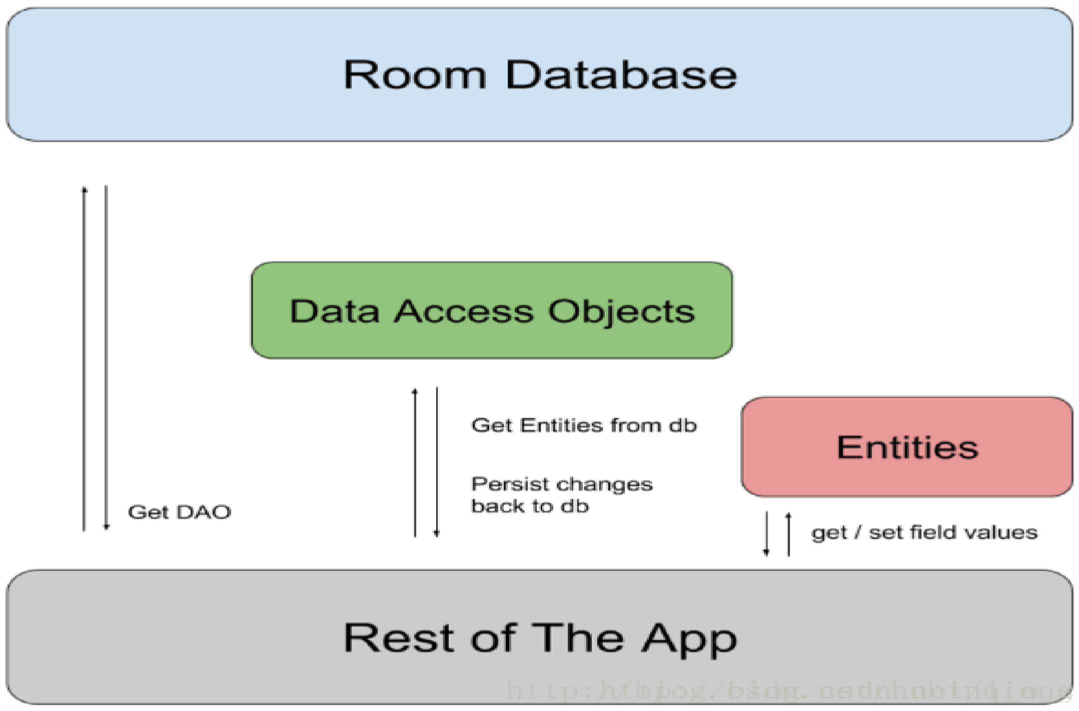


FIGURE 1. Room architecture diagram[developer, 2020]

Room database has three different components:

- Entity: It is a table within the database.
- Data Access Objects (DAOs): It is an object that defines the methods used to access the Entity.
- Database: It represents a database class. Through the database class, the user can get the DAO that is bound with this class. Then the user can manipulate the Entity through the DAO.

As long as the developer wants to use the Room database, he has to follow the rule to read or set data into the database, which is going through a database class to access a Data Access Object, then using the Data Access Object to manipulate an Entity's data.

3. Design Choice

3.1 Authentication process design

Some companies or government services have to send the same SMS messages to more than thousands of recipients, and this process usually is done through Bulk Messaging Service. The Bulk SMS service is a messaging service that can send a single message to many recipients at once. Those companies usually use Sender ID spoofing to create a number as its common number, then use the common number to send messages through Bulk SMS service. So the recipient can identify the received message from which company without remembering all of the Sender ID that the company has. Because of that, the user of the SMS application has to separate into two different groups. One group of users is the ordinary people who don't have the demand to use Sender ID spoofing and they won't need to send the same message to many different recipients. This group of users will be called “Individual users.” The other group includes companies or government services, which have the demand to send the same message to more than thousands of recipients for business usage, and this group will be called “Official user.” Except for these two types of users, the last type of user is named “Anonymous user”, and this type of user will be introduced later.

There are two different authentication methods to be considered for the two types of users. The first method is called the “HOTP authentication process.” It is to ask the sender to use an HMAC-based One Time Password (HOTP)[RFC 4226, 2005] to add to the hashed SMS message as salt and send the salted hashtag to the recipient. Then, the recipient also creates a salted hashtag by using the information received through the SMS channel and Firebase Realtime Database. If these two hashtags are the same, then the sender’s identity is verified. The second method is called the “Digital Signature authentication process.” It is to generate a public and private key pair, then using the private key to sign the hashed SMS message to create a digital signature. After the recipient receives the digital signature, he can use the public key, which gets from the Firebase Realtime Database to verify the digital signature. If the hashtag after decryption is the same as the hashed incoming message, the sender's identity is verified. Both of the two methods have their advantages and disadvantages.

- Because of the definition of the OTP (One Time Password) algorithm, the HOTP authentication process can generate a new HOTP for authentication every time. It can protect the user from the replay attack successfully.
- Since the HOTP authentication process requires a new secret every time, it is hard to achieve non-repudiation.
- The HOTP authentication process uses symmetric encryption to build the HOTP value. It will take a shorter time than the asymmetric encryption, which allows this authentication process to generate a new shared secret for every message.
- The digital signature can provide data integrity for the user of the application, so the hacker can not generate a new hashtag and sign it using his secret key.
- The digital signature authentication process requires asymmetric encryption to generate the public and private key pair. This process will take a longer time than symmetric encryption.
- The digital signature provides non-repudiation for the users. So no matter who signed the hashed message, it is hard for him to repudiate the authenticity of the messages sent by him.
- If using the digital signature as the authentication process, the application can not regenerate a new public and private key pair whenever the user sends a message. Because of the non-repudiation that the digital signature provides, the SMS application has to generate the public and private key pair when its user becomes a registered user.
- Once using a digital signature for authenticating, the sender won't need to generate many key-pair for different recipients. The key-pair will only be created once.

3.1.1 Individual user

Individual users are a group of users who do not require Sender ID spoofing for legitimate use, and they do not need to send the same SMS message to too many recipients. The most important thing to them is the authentication process for Individual users has to prevent the replay attack. Because under the specific situation, a hacker might intercept a message that was sent to an individual user. The content of the message can be "Give me the bank account number and the password." The hacker can easily copy this message and send it to the Individual user's friend or family. Because of this kind of situation, the authentication process of Individual user has to prevent replay attacks.

The HOTP authentication process is a suitable method to solve the problem. The HOTP authentication process will regenerate the shared secret every time that can easily prevent the replay attack, and this process will not spend too much time on the generation. And compared to the digital signature authentication process, the digital signature can not prevent a replay attack.

3.1.2 Official user

Official users are a group of users who will need Sender ID spoofing for legitimate use. They usually use Sender ID spoofing to create a Sender ID for them, then using the Bulk Messaging Service to broadcast an SMS message to their customers. The user who requires this usage is often a famous company or government service that wants the recipient to figure out its identity quickly. However, if the Official user wants the recipient to identify them quickly, the proof he provides has to achieve data integrity and non-repudiation. The reason for that is most of the Sender ID spoofing scams are pretend as bank phone numbers and government service's phone numbers. By achieving data integrity, the authentication process can bind the information for authentication with the original SMS message, so the hacker can not replace the original SMS message sent by the Official user during message transfer. By achieving non-repudiation, the authentication process can guarantee the incoming message is from the Official user. Except for the data integrity and non-repudiation, Official users also have the demand to send a single message to many different recipients. So this type of user needs an effective and cost-efficient solution for the authentication process.

