E2-Chat: A Web-Based End-to-End Encrypted Messaging Service

SAITEJA BEVARA, University of Virginia School of Engineering and Applied Science, USA
ASHWIN PATHI, University of Virginia School of Engineering and Applied Science, USA
PHILLIP PHAN, University of Virginia School of Engineering and Applied Science, USA
RITHIK YELISETTY, University of Virginia School of Engineering and Applied Science, USA

End-to-end encrypted (E2EE) messaging services currently require mobile devices, since they are portable and do not require the cumbersome transfer of keys around devices. Even current desktop E2EE messaging services such as WhatsApp, interface through phones for their chat services. This project aims to implement a new web-based E2EE messaging service based on previous research of encryption protocols for public/private key exchange using RSA, or Diffie-Hellman[11]. In areas with low smartphone usage such as emerging economies like India, Nigeria, and Indonesia where more than half the population do not own smartphones, but majority have access to the internet, this service will enable users to have encrypted channels of communication[10]. This can also be useful for businesses, who might use an E2EE chat internally or with customers. The project supports one-to-one chats, group chats, and sending various media such as images, all without the use of an auxiliary mobile device. This project was developed using technologies such as React.js, GraphQL, Node.js, and web browser key generation libraries. Technical challenges for this project included the pagination required for loading chats efficiently and the encryption protocols for group messages. Results showed the data in the database was fully encrypted and contained no identifiable plain text messages. Since private keys are not stored on the server, these encrypted messages cannot be decoded by third parties. These two conditions were verified by examining the contents of the database after sending messages between two users.

ACM Reference Format:
Saiteja Bevara, Ashwin Pathi, Phillip Phan, and Rithik Yelisetty. 2020. E2-Chat: A Web-Based End-to-End Encrypted Messaging Service . , (2020), 6 pages.

1 INTRODUCTION

Chat services allow parties to communicate in real-time via the Internet. For popular chat services like Facebook Messenger and Slack, messages are sent through an intermediary service and are stored in a third-party database [6]. These messages are only encrypted in transit, which means the messaging service provider has access to one's messages. Other third parties can also view an individual's messages if, for example, a malicious actor illegally obtains access to the database containing one's messages or a law enforcement entity has a subpoena to obtain an individual's message records[9]. In the United States, roughly eight in ten adults believe they have little or no control over the data that companies and the government collect, and a large majority believe that the risks of collection outweigh the benefits [1]. These concerns illustrate the importance of a messaging service where only the sender and receiver of a message can see the messages.

End-to-end encrypted (E2EE) services allow for the encryption of text, such that content is only readable by the communicating parties and not any third-parties. End-to-end encrypted systems have become some of the most popular communication methods in today's connected world, the most common services being WhatsApp (owned by Facebook) and iMessage (owned by Apple). WhatsApp has over two billion monthly active users and iMessage reaches over one billion users monthly [8] [3]. However, most messaging platforms that utilize E2EE, such as WhatsApp, interface explicitly through smartphones to enable encryption [2]. Those without access to smartphones or specific hardware are relegated to using unencrypted mediums of communication. This is significant particularly in emerging economies such as India and Brazil, where more than 60% of adults use the internet but only 45% have access to smartphones [10]. This lack of access to encrypted communication for large populations motivates the need for a web-based end-to-end encrypted messaging service. By being web-based, the chat service is available to everyone with an internet-enabled device, allowing users to overcome barriers in pursuit of data privacy.

The project resulted in E2-Chat, a fully web-based end-to-end encrypted platform built to allow every internet user to have secure means of communication. E2-Chat gives users the ability to interact in one-on-one chats and group chats by sending text messages, images and files. E2-Chat is built using a variety of open-sourced technologies including React.js, GraphQL and Node.js. End-to-end encryption was verified through checking the database and ensuring that no content and private keys were stored in plain-text.

2 BACKGROUND

In this section, we will provide a brief overview of the key concepts and components that were used in building E2-Chat. These include the RSA and AES encryption schemes. They also include open source technologies such as React, Node, GraphQL, and SQL.

2.1 RSA

Rivest-Shamir-Adelman (RSA) encryption is a public-key encryption scheme. Public key encryption schemes are processes used for encryption and decryption of text. Among existing public-key encryption schemes, the RSA system is both widely used and sufficiently secure. Categorized as an asymmetric scheme, it relies on a pair of public and private keys to encrypt and decrypt messages, and guarantees that key generation is efficient for computation, it is impossible to derive the private key from the public key, and it is infeasible to obtain the decrypted text from only the encrypted text and public key. This encryption system thus is an ideal fit for

Authors' addresses: Saiteja Bevara, University of Virginia School of Engineering and Applied Science, , Charlottesville, USA, sb2xf@virginia.edu; Ashwin Pathi, University of Virginia School of Engineering and Applied Science, , Charlottesville, USA, asp7pr@virginia.edu; Phillip Phan, University of Virginia School of Engineering and Applied Science, , Charlottesville, USA, pp5fb@virginia.edu; Rithik Yelisetty, University of Virginia School of Engineering and Applied Science, , Charlottesville, USA, ry9bf@virginia.edu.

this project as it allows users to generate key pairs efficiently upon registration, which can be used securely to encrypt and decrypt messages [7].

