Global Secure Messaging App (GSEM)

Louis Thomas Kavouras

E-mail: louis.kavouras1@marist.edu

Abstract

Today's modern day, just about everyone has a smart device of some kind. Along with these smart devices, users have applications from utilities to social media. However, with this great technology, many of us are sacrificing our data's security for usability. In additon, with all the great social media platforms, messaging services, other forms of communication, we are prone to attacks. Facebook had in the past was in the cross hairs for having user passwords left in plaintext form, which left their clients very vulnerable to fraud and other attacks. Global Secure Messaging App (GSEM) for short, is a solution to all of those worries by incorporating AES and RSA algorithms, communicaton from one client to another is all secured on the client and server side of the application.

Keywords: AES, Advanced Encryption Standard, Rivest Shamir Adleman, Android, Firebase, Hashmap, Socket, Java

1. Introduction

Global Secure Messaging App also refered to as GSEM, can be thought of as a form of communication that keeps privacy in mind. My inspiration and competition was the application known as *WhatsApp*. Similar to *WhatsApp*, GSEM provides **end-to-end** encryption for clients across the world. GSEM is utializes a custom **AES** implementation, which will be discssued further in detail.

Security in today's day is very ugly, not the cleanest and most intuative which leaves it prone to human error. GSEM is currently designed for the Android operating system, and is designed in a way that is intutative and modern to the end user. Within software development there is something known as the three-click rule, which means any user should be able to get to a page or complete an action withing three clicks. I took this methodology and implemented into the graphical user interface and functionality of the application GSEM. There are three main screens, the Chat, Settings, and Profile. To navigate between each of them, there is an icon for each on the navigation bar.

2. Background

During the early phases of the development for this project, I had many ideas and still do about how to tackle this task. At first I thought to incorporate a combination of RSA and AES to make an asymmetic based crytosystem implementation. However, I chose to go the private key (symmetric) route. I may incorporate an RSA / AES implementation for another feature for this application down the road, however, due to the targeted audience, AES alone was sufficient.

The audience / users for this application are most likely going to be close to one another prior to the use of this app. Therefore, sharing the top-secret phrase via casual conversation, or another method should be fairly easy.

3. Methodology

Now you may have several questions such as the following; How does this application know how do its job? Are you one-hundred percent certain that my conversations are secure?

The answer is yes, all of your conversations are securely encrypted and stored on a server. Even if someone was to gain physical or remote access to the database on the server, they would not be able to decrypt the message that they intercepted. On startup of the application, you will be brought to the chat page automatically. Since this will most likely be your first time using the application, you will need to define or create a private key to gain access or start a new conversation. Conversations can only be viewed once the private key is stored on the users device.

Currently, GSEM utilizes an one twenty eight bit encryption key implementation. The main characters used in this document are **Alice**, **Bob** and **Eve**, where **Eve** is the malicious hacker. In this demo using the phrase mentioned before **Alice** and **Bob** will like to talk to eachother without **Eve** evesdropping on their conversation. GSEM automatically does the encryption and decryption on every message sent and received to eliminate another tedious step to be done on their side. With regards to encrypting the message, GSEM generates the **round keys** K_i , for i = 0, 1, … , 10, from the original 128-bit key K_e. The reason why I mentioned that the key must be sixteen characters long is due to the key being a four by four square matrix.

$$K_e = \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}$$

To decrypt the message, GSEM simply inverts the process which can be represented as **m=** $\mathbf{D_K(c)}$**.**

It is important to note that the messages that are rendered on the users device are **snapshots of the data**. The state of the data stored in the **data store** are **never modified**. In other words, to view the messages, the messages are **not** decrypted and left in plaintext form on the database. The snapshots of the data on the local device allows the end-user to view the contents of the message securely.

Next topic ***Storage of the key***. The ***private*** key is stored on the users device. This was done for the most obvious reason, you would not keep the key to your safe that holds all of your possessions ontop of the box. Same analogy applies to this symmetric cryptosystem. The key to access the messages is stored locally and used only to encrypt and decrypt the messages. All of the messages are stored in their **encrypted form** on the server in the database, which this demo utilizes **Firebase** for the back-end functionality. To generate the key you will need to visit the settings tab.