In the two authentication processes, the HOTP authentication process can also achieve data integrity, but it is hard to achieve non-repudiation. So it is not suitable for the Official user. Compared to the HOTP authentication process, the digital signature authentication process can achieve data integrity and non-repudiation simultaneously. But most importantly, the HOTP authentication process is hard to deal with sending a single message to many different recipients since it will cost too much space to save the shared secret that used to verify HOTP. Instead, the digital signature can easily solve this problem since it can use the same secret for many different recipients.

3.1.3 Anonymous User

It is impossible to force every user to register and become a verified user because some of the users might not worry about somebody pretending to be them and don't want to spend time on the registration. The only thing they are concerned about is getting scammed by others. So this type of user will be named as "Anonymous users". It is not necessary to let them register an account and become the registered user, but it is also dangerous to let them upload or send the same type of information as other registered users (i.e the Individual and Official user) since we can not identify them. So for the Anonymous user, they can only send the SMS message that does not include any information for authentication, but they can still verify the incoming message they received from others.

3.2 Implementation

In the following parts, the design choices of the implementation will be introduced.

3.2.1 Programming Language

Many different programming languages can be used for this Android application project. Three selected programming languages will be discussed in the following.

The first programming language is Kotlin, and it is a cross-platform programming language designed to replace Java for Android application development. It removed many features that are not very useful in Java. For example, the null pointer exception. Kotlin is a more comfortable and more straightforward programming language compared to Java. Since it removed some features of Java, Kotlin can not use constructor, concurrency and checked exceptions. Unfortunately, some of the features are necessary for this project.

The second programming language is C++. The developer has to use the Android Native Development Kit(NDK) for Android application development. NDK can support C++ to implement the Android application and let the app can access C++ libraries when the application needs the feature of C++. But, C++ is challenging to set up for Android application development, and it might cause more bugs because of its complexity. So C++ is not the right choice.

The third programming language is Java. It is the official language for Android application development and the most supported programming language by Google. Since this project requires the Firebase Realtime Database and Room Persistence Library, which both provided by Google, Java is a better choice than the two programming languages above.

3.2.2 Online Storage & Local Storage

Since the application has to receive a separate secret from two different communication channels, the project expects two different communication channels not to get involved in each other. Since this project is about Sender ID spoofing, one of the communication channels is Short Message Service(SMS) channel, and another communication channel is an online database that only the user of the SMS application can access.

3.2.2.1 Online Storage

There are two options for the online database: Firebase Realtime Database[Firebase, 2020] and PostgreSQL. Since these two databases will face the same task, the features of these two databases will present below.

Firebase Realtime Database:

- It is a cloud-hosted real-time database that can automatically receive updates from clients.
- The primary database model is the document store, which stores every new information as a leaf of a tree. This kind of model can also be called the NoSQL database.
- It allows the developer to connect their application with itself without the back end server.
- It provides an authentication method for the user to access its database.
- When creating a cross-platform application, all clients can share the same realtime database instance to receive the update of the newest data.

PostgreSQL database:

- It is a widely-used open source Relational Database Management System(RDBMS).
- The implementing language is the C language.
- It requires the developer to set up a back-end server so that he can connect the database to the developer's application through the back end server.

Through the comparison between the Firebase Realtime Database and the PostgreSQL database, it is easy to see that PostgreSQL has a better query structure than the Firebase Realtime Database since the Firebase Realtime Database is a NoSQL database. But this project does not require complex queries for searching data, so the query structure is not an essential issue for the project. Since the main idea of the project is to implement an SMS application, then the Firebase Realtime Database is a suitable choice since it can connect the Firebase Realtime Database with many different platforms, such as Android, IOS and Javascript SDK.

3.2.2.2 Local Storage

There are also two different options for the local storage, which are used to store the received SMS message. Because the local storage will save another part of the shared secret, it is necessary to separate the shared secret from the message box that exists in the Android system. One of the local storage options is SQLite, and the other option is Room persistence library[developers, 2020]. Now the features of the two different options will be listed.

SQLite:

- SQLite is open-source and does not require any server for its operation.
- SQLite is a cross-platform DBMS that can be used on many different operating systems.
- SQLite has an excellent performance in reading and writing operations.
- SQLite is a light weighted database, and it can be embedded in many devices, such as mobile devices.
- SQLite will always update the content, so the user will not instantly lose all of their current work when their device or phone is dead.

Room persistence library:

- The Room is an Object Relational Mapping library that can easily map the database object to Java object.

- The Room persistence library provides an abstraction layer over SQLite to use all of the functionality of SQLite.
- The Room database can use LiveData and RxJava to achieve data observation, which can easily observe the change of the database and update the changes to the Android application instantly.

Comparing the features of SQLite and Room persistence library, it is not hard to see the Room persistence library can provide all of the features that SQLite can afford since it allows an abstraction layer to cover all the features of SQLite. Besides, the Room database can achieve data observation, so the application which connects with the Room database can see the changes without refreshing the page. Thus, the Room persistence library is a better option compared to SQLite.

3.3 Summary

In this section, every useful technology is evaluated and compared. The required technology for this project will be listed below.

- The SMS application users have to separate into two different groups since one group may need to send the same SMS message to many different recipients, but the other group doesn't have this concern.
- For the user who does not require the usage of Sender ID spoofing, the application needs to provide the HOTP authentication process for this kind of user. To this type of user, the replay attack is a bigger problem than the non-repudiation.
- For the user who has legitimate use of Sender ID spoofing, the application needs to provide the Digital Signature authentication process for this kind of user. Because non-repudiation is more necessary to this type of user, so the user can guarantee nobody can pretend as them by using Sender ID spoofing. More importantly, the digital signature can guarantee the sender only has to generate one shared secret for all of the recipients, so all of the recipients can verify the incoming message successfully when they receive the same message.

- Java will be used to implement the application because it is a better choice to implement other technology into this application than the other programming languages. And Java can also provide all the library and functions that Android application development needs.
- Since Firebase Realtime Database can be cross-platform and does not require the developer to set up the back-end server, it will be used as one of the communication channels for the shared secret.
- The Room database will be used for storing the shared secret that gets from the incoming SMS messages.

