**CS6348 Group Project Report**

Jin Liu, Ashutosh Agrawala, Hao Fu, Gandhar Satish Joshi

**Introduction**

In the ever-growing age of social media, it is highly important and imperative that all the users are assured that the metadata or sometimes the actual data collected by their respective social media platforms are not being used in a way that would violate their privacy and integrity. The major focus in terms of data privacy and security should be in private chat rooms between two individuals. It is highly important that the platform should not extract any information from the chat messages and all parties apart from the two users should never have exposure to these chat messages. Various allegations have been made on tech giants like Facebook and Google that they tried to hoard data and used them for marketing and generating revenues which truly violates an individual’s privacy. In light of such allegations all the private chat applications have implemented end to end encryption wherein the chat data when stored in the company’s server is encrypted and no individual from the company has access to the any of the chats in a private chat room. In this quest we plan to build a secure chat application with all the modern facilities that the popular chat application currently boasts of with great focus on security features. Also, this would give us exposure and learning on handling and implementing security features in applications where the user base is generally large and security measures should be robust.

**Development**

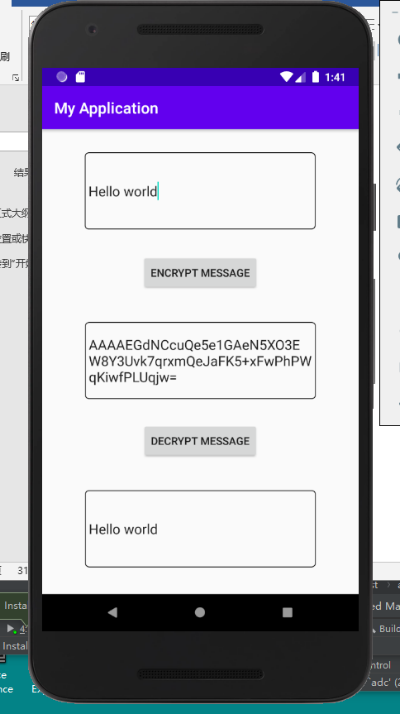
In the beginning of our development, we decided to split up the work, using python to implement the logic of Encryption/Decryption and message transmission, respectively.

Our First step was to spin up a basic encryption utility in python and then try to port it to android. This would have given us a chance to explore android at the beginning and at the same time implement the main algorithm of our application.

In the initial python implementation, we implemented a basic RSA public key, private key pair generation. In the python implementation we used the library called cryptography. After creating the keys, we generated a ciphertext for the given plain text with the public key and converted the raw ciphertext bytes to base64 encoding. To test the decryption part, we then went on to decrypt the plain text using the private key and made sure that the same plain text was decrypted back. Once this piece of code was implemented with python the next step was to port it to android.

For implementing in java this native java standard library has all the required cryptographic tools. Javas security package and javax.crypto packages were used to implement the same. In the initial implementation we generated an RSA key pair on our main activity code so that we could use the same public and private key for the entire course of the program execution. We implemented two driver functions namely the encrypt driver and decrypt\_driver. In the encrypt\_driver we passed the plain text to encrypt and the public key from the RSA keypair generated in the main activity class. This driver encrypted the plain text and stored it as the encrypted text. Now the decrypt driver was passed the private key from the RSA pair and used to the decrypt the text in the previous step. This was thus a basic implementation to get things started. The obvious next step was to show the code functionality in a basic User Interface.

Meanwhile, since all group members are new to Android development, we started with a basic UI to test Encryption/Decryption implementation on Android platform (Figure 1)



**Figure 1** Basic Android UI to test encryption and decryption feature.

After we port the python RSA encryption/decryption to Android, we found out that the encrypted message cannot be correctly decrypted. After some research, we realized that despite RSA asymmetric encryption seems ideal for this use case, it is not ideal for encrypting long messages. Therefore, we decided to use symmetric encryption algorithm (AES) to encrypt the message, then use asymmetric encryption on the AES key. To provide sufficient security to our cryptography, the app first generate a 256-bit AES key, 512-bit RSA keypair, and utilize SecureRandom to create the initialization vector.

For the encryption, to prevent vulnerability to differential attack, we used GCM mode to encrypt the user input message, then encrypt the AES key using RSA public key. In order to pack the initialization vector together with the cipher text, we created a byte array, first put in the length of the iv, then the iv, then finally the cipher text itself. In order to send the cipher text and the encrypted key, it is better to turn the byte array to a string. Therefore, the encrypted key and message were transformed encoded to Base 64 strings before sending out, since regular utf-8 string cannot correctly represent the encrypted objects.

For the decryption, the logic is opposite. First use Base 64 to decode the strings into byte arrays. Then, use the RSA private key to decrypt the AES key. After that, from the message, load the length as an integer from the first byte, then based upon the length to read the initialization vector, then the remaining array is the cipher text. Finally, use AES/GCM mode to decrypt the cipher text and get the original message.

The cryptography described above were written the CrptoUtil class. As shown in Figure 1, the encryption/decryption was perfectly implemented in Android.

After we make sure the messages can be encrypted on the fly, we look into the transmission of messages to find out how to make them “fly”.

In terms of messaging, we decided to begin with socket as our mean of communication. First, the server and the client would create a socket separately and try to establish connection with each other. Then, when sending out a message, the content will be encrypted locally and the cipher text will be sent through the socket with the private key packed together. When the other side receives them, it will use the key to decrypt the message and displays it. We are able to implement duplex communication in the end.

Despite the success in implementing basic socket transmission, it is not an ideal solution as backend service for a practical Android app. Therefore, after some research, we found that Firebase, which is a real time NoSQL database, is widely used in chat app tutorials as the backend to store and distribute the chat messages. Thus, we followed the tutorial written by Pradyuman Dixit [1] and built a prototype chatting app. We applied some adjustments to make it more user friendly, and incorporated the CrpUtil module to the app.

电脑萤幕的截图

描述已自动生成In our final solution, the implementation of message transmission works like this: one user encrypts the message, put the cipher text and the key in a hashmap, then push the hashmap to Firebase, which will then add a new child to the message node of the database. This event triggers the add child listener of the receiver. The listener takes a snapshot at the database, and take the new child node as a hashmap, then get the key and message from it, then decrypt the AES key and then decrypt the cipher text. In this process, however, the two users must first send the RSA public key to each other before they text anything. To address this issue, we make the app to generate RSA key pairs and send public key once two users enter the chat screen (OnCreate() method of the chat screen). The other user’s listener will receive and store the public key, then the two users can start chatting. Here is the screen shot showing the actual chatting between two device emulators. Notice that you can see the Firebase database structure and actual content in the background showing that the messages are encrypted.

**Figure 2** The screenshots showing two users chatting with each other.

**Summary**

In this group project, we obtained the essential skill sets on how to develop android apps, how to use Firebae, as well as the fundemantals and implementation of end to end encryption in real life. As a result, we successfully created an end to end chatting app on Android. Despite our final product is still a prototype, we believe by digging deeper into the security and features of Firebase service to handle the user login and logout, as well as fine tuning the UI design, it might become a solid and secure chatting app that can be used in real life.

**Contributions**

Hao Fu and Gandhar Satish Joshi created the implementation of message transmission and Hao also set up the Firebase Database. Ashutosh Agrawala created the python implementation of cryptography and ported to Android, as well as implementing a Firebase database for the initial real-time storage of the encrypted messages. Jin Liu created the basic UI, and provided further support on the UI design as well as the cryptography implementation on Android. All team members are involved in the development of the final product.

**References**

1. https://hackernoon.com/how-to-make-a-personal-chat-application-like-whatsapp-fda6dd4bcb5b