Once the end-user navigates to the settings tab you can enter the phrase, which will automatically generate a key. This phrase must be **sixteen characters** in length. To assist with this process, the input field has a limit in place, to stop you once you reach the sixteen character limit. For the demonstration purposes, our phrase is ***candydailyisgood***. Once the key is entered, the application will generate the key on your behalf using the phrase entered which will then provide you access to the chats (if any exist).

If this is the first conversation or a new one, the chat page will be empty, until someone with in your inner circle sends a message.

Another feature of GSEM is ***total annomity*** which means with every message sent it will display your name next to the message. However, you do not need to use your real name, a screenname will suffice. This can all be done under the profile page, and once again this is all done within three-clicks, or taps in this case.

So far I have discussed the key generation as well as the overview of the AES implementation. However in order to create the ciphertext, you must have plaintext to encrypt. This is where the message field on the chat page comes in handy. Below is an overview of how AES works visually (see Figure 1).

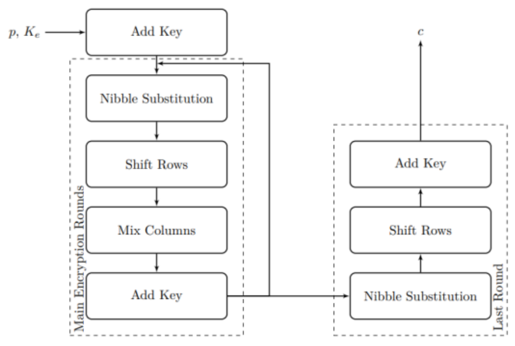
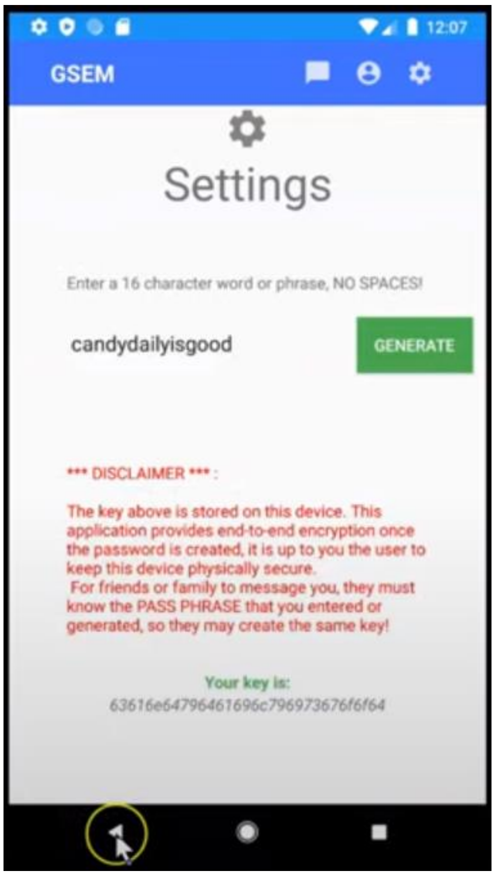


Figure 1: AES sequence diagram.

4. Experiments / Demo

Here is a screenshot of my demonstration where I am entering the demo key ***candydailyisgood***.



Enter a 16 character word or phrase, NO SPACES!