4. Project Management

4.1 Development plan

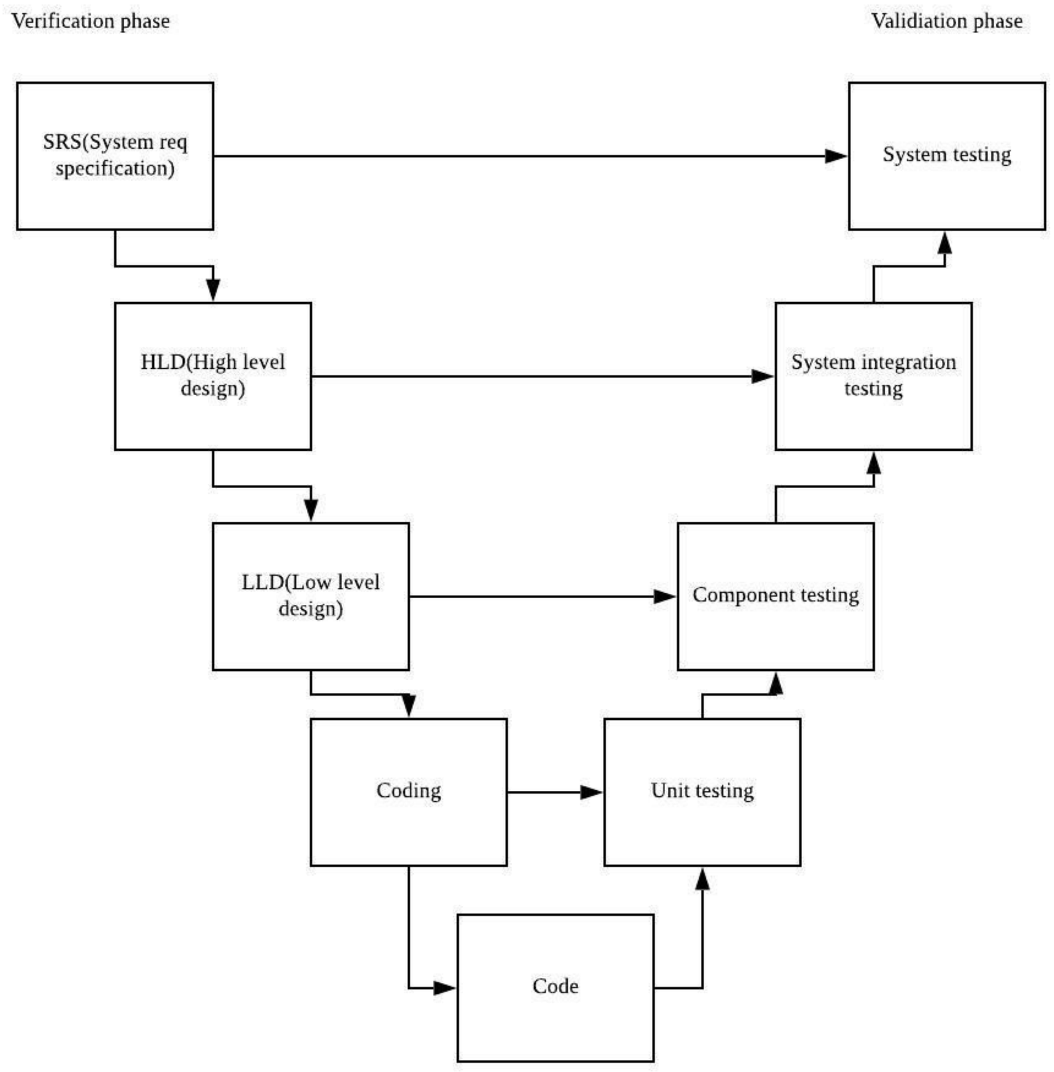


FIGURE 2 : V Model

Before the presentation of implementation, the plan of the development process will be introduced. In the project, the V model(see FIGURE 1) has been chosen as the development model of this project.

At the start of the project, it is necessary to understand the main task that requires this project to solve. By following this problem, the developer will have clear objectives of what should be done to solve the problem. Those objectives are the requirements of the project, and the developer can see the requirements as the subtasks of the main task. Once the main task has been broken down, the developer can figure out every subtask's possible solution through online research. By finding all of the solutions of every subtask, the developer can follow all the possible solutions to build the actual software components.

To guarantee the application's performance, the developer has to test each of the components to check if the app can perform correctly. By doing this process, the developer can find the component's problem at the early stage, and it can prevent the issues from getting mixed when the new component is added into the testing.

The reason for following the V model to develop the application is that it can help the developer find out the design problem when the developer just creates. Doing this can save a lot of time for the debugging and make sure the developer won't need to change the whole application if he finds out the design is not possible.

4.2 Design plan

The following will present a workflow diagram to show how the SMS application worked in general.

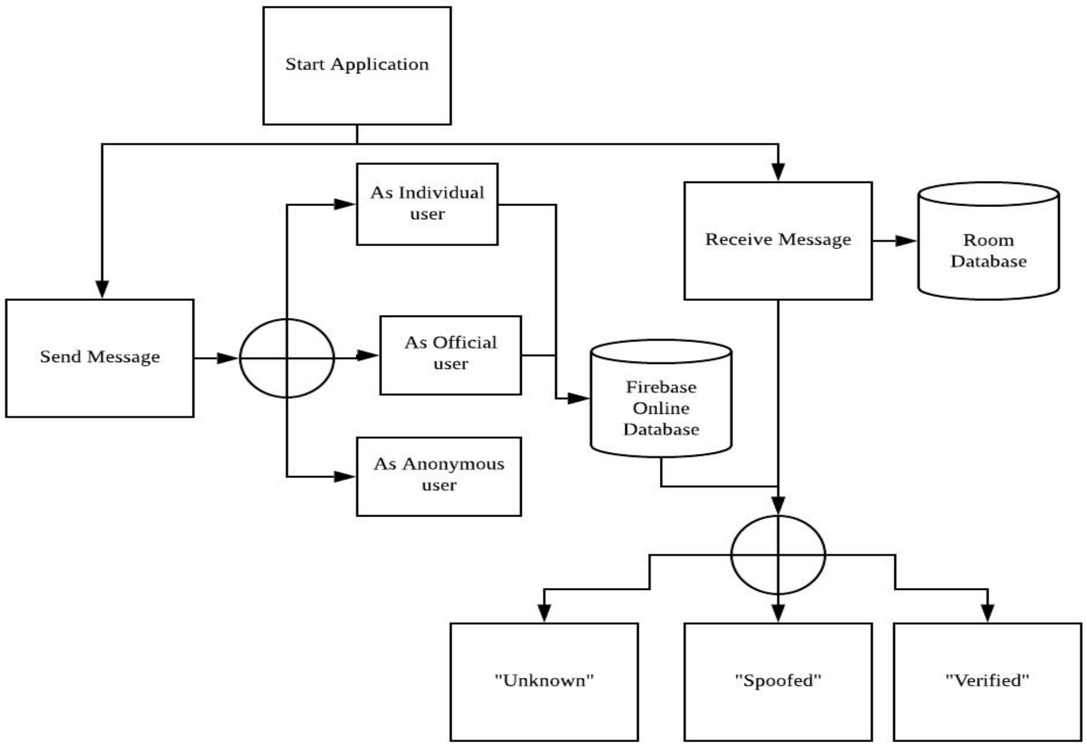


FIGURE 3. Workflow Diagram of the SMS application

Here will explain how this application works. This application will have two general features as send and receive messages, since it is an SMS application.

- Sending messages: Only three types of user can use this SMS application,which are the Individual user, Official user and Anonymous user.
    1. **Individual user / Official user:** If the current user is an Individual user or Official user, he will follow the HOTP authentication process or the Digital Signature authentication process to send the information for authentication to the recipient and the online database.

    2. **Anonymous user:** If the current user is an Anonymous user, then he will not generate anything for authentication, or upload anything to the online database. The only thing he is allowed to do is send a message which does not have any attachment.

- Receive messages: For the user who wants to verify the incoming message, the identity of the user is not important. So every user can retrieve the data they want from the online database, but can not upload unless they are a registered user. Through the HOTP and digital signature authentication process, the SMS application will be the result of the authentication as either “Verified”, “Spoofed” or “Unknown”.
    1. “Verified”: The incoming message has passed the authentication successfully.
    2. “Spoofed”: The incoming message can not pass the authentication.
    3. “Unknown”: The incoming message does not include any attachment for authentication, or can not find the sender of the incoming message from the online database.

