E2EE Messaging App

Evin Bour-Gilson, Nicholas Baird, Pratham Vadhulas, Jay Shukla, Avinash Pandey

Indiana University Purdue University Indianapolis

CSCI 43200

Professor Scott Orr

April 19, 2024

Table Of Contents

Project Source Code:

https://github.com/NickBaird/CSCI43200-Project/

Video Demonstration of Project:

https://drive.google.com/file/d/1JccJMUCiDzNrKSQJL-lv7r1LEyx-qqT3/view?usp=sharing

INTRODUCTION

The purpose of this document is to explain the development, implementation, and testing of an end-to-end encrypted messaging application. By implementing a variety of security protocols and industry leading standards we were able to create a robust and secure application. Through this application, we hope users will have confidence that their messages are protected from any outside threats.

To manage authentication and conversation storage we leverage Google Firebase real-time database [1]. The back-end is built using JavaScript and the cryptographic library libsodium which provides an advanced elliptic-curve Diffie-Hellman key exchange mechanism [2, 3].The front-end is developed with Vue.js and stylized by Tailwind CSS for a better user experience [4, 5]. The functionalities allow users to communicate in real time securely with additional security features that prevent unauthorized message sending from unrecognized devices.

We begin the report by introducing the history of messaging applications and the importance of end-to-end encryption. Then we discuss the technical implementation of the app, talking about the various technologies used and why they were used. After this we include the various ways we tested our application for security and privacy. Finally, we mention the challenges we faced during the development process and how we overcame them.

This project showcases our commitment to creating a user friendly and secure platform with various technologies we'll talk about later in the report. We have also included a video demonstration of the project linked above which showcases the user experience.

PROBLEM/BACKGROUND

In an increasingly digital world, concerns such as privacy and security have taken a backseat. Companies stand to benefit from information shared in personal conversations by selling data to advertisers and other third parties. Furthermore, bad actors with access to messaging information through interception could glean valuable information such as financial and health related records. Even government agencies use online communication to conduct mass surveillance, infringing on the personal rights of their citizens [6].

Given these threats, safeguarding communications from unauthorized access has never been more crucial. End-to-end encryption has emerged as a key solution, ensuring only intended recipients can access messages. The need for this technology has given rise to multiple messengers such as Signal, WhatsApp, and Telegram. Through our application, we intend to employ the same robust, industry leading security technologies to give users confidence that their conversations are not compromised or misused at any point during transmission [7].

Although end-to-end encryption might be the gold standard for messaging applications today, these standards have not always been in place. Before Signal launched in 2014, systems such as short message service (SMS), instant messaging, and mobile messaging apps such WhatsApp offered cheap and reliable communication channels, but they lacked robust security protocols. Even since the widespread adoption of end-to-end encryption, technology has advanced to protect users. Older algorithms such as DES (data encryption standard) have been replaced by AES (Advanced Encryption Standard) and elliptic curve cryptography (ECC) [8]. Our application aims to use the most up to date encryption algorithms alongside Diffie-Hellman key exchange to give users the utmost confidence in their privacy.

CHALLENGES

Through this project we wanted to create an application which combined rigorous security protocols with a seamless user experience. While there are many modern tools available to help with this endeavor, we still faced numerous roadblocks in development. Some of these challenges along with our solutions are highlighted below.

End to Encryption Related:

- Managing key lifecycle and ensuring keys are securely exchanged without exposing them to potential interception was complex.
- Ensuring that message encryption and decryption processes are efficient enough to handle real-time messaging without delays.

Database and Communication Related:

- Handling real-time data synchronization across devices proved challenging, especially in scenarios with poor network conditions.
- Ensuring data integrity and consistency when simultaneous database writes occur was a significant hurdle.

JavaScript:

- Debugging asynchronous JavaScript code, especially with multiple nested callbacks and external library integrations, was challenging.

Vue.js:

- Using the js.js file as a source of all main functions and variables was difficult to work with. We had to modify the common Vue.js setup to accommodate this unique and rare

setup of using functions that were within a vanilla JavaScript file rather than within a Vue file.

- When importing variables within JavaScript, they are imported as constants, so they do not change after importing even if they are changed in the file from which they were imported from. This caused issues with asynchronous functions and global variables when importing them to a Vue file. The variables would import as null due to the fact that the asynchronous functions that the variables depended on would import the data immediately instead of waiting the time needed to pull the data from the database. We solved/worked around this issue by utilizing promises and creating separate functions that first pull the data, and then display the data.

Styling:

- Regular CSS offers a lot of flexibility, but in order to maintain a steady design throughout the app, it is challenging. Huge companies take years to develop a good consistent design. We found a solution to this problem, and that is TailwindCSS, a utility-first CSS framework.

IMPLEMENTATION DETAILS

When implementing our end-to-end encrypted messaging web application, many different libraries and services were used. To begin, users would use Firebase authentication either to sign up an account or login to an existing account. Here, when the user signs up, two pairs of public and private keys are generated using libsodium [3]. One pair is used for encryption and decryption (X25519) and the other pair is used for signing and verifying messages (Ed25519). The user's public keys will be sent to the database, while their private keys will be saved locally to their browser using IndexedDB [9].

With the keys established and authentication granted, then the user is then able to use the Firebase Real-time Database. Of course, there are only certain parts where such a user can read and/or write data to ensure security. Then, the user can invite another user to have a conversation, which is 1-to-1 messaging. If the user accepts the invite, both users will be able to use each other's public keys to exchange messages or files to one another. Our web application also supports group chats; therefore, a user is able to create a group chat, with some encrypted name, and then invite people to it. Like conversations, a user can accept the invitation, and will receive the name of the group chat and any messages put in the group chat since they have joined the group, all encrypted. Unlike a conversation, group chats require a message to be sent to multiple users; therefore, the way we decided to do this follows how Threema, a popular end-to-end encryption messaging app, does it. Our implementation was to send the message to each user in the group, using their respective public keys for encryption [5]. This would not be ideal for files, as files are way larger in size than text and would eat up too much bandwidth. This is why we upload the file once, encrypt it with some generated key just for that file, then send the decryption key to each user in the group chat, using the traditional encryption method.