candydailyisgood

*** DISCLAIMER *** :

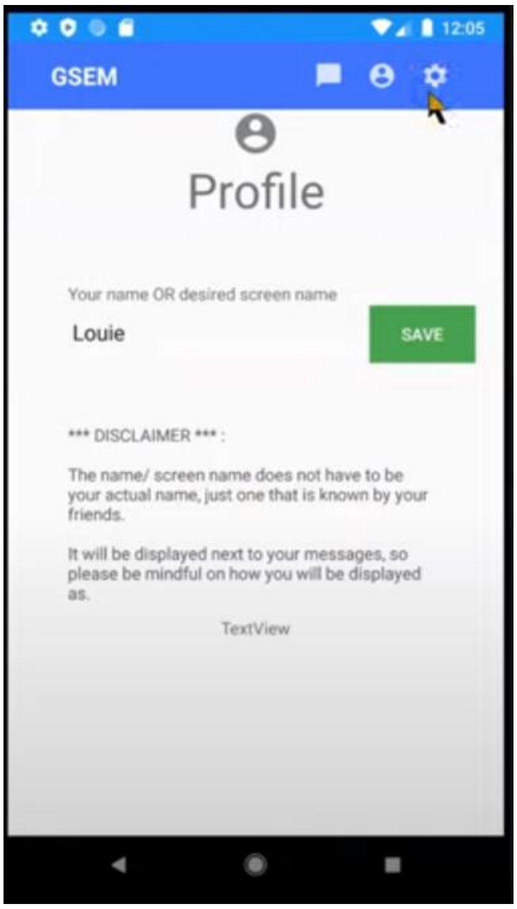
The key above is stored on this device. This application provides end-to-end encryption once the password is created, it is up to you the user to keep this device physically secure.
For friends or family to message you, they must know the PASS PHRASE that you entered or generated, so they may create the same key!

Your key is:
63616e64796461696c796973676f6f64

Notice if the phrase is within the sixteen-character limit, it will generate the key and display it in hexadecimal format. Currently GSEM converts that hexadecimal key into it's decimal notation. (May be a bit redundant or excessive), however it was my take on keeping the data secure.

Next, I navigated to the settings tab by tapping on the gear.

Here you can enter your name or screen name of choice.



Your name OR desired screen name

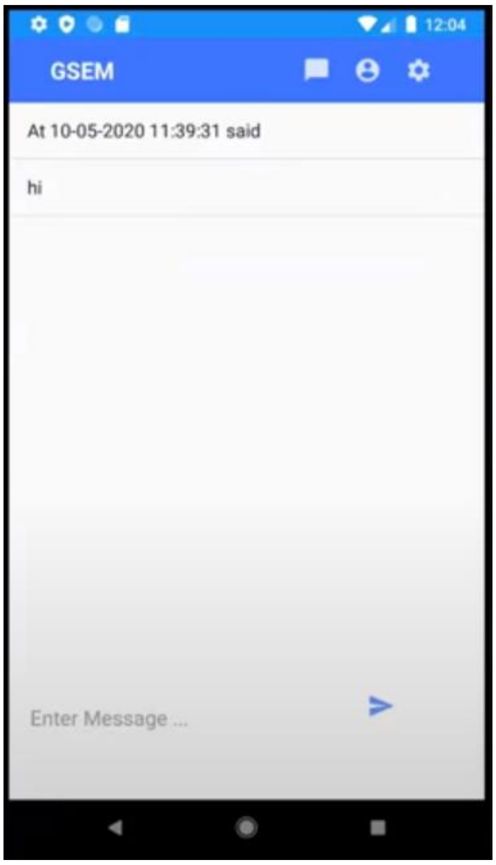
Louie

*** DISCLAIMER *** :

The name/ screen name does not have to be your actual name, just one that is known by your friends.

It will be displayed next to your messages, so please be mindful on how you will be displayed as.

Finally with all of that done, you can now tap on the chats tab to view messges. Notice this is not my first time using this conversation thread, so prior to me adding my name, I sent a demo message "hi" to the database and since I did not have a name it was left blank. I allowed this to occur to demonstrate that you may remain annonomous if you want. However attaching a screenname or your real name will allow the other receipients to know who sent the message.

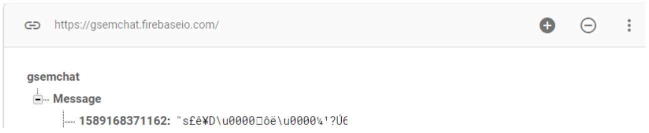


At 10-05-2020 11:39:31 said

hi

Once the end-user enters content aka the message, and taps the blue arrow, GSEM will encrypt the message using the private key and store it on the database as well as render it visible to the GSEM client app.

Finally, this is how the data is stored within the database,



```
gsemchat
 Message
   1589168371162: "s£ê¥D\u0000□ôë\u0000¼¹?Ú6
```

It is important to note in the screenshot that the only thing visible from the server-side is the timestamp of the message and the message is unable to be read by anyone with access to the server or the server itself.