In the SMS application, the Firebase online database is a ‘mailbox’ for the user to store information for authentication, and the Room database is for the user to store the received and sent messages for the UI performance.

5. Implementation

From the previous discussion, the technologies that the project required had been decided. Regarding the Sender ID authentication, the HOTP authentication process will be used for the Individual user, and the digital signature authentication process will be used for the Official user. Regarding setting up the online and local storage to save the information needed for authentication, the Firebase Realtime database will use it for storing a part of the information, and the Room database will use it for storing the other part of the information that received from the incoming SMS message. Since either the authentication process or the online/local storage is not necessary until the final stage, these two parts can be implemented separately.

5.1 Implement the Sender ID authentication

The implementation should start from the Sender ID authentication. It is the main focal point of this project, so it is necessary to implement and test if this authentication process can work correctly.

5.1.1 HOTP authentication process

5.1.1.1 Requirements

In the HOTP authentication process, two elements are required. The first requirement is to generate the HOTP by following the standard HOTP algorithm. Then in cases to provide data integrity for the authentication process, the second requirement is to guarantee the shared secret or the content of the SMS message will not get changed during the message transferring.

5.1.1.2 Implementation

Following the previous design, this authentication process should be able to do the following requirements:

- To follow the standard HOTP algorithm to generate a HOTP value for authentication purposes.
- The HOTP authentication process should provide data integrity for the authenticated message.

The HOTP algorithm is the abbreviation of the HMAC-based One Time Password algorithm. It uses a random event value and a secret seed value as the input, then calculates a consistent password through the Hash-based MAC (HMAC) algorithm. Because of the HOTP algorithm's standard, only the HMAC-SHA1 algorithm is capable of being used in the calculation of the HOTP value.

- ***K, C = OTP (K: secret seed   C: random event value)***
- ***HOTP(K, C) = HMAC-SHA1(K, C)***

To ensure the HOTP value's randomness, the K value and the C value have to be the pseudo-random number, so the best option is to create them as the OTP value. Since both the K and C values will be generated and used automatically by the system, these two values will not be limited too soon. But only the C value should be limited since it will be sent as a shared secret. Based on the GSM protocol, a single SMS message can only contain 160 seven-bits characters. If the SMS message's length is longer than 160 characters, then this SMS message will be separated into two SMS messages, and the sender will cost twice for sending the SMS message. So the C value should not be too long to make the user cost twice for one SMS message. For now, the C value is limited to 10 seven-bits characters.

After the computation of the HMAC-SHA1 algorithm, the length of the output message will be 20 bytes. But since the authentication process does not require the user to create any input unless the original SMS message, so the truncation method is not necessary for this authentication process. Doing this can increase the HOTP value's complexity, and the hacker won't be able to use a brute force attack to crack the HOTP.

Here is the presentation for the workflow of the HOTP and the salted hashtag's generation:

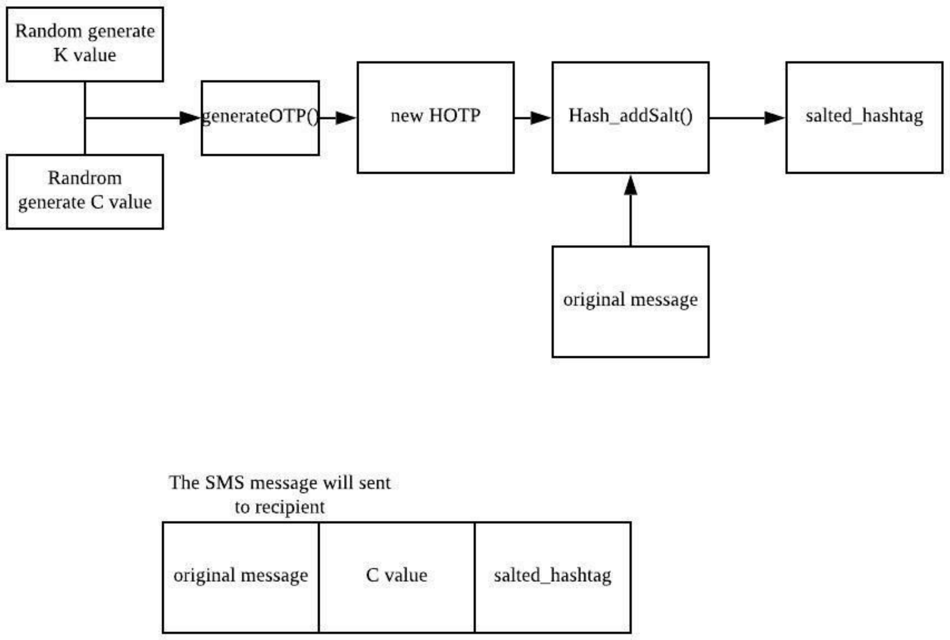
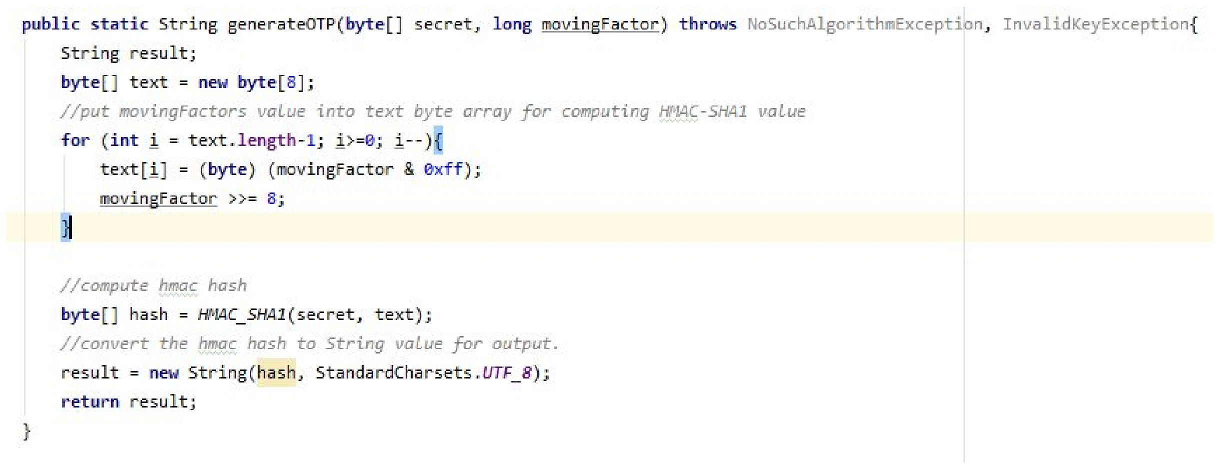


FIGURE 4. HOTP and salted hashtag's generation process

The function generateOTP() performs the HOTP generator's functionality, and the code is looked as follows.