When a user leaves a group, or is kicked, they will lose access to the messages that have already been sent and any further messages sent in the group chat. Since they do not have access anymore, they would not be able to see the messages they have sent or received when in the group; however, the messages that were sent to or by the user, still reside in the database.

In our implementation, privacy is of the utmost importance; therefore, the only information that is visible or can be derived from when looking at the database, as a database administrator, is the user's display name, a user's public keys, who the user is messaging either as a conversation or group chat, and the timestamp of a message being sent. Every other information, such as the message or file themselves, are fully encrypted and signed.

TESTING/ RESULTS

Our goal in this project was two-fold: first to create a robust end-to-end encrypted messaging system, and second to create a seamless user experience. To ensure we met these goals, we employed both unit testing and functional testing methods. Through this technique, we were able to be certain that both the average user and extreme cases were handled appropriately by the application. Below we have highlighted how we approached testing various functions of the messenger alongside the results. Further implementation is shown in the attached video.

Authentication:

- Users are able to create accounts with a valid email address and password
- Make sure all edge cases, like an already existing account with the same email address, are handled
- We made sure that even if someone is able to login with credentials, that the private keys, which are stored locally on a device, are necessary in order to receive interpretable information from the database

Real-time Database:

- The real-time database has built-in database rules that require certain conditions to allow reads and writes to a specific part of the database
- They also support a rules playground tool which allows various testing capabilities for the database rules [10].
- This tool was used during the development process for implementation/debugging, and it was also used in this testing process to ensure secure database reads and writes.

Encryption/Decryption:

- Tested that encryption and decryption works only with the correct shared keys
- Tested that signing messages and verifying messages work and prevent the possibility of a man-in-the-middle attack

Local Test:

- We were able to setup two different accounts in two different browsers on the same machine
- As connecting to Firebase is required, no communication is done locally through the machine and only through Firebase
- Because of this, this test is practically the same as a non-local test, where two separate devices, which can also be on different networks, are used

Device-to-Device:

- Have two different people on different machines setup an account
- Both people exchange user IDs and establish communications between each other, either through a conversation (1-to-1) or a group.
- Are able to send and receive messages and perform the other functions mentioned previously, with ease

SUMMARY

Thus, this documentation talks about the implementation of an end-to-end encrypted messaging application, which is built to conduct a safe interaction between the users. This program uses technologies like Firebase Realtime Database, Firebase Authentication, cryptographic protocols via Libsodium, and security features like Diffie-Hellman key exchange.

During the project, we came across many problems, especially in the development process. Some problems were maintaining important life cycles, guaranteeing real-time data integrity, and streamlining asynchronous processes in our framework. All of this was built with JavaScript and Vue.js. We also incorporated TailwindCSS for the front end part.

In the testing stages we were able to confirm the dependability and security of the application. The tests were validated when we were only able to decrypt the encrypted messages by the intended recipients and protected the user's privacy from interceptions.

With the potential to significantly enhance the status of secure communication technology, this project demonstrates our dedication towards both security and innovation. Looking ahead, our goal is to further improve our application with the highest levels of security and privacy.

If we were to continue this project as a group, there are a few features we could add to enhance the user experience. One really useful feature would be to make it easy to transfer an account (private key) from one device to another. For example, a user could use their phone to scan a QR code of an account they already have set up on a desktop. Other useful features would be to incorporate two-factor authentication and notifications. Luckily, Firebase does support both of these features in their service, making these features easily compatible with our application. In our JavaScript file, we created, but were not able to implement and test, functions that would

also be beneficial to implement if we had time to continue this project. Some of the functions that we have created, but did not have time to implement, are: blocking users and group chats, deleting all messages in a chat with one button, and promoting someone else in a group chat to admin so that they would be able to invite other users as well. Lastly, another small, but useful, feature would be the ability to react with an emoji to a sent message.

REFERENCES

[1] "Firebase documentation," Google, https://firebase.google.com/docs (accessed Apr. 19, 2024).

[2] "Quickly validate Firebase Security rules," Google, https://firebase.google.com/docs/rules/simulator (accessed Apr. 19, 2024).

[3] "Using IndexedDB," MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB (accessed Apr. 19, 2024).

[4] K. Hassel, "What is end-to-end encryption & why does it matter?," PIA VPN Blog, https://www.privateinternetaccess.com/blog/what-is-end-to-end-encryption/ (accessed Apr. 19, 2024).

[5] "Cryptography Whitepaper," *Threema*. Apr. 4, 2024

[6] D. K. Gillmor and J. S. Granick, "The vital role of end-to-end encryption: ACLU," American Civil Liberties Union, https://www.aclu.org/news/privacy-technology/the-vital-role-of-end-to-end-encryption (accessed Apr. 19, 2024).

[7] "Resources," tailwindcss, https://tailwindcss.com/resources (accessed Apr. 19, 2024).

[8] "Introduction," Vue.js, https://vuejs.org/guide/introduction.html (accessed Apr. 19, 2024).

[9] "Introduction," libsodium, https://doc.libsodium.org/ (accessed Apr. 19, 2024).

[10] J. Lake, "Demystifying Diffie-Hellman key exchange and explaining how it works," Comparitech, https://www.comparitech.com/blog/information-security/diffie-hellman-key-exchange/ (accessed Apr. 19, 2024).