5. Discussion / Analysis / Conclusion

Future Developments / Ideas:

- Add a call method, where you can video call and voice call the group in a secure manner
- Have the ability to send images and other multimedia.

6. References

Babic, Filip. "Using Firebase to Create a 'Simple' Chat Application in Android." Medium. COBE, August 17, 2017. https://medium.cobeisfresh.com/using-firebase-to-create-a-simple-chat-application-in-android-4b32fdbf565e.

"Cryptography : Android Developers." Android Developers. Accessed April 13, 2020. https://developer.android.com/guide/topics/security/cryptography.

DigitalOcean. "SSH Essentials: Working with SSH Servers, Clients, and Keys." DigitalOcean. DigitalOcean, January 12, 2017. https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys.

"Generate Public and Private Keys." Generate Public and Private Keys (The Java™ Tutorials > Security Features in Java SE > Generating and Verifying Signatures). Accessed April 13, 2020. https://docs.oracle.com/javase/tutorial/security/apisign/step2.html.

"GNU Handbook - Distributing Keys." Distributing keys. Accessed April 13, 2020. https://www.gnupg.org/gph/en/manual/x457.html.

Instructables. "Creating a Chat Server Using Java." Instructables. Instructables, September 30, 2017. https://www.instructables.com/id/Creating-a-Chat-Server-Using-Java/.

“Java Cryptography - Encrypting Data.” Tutorialspoint. Accessed April 13, 2020. https://www.tutorialspoint.com/java_cryptography/java_cryptography_encrypting_data.htm.

“Java Digital Image Processing Tutorial.” Tutorialspoint. Accessed April 13, 2020. https://www.tutorialspoint.com/java_dip/.

Kaustav, Kaustav, and Chanda. “Creating an Asynchronous Multithreaded Chat Application in Java.” GeeksforGeeks, April 1, 2019. https://www.geeksforgeeks.org/creating-an-asynchronous-multithreaded-chat-application-in-java/?ref=rp.

“Multi-Threaded Chat Application in Java: Set 2 (Client Side Programming).” GeeksforGeeks, June 17, 2017. https://www.geeksforgeeks.org/multi-threaded-chat-application-set-2/?ref=rp.

Payán, Abel Suviri. “Create RSA Key on Android for Sign and Verify.” Medium. Medium, March 26, 2019. https://medium.com/@abel.suviri.payan/create-rsa-key-on-android-for-sign-and-verify-9debbb566541.

“Public Key Encryption.” Tutorialspoint. Accessed April 13, 2020. https://www.tutorialspoint.com/cryptography/public_key_encryption.htm.

RichardCypherRichardCypher 12366 bronze badges, and Marvin PintoMarvin Pinto 16333 silver badges1212 bronze badges. “Implementing RSA into an Android Messaging App.” Software Engineering Stack Exchange, December 1, 1961. https://softwareengineering.stackexchange.com/questions/137664/implementing-rsa-into-an-android-messaging-app.

Somnium, SomniumSomnium 2, G. H. FaustG. H. Faust 69655 silver badges44 bronze badges, edc65edc65 31.1k33 gold badges2828 silver badges8787 bronze badges, Todd LehmanTodd Lehman 1, trichoplaxtrichoplax 10.5k66 gold badges4444 silver badges7676 bronze badges, DenDenDoDenDenDo 2, et al. “Rearrange Pixels in Image so It Can't Be Recognized and Then Get It Back.” Code Golf Stack Exchange, May 1, 1964. https://codegolf.stackexchange.com/questions/35005/rearrange-pixels-in-image-so-it-cant-be-recognized-and-then-get-it-back.

Stanoyevitch, Alexander. *Introduction to Cryptography with Mathematical Foundations and Computer Implementations*. Boca Raton: Chapman & Hall/CRC, 2013.

“Store RSA Public and Private Key to Mysql Database.” Store RSA public and private key to mysql database (Security forum at Coderanch). Accessed April 13, 2020. https://coderanch.com/t/651828/engineering/Store-RSA-public-private-key.

“Understanding RSA Algorithm.” Tutorialspoint. Accessed April 13, 2020. https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_understanding_rsa_algorithm.h