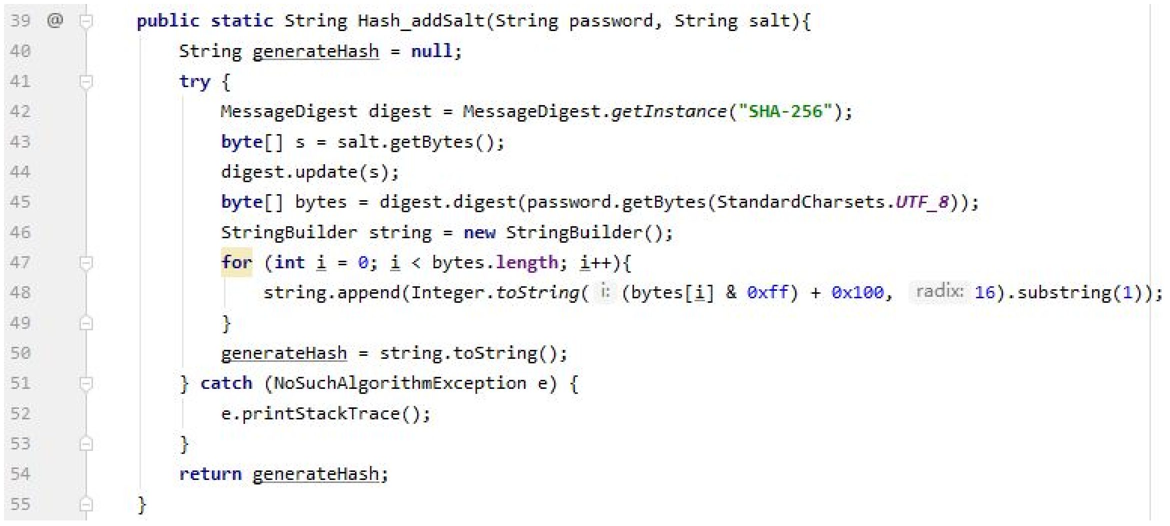


```java
public static String generateOTP(byte[] secret, long movingFactor) throws NoSuchAlgorithmException, InvalidKeyException{
    String result;
    byte[] text = new byte[8];
    //put movingFactors value into text byte array for computing HMAC-SHA1 value
    for (int i = text.length-1; i>=0; i--){
        text[i] = (byte) (movingFactor & 0xff);
        movingFactor >>= 8;
    }

    //compute hmac hash
    byte[] hash = HMAC_SHA1(secret, text);
    //convert the hmac hash to String value for output.
    result = new String(hash, StandardCharsets.UTF_8);
    return result;
}
```

FIGURE 5. generateOTP(). Use the HMAC-SHA1 algorithm for the generation of HOTP value.

The HOTP algorithm is called the HMAC-based One Time Password algorithm, and it will generate a one-time password called HOTP value. Because the hacker can change the message's content when he intercepts it, the HOTP authentication process will use the auto-generated HOTP value as salt and add it to the hashed original message. Then the HOTP authentication process can guarantee data integrity, and only the person who can compute the HOTP value can pass the authentication.



```java
public static String Hash_addSalt(String password, String salt){
    String generateHash = null;
    try {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] s = salt.getBytes();
        digest.update(s);
        byte[] bytes = digest.digest(password.getBytes(StandardCharsets.UTF_8));
        StringBuilder string = new StringBuilder();
        for (int i = 0; i < bytes.length; i++){
            string.append(Integer.toString( i: (bytes[i] & 0xff) + 0x100,  radix: 16).substring(1));
        }
        generateHash = string.toString();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return generateHash;
}
```

FIGURE 6. Hash_addSalt(). Use this function to add the HOTP value into the hashed SMS message as a salt.

In this Hash_addSalt() function, it has accomplished the following tasks:

- Line 42: Get the MessageDigest instance to use the SHA-256 algorithm.
- Line 43: add the received salt to input through the MessageDigest's update() function.
- Line 45: generate the salted hash by adding the salt into the hashed message.
- Line 46 to 50: convert the byte array to a String object for the output.

When accomplished all of the tasks presented above, the HOTP authentication process is fully functional. The salted hash will then be sent to the recipient for data integrity. Meanwhile, the C value will also be attached to the original SMS message, so the content of the SMS message is looked as follows: (the original content of the SMS message)(the C value)(the salted hashtag).

5.1.2 Digital Signature authentication process

5.1.2.1 Requirements

In the digital signature authentication process, there is only one requirement. This requirement also follows the standard algorithm to set up the public and private key pair.

5.1.1.2 Implementation

The first things that should be implemented are the public and private key pair. The digital signature authentication process is built for a public organization. So when the organization sends the same message to many different recipients, all of the recipients won't have any trouble for the authentication when they receive the same message at the same time.

The next thing is to decide which asymmetric cryptosystem shall be used in the digital signature authentication to set up the public and private key pair. The RSA cryptosystem[RFC 3447, 2003] is the better and more common choice for this project because of security and portability. In the RSA cryptosystem, the generation of the key pair is based on integer factorization. The details of the key pair generation will list below.

1. Randomly choose two different prime numbers(**p** and **q**) from a large group of numbers. The digital length of the two prime numbers should not be similar so that it can increase the complexity of factorization.
2. Calculate a value **n** which **n = p * q**. It is used for the public and private keys as their modules.
3. Calculate the totient as **$\varphi(n) = (p-1)(q-1)$**
4. Randomly select an integer e which fits the following requirement: **$1 < e < \varphi(n)$** and e is co-prime to **$\varphi(n)$**
5. Then compute the value **d** which fits **$ed \equiv 1 \pmod{\varphi(n)}$**
6. By going through these processes, the public key consists of two components as **(n, e)**, and the private key consists of two components as **(n, d)**.

Because of the digital signature usage, one of the keys should be shared with others for authentication purposes. So that key will upload to the online database for the authentication when the recipient receives a message which contains a digital signature. Since this project requires the digital signature to provide the non-repudiation for the Official user, the Official user will be forced to upload the key pair for themselves when they register for the application. Once the registration is finished, the app will upload one key to the online database for verification and save one key for signing messages. Then the application will not request the Official user to generate or enter the key pair anymore.

Here is the presentation for the workflow of the digital signature generation:

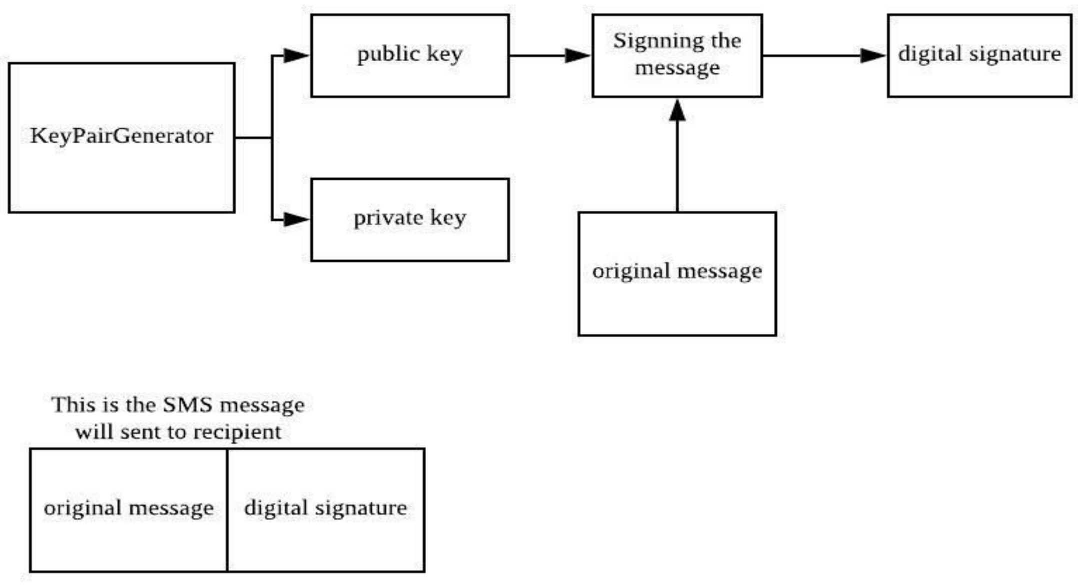
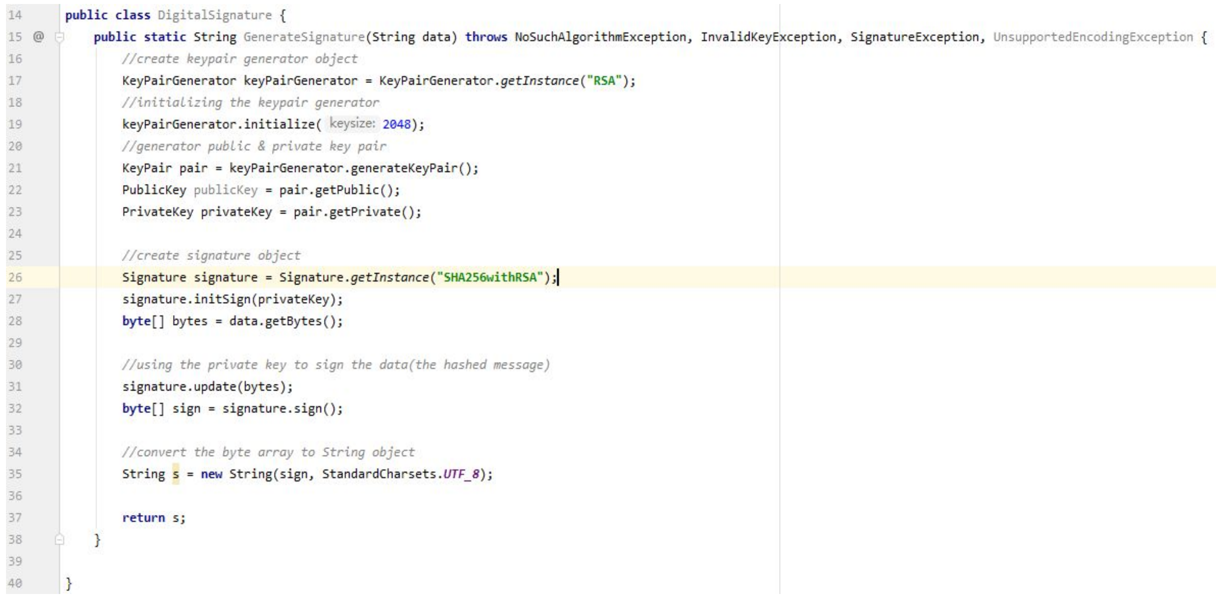


FIGURE 7. Digital Signature generation

Fortunately, Java has an abstract public class called KeyPairGenerator, and it supports the generation of pairs of public and private keys[1]. It gives the option to use the KeyPairGenerator class to generate the public and private keys. Here will present the generation of the public and private keys using KeyPairGenerator.



```java
public class DigitalSignature {
    public static String GenerateSignature(String data) throws NoSuchAlgorithmException, InvalidKeyException, SignatureException, UnsupportedEncodingException {
        //create keypair generator object
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
        //initializing the keypair generator
        keyPairGenerator.initialize( keysize: 2048);
        //generator public & private key pair
        KeyPair pair = keyPairGenerator.generateKeyPair();
        PublicKey publicKey = pair.getPublic();
        PrivateKey privateKey = pair.getPrivate();

        //create signature object
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(privateKey);
        byte[] bytes = data.getBytes();

        //using the private key to sign the data(the hashed message)
        signature.update(bytes);
        byte[] sign = signature.sign();

        //convert the byte array to String object
        String s = new String(sign, StandardCharsets.UTF_8);

        return s;
    }

}
```

FIGURE 8. Key pair and digital signature generation.

In the GenerateSignature() function, it has accomplished the following tasks:

1. Line 17: Create the KeyPairGenerator instance to use the RSA algorithm.
2. Line 19: Initializing the KeyPairGenerator object and setting its key size to 2048.
3. Line 21 to 23: Generate the pair of public and private keys through the built-in algorithm from the KeyPairGenerator.
4. Line 26: Create the Signature object for the generation of a digital signature.
5. Line 27: Update the private key as the signing key into the Signature object.
6. Line 31 to 35: Use the updated private key to sign the message and convert it to a String value for the output.

When accomplished all of the tasks presented above, the digital signature authentication process is fully functional. Since only the different parts of the pair can verify the digital signature, the recipient is encouraged to trust the message that can pass the digital signature verification. The content of the sent message will be: (the original content of the SMS message)(digital signature)

5.2 Implement the data storage for online and offline

5.2.1 Online database: Firebase Realtime Database

5.2.1.1 Requirements

There are two requirements that Firebase Realtime Database needs to achieve, the first one and the most important one is to guarantee only the authorized user can write data to the Firebase Realtime Database. If everyone can easily write data to the online database, there will be no security in the online database, and the whole authentication process is not trust-worthy. The second requirement is the online database shall have a concise structure to store the user's information and the information that will be used for Sender ID authentication.

5.2.1.2 Implementation

To solve the first requirement, the Firebase Realtime Database shall have the ability to identify if the access request is from the clients. Fortunately, Firebase already has a built-in authentication method called Firebase Authentication. It provides the Firebase's backend server for the user authentication and can support the phone number authentication by sending verification id to the user through SMS. The Firebase authentication is connected with the Firebase Realtime Database, so it is easy and flexible for the user who wants to access the Firebase Realtime Database.

The Firebase Authentication can provide many different credentials for users who want to access the Firebase Realtime Database. All of the credentials follow the industry standards like OAuth 2.0 and OpenID Connect. The credentials can be summarized as the following:

- Email and password-based credential
- Phone number based credential
- Anonymous credential
- Federate identity provider integration(Google, Facebook, Twitter, Github, etc.)

In all of the credentials, the selected authentication method is the anonymous authentication and the phone number-based authentication. The phone number based credential can simply authenticate the user who wants to gain the writing access of the Firebase Real-time Database when he wants to register to become a verified user. Only the user who can receive the authentication message from the registered number can prove himself is the owner of the phone number. Meanwhile, phone number-based authentication provides non-repudiation for the registered user, since only this user can get the verification message if the hacker did not intercept the verification message.

It is impossible to force every user to register and become a verified user because some of the users might not worry about somebody pretending to be them. The only thing they are concerned about is getting scammed by others. It is dangerous to let this type of user also upload some data to the Firebase Realtime Database. Because this project can not identify a person without having any of his information, and the possibility of the user being a hacker is pretty high. The solution to this problem is anonymous authentication. The anonymous authentication does not require any information from the user. Still, it is only a temporary solution for the anonymous user (the user who uses anonymous authentication to register) to access the Firebase Realtime Database. Because the developer can set up a lifetime limit for the anonymous users, so they won't always have access. Every installed application will be considered as a registered user by letting every installed application use anonymous authentication to access Firebase. Every user will be controlled by the access rule of the Firebase Realtime Database. In the end, the developer can make every anonymous user only read data from the Firebase Realtime Database to achieve the first requirement. Here is the diagram of how these two types of users can access the Firebase Realtime Database.

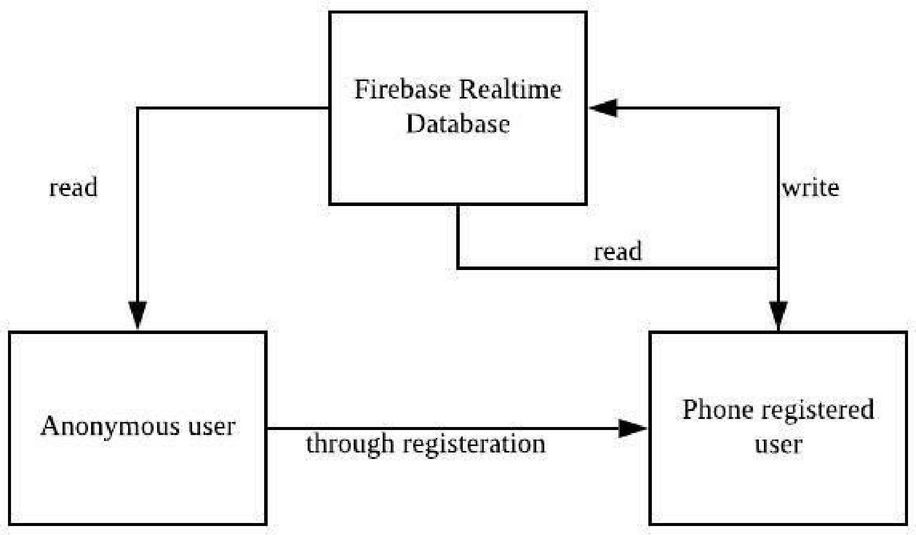


FIGURE 9. The relationship between the online database and the users

The Firebase Realtime Database shall have a concise structure for storing data. The whole database should include two different groups of data for the Individual user and the Official user.

Since the two different types of users use the different authentication processes to prove their identity, the table structure will be present separately.

Table: User

Firebase_UID (primary key)	Phone number

User table is the table that stores the information of the Individual user in the online database. Here is the detail explanation of the table:

- Firebase_UID is the primary key. It is a unique ID created by the Firebase server, and the Firebase uses this id to identify all the actions in the Firebase server.
- The phone number represents the phone number of the registered user.

Table: User_Official

Firebase_UID (primary key)	PublicKey_modulus	PublicKey_exponent	Phone number

User table is the table that stores the information of the Official user in the online database. Here is the detail explanation of the table:

- Firebase_UID is the primary key. It is a unique ID created by the Firebase server, and the Firebase uses this id to identify all the actions in the Firebase server.
- The PublicKey_modulus is the modulus of the public key, and it can be used to build the public key with the PublicKey_exponent.
- The PublicKey_exponent is the exponent of the public key, and it can be used to build the public key with the PublicKey_modulus.
- The phone number represents the phone number of the registered user.

To access the user of the Firebase Realtime Database user easier, all of the information for authentication will be stored in the Chat table, except the anonymous user, because the anonymous users are not able to write their data to the Firebase Realtime Database.

Table: Chat

Sender_Phonenumber (primary key)	Secret	Receiver_Phonenumber	status

Chat table is for the two types of users to store their information for authentication. Here is the detail explanation of the table:

- Sender_Phonenumber is the primary key of the table, and it represents the phone number of the user who sent the message.
- The Secret is the K value used to generate the HOTP value when the sender is an Individual user. The Secret is the public key to verify digital signature when the sender is an Official user.
- Receiver_Phonenumber is the recipient's phone number.
- Status is the value to define if the value has been read or not, and it will not let another user access when one user has read the seed.

5.2.2 Local database: Room Database

5.2.2.1 Requirements

The local database only requires the local database to have an excellent performance for the data storing and data loading. Also, the application should be able to display any change in the local database instantly.

5.2.2.2 Implementation

Based on the requirement, the application will use a software architectural pattern called Model-view-ViewModel(MVVM)[Wintellect, 2014]. It separates the software development between the graphical user interface(GUI) and the back end logic and lets the ViewModel become a value converter. So the software can easily transform the data from the back end and present it to the user through GUI. Here is the relationship diagram of how the MVVM is used in the application.

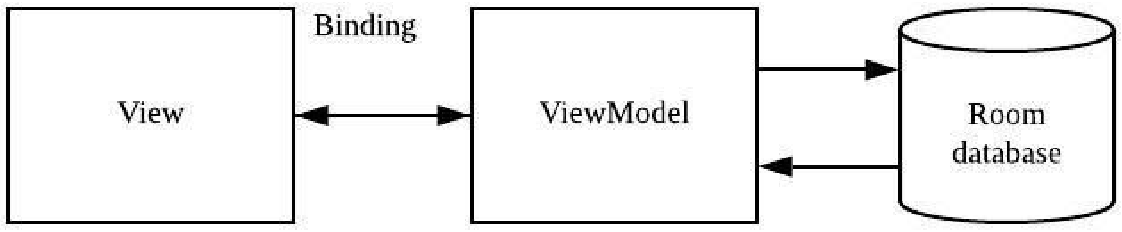


FIGURE 10. Model-View-ViewModel

The Room database will be used to store the information from incoming SMS messages, and all of the messages will be loaded into the Room database directly through the message box. The structure of the Room database should have different tables to store the information. The Room database will require a table to display all the numbers that have sent a message to the current device for a better user experience of the SMS application. All of the implemented tables will list below.

Table: ChattedUser

Chat_id(primary key)	Phone number

ChattedUser table is to store the phone number that had contacted the current user. Here is the detail explanation of the table:

- The Chat_id is the number to order the phone number that has contacted the current user.
- The phone number is the Sender ID of the incoming SMS message.

Table: Localchat

conversation_id(primary key)	message	sender	receiver	identity	status

Localchat table is to store the incoming message and the shared secret that was received by this mobile device. Here is the detail explanation of the table:

- The conversation_id is to identify the received SMS message.

- The message is the content(which includes the information for authentication) of the SMS message that exists in the message box.
- The sender is the Sender ID of the SMS message that exists in the message box.
- The receiver is the designation address of the SMS message that exists in the message box.
- Identity is the result of the authentication process.
- The status is to identify if the current has been loaded to the Room database or not.

Table: LocalUser

Phone Number (Primary key)	private_modulus	private_exponent	status

LocalUser table is to store the Official user’s private modulus and exponent for generating digital signatures. Here is the detail explanation of the table:

- Phonenumber is the current mobile device’s phone number.
- Private_modulus is the modulus of the private key, and it will be used to generate digital signatures with the private_exponent.
- Private_exponent is the exponent of the private key, and it will be used to generate digital signatures with the private_modulus.
- Status helps the application to define if the current user is the Individual user or Official user or none.

5.3 The Workflow of the authentication process

By finishing the implementation of all the subtasks, all of the small components that were created can be used to build a complete application. The following workflow diagram will present how the authentication process works in the SMS application.

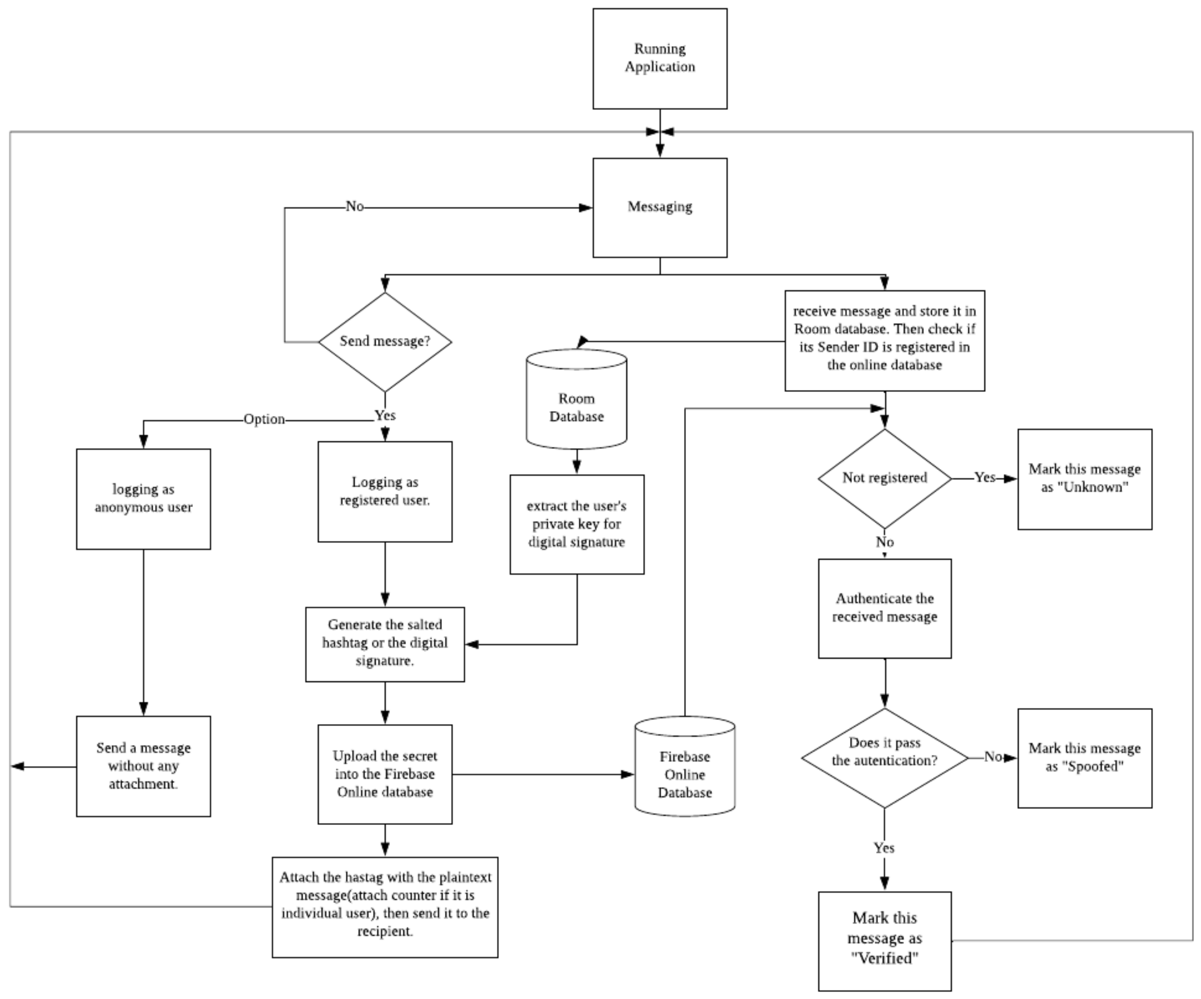

Running Application
Messaging
No
Send message?
Yes
Option
logging as anonymous user
Logging as registered user.
Send a message without any attachment.
Generate the salted hashtag or the digital signature.
Upload the secret into the Firebase Online database
Attach the hastag with the plaintext message(attach counter if it is individual user), then send it to the recipient.
Room Database
extract the user's private key for digital signature
Firebase Online Database
receive message and store it in Room database. Then check if its Sender ID is registered in the online database
Not registered
Yes
Mark this message as "Unknown"
No
Authenticate the received message
Does it pass the autentication?
No
Mark this message as "Spoofed"
Yes
Mark this message as "Verified"

FIGURE 11. Workflow diagram of the authentication process

In this application, the user shall decide what he wants to do at the start. Follow his decision. The application will provide different methods for him.

If the user wants to send a message:

- If the user is the anonymous user.
    1. He sends the message, and this message will not have any attachment.
- If the user is the Individual user.
    1. The application will generate two random values called C value and K value for creating HOTP.
    2. After generating a HOTP value, the application will hash the message typed in by the user, then add the HOTP into the hash as a salt.
    3. Then upload the K value to the Chat table in Firebase Realtime Database and attach the original message with C value and the salted hashtag.
    4. Send the original message to the recipient with its attachment.
- If the user is the Official user.
    1. Once the Official user's account is created, the SMS application will generate a pair of public and private keys, and this generation will not happen again.
    2. Using the saved private key (which pre-saved in the Room database) to sign the hashed original message to create a digital signature
    3. Attach the digital signature with the original message, then send it to the recipient.

If the user receives a message (Since all of the users can read data from the online database, the receiver's identity will not be considered a problem in this project)

- If the message is sent from an anonymous user(does not contain a specific value).
    1. Since anonymous users' messages will not include any specific information, this message will be marked as unknown.
- If the message is sent from an individual user(contains specific value).
    1. The message contains the C value and salted hashtag, then search the online database for the K value.
    2. After getting the K value, using K and C values to compute HOTP, then add it into the hashed original message as a salt.
    3. Compare the two salted hashtags, if the two hashtags are the same, mark "Verified." Otherwise, mark "Spoofed."

- If the message is sent from an Official user(contains specific value).
    4. The message contains the digital signature, then searches the online database for the public key.
    5. After getting the public key, using the public key to verify the digital signature to get a hashtag, then hash the original message using the same hash function.
    6. Compare the two hashtags, if the two hashtags are the same, mark “Verified.” Otherwise, mark “Spoofed.”

6. Conclusion

By finishing all of the subtasks, the project can perform the designed feature successfully. But it still can be improved to be a better solution for Sender ID spoofing by implementing this application in other platforms such as IOS and Windows. In fact, it is pretty interesting to solve a real-life problem by using the knowledge I learned. This project also provides for me an excellent chance to learn what I did not know.

References

[Message Media, 2020], "**3 COVID-19 (Coronavirus) SMS Scams To Look Out For**", https://messagemedia.com/us/blog/3-covid-19-coronavirus-sms-scams-to-look-out-for/, Retrieved 30 July. 2020.

[SendSMS, 2019], "**SMS spoofing: What is It and How to Be Secured?**", https://sendsms.global/blog/sms-spoofing/, Retrieved 30 July. 2020.

[Firebase, 2020], "**Firebase Realtime Database**", https://firebase.google.com/docs/database, Retrieved 30 July. 2020

[RFC 4226, 2005], "**HOTP: An HMAC-Based One-Time Password Algorithm**", https://tools.ietf.org/html/rfc4226, Retrieved 30 July. 2020.

[developer, 2020], "**Save data in a local database using Room**", https://developer.android.com/training/data-storage/room/, Retrieved 30 July. 2020.

[RFC 3447, 2003], "**Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1**", https://tools.ietf.org/html/rfc3447, Retrieved 30 July. 2020

[Wintellect, 2014], " **Model-View-ViewModel (MVVM) Explained**", https://www.wintellect.com/model-view-viewmodel-mvvm-explained/, Retrieved 30 July. 2020.

[sectigostore, 2020], "**What is SMS spoofing & How Can You Prevent It?**", https://sectigostore.com/blog/what-is-sms-spoofing-how-can-you-prevent-it/, Retrieved 06 August, 2020

List of Figures