2.2 Advanced Encryption Standard and Symmetric Key Encryption

Advanced Encryption Standard (AES) is a symmetric key encryption scheme. In this form of encryption, a single secret key both encrypts and decrypts information. This key must be distributed to communicating parties, and the key will then be used to convert plaintext to ciphertext and vice versa for senders and recipients of content. AES is also categorized as a block cipher, which operates on fixed-length groups of bytes called blocks, and uses the fixed key to encrypt these individual blocks.

2.3 Browser Local Storage

Local storage is space allocated in the browser for each domain to store data that is relevant to an application. In most modern browsers, local storage has a capacity of 5 megabytes and is stored in key-value pairs. This format is comparable to the format of a JSON file.

2.4 React

React.js is an open-source frontend framework written in JavaScript. React.js uses a component-based structure that allows developers to create a component once and use it in multiple places across their codebase. The framework dynamically converts code from JSX to native HTML and vanilla JavaScript to allow applications to be run natively in the browser.

2.5 Node

Node.js is a scalable server side scripting language built entirely on native JavaScript. This server technology is primarily used in an event driven model meaning that the server is only actively working when a request is being made. Node.js also follows the asynchronous model from vanilla JavaScript which helps code run in a much more efficient manner.

2.6 GraphQL

GraphQL is a querying language developed by Facebook that interacts with both server side technologies and databases. GraphQL acts as an alternative to the popular REST framework (which completes requests using a stateless model), and has the advantages of flexibility with regards to data types and formats, and reduced overhead by removing unwanted data in query results [4]. This technology also comes integrated with pagination which ensures that only data that needs to be used is sent from the server to the end user. A GraphQL query is used to receive data from the server, while a GraphQL mutation is used to send data from the client to the server to update existing data. A GraphQL subscription is used to stream a set of data from the server to the client.

2.7 Structured Query Language

Structured Query Language (SQL), is the primary method of communication between a backend server and a database system. SQL's main goal is to help set up the structure of the tables in the database, add and remove data from the database and retrieve data from the database based on filter restrictions provided. There are many flavors of SQL, the most common ones being MySQL, PostgreSQL, SQLite and Microsoft SQL Server. The first three mentioned are all similar, however, have differences in how memory is structured to actually hold the data.

3 RELATED WORK

Many messaging services have amassed over a billion monthly active users, including Facebook Messenger and WeChat. However, many of these messaging services are not end-to-end encrypted, meaning that companies and governments have the ability to view private message content. There are many end-to-end encrypted messaging services available for consumers today including WhatsApp, Signal, Telegram and iMessage. The two primary forms of communication that will be focused on are WhatsApp (owned by Facebook) and iMessage (owned by Apple). Whatsapp, at a high level, uses an encryption protocol that is based on the Signal Protocol, which generates key pairs for users upon registration and uses them to create master secrets that define encrypted messaging sessions for each chat. According to their whitepaper, WhatsApp uses the "client-side fan-out" method to distribute keys to every user that is within a group and then uses the "server-side fan-out" idea to send every message from there. This improves the overall efficiency of the service as each message is only encrypted one time [11]. WhatsApp, aside from the base feature of having one-to-one chats, is capable of serving group chats and media transfer. The primary disadvantage of WhatsApp is that every account must be linked to a smartphone. To use the service via a desktop, users must use the web client and ensure that both the linked mobile phone and the laptop have internet access. Messages from the laptop are then transmitted via the phone, meaning that a network eavesdropper may be able to decrypt messages during this transfer via a Wi-Fi network. Another disadvantage of WhatsApp is that each user can only have one account and cannot use the same WhatsApp account on multiple devices.

The other major E2EE platform, iMessage, differs significantly from the Facebook owned app. iMessage uses the underlying Apple hardware to generate keys and connects to their iCloud service to encrypt and decrypt messages. This helps ensure that all of the devices that are logged into the same account can see all of the messages. The primary issue with iMessage's approach is that every message is encrypted several times, based on the number of devices linked to the sender's and recipient's iMessage account [5]. This Apple hardware requirement restricts people from using E2EE services and further decreases their reach by only allowing their users to communicate with other users that have these devices. From this, it is evident that there is currently no application that serves as an encrypted messaging platform that does not require smartphones.

4 SYSTEM DESIGN

E2-Chat uses a form of hybrid encryption to facilitate E2EE messaging. AES encryption is used to encrypt message content within group chats and RSA encryption is used for group key fan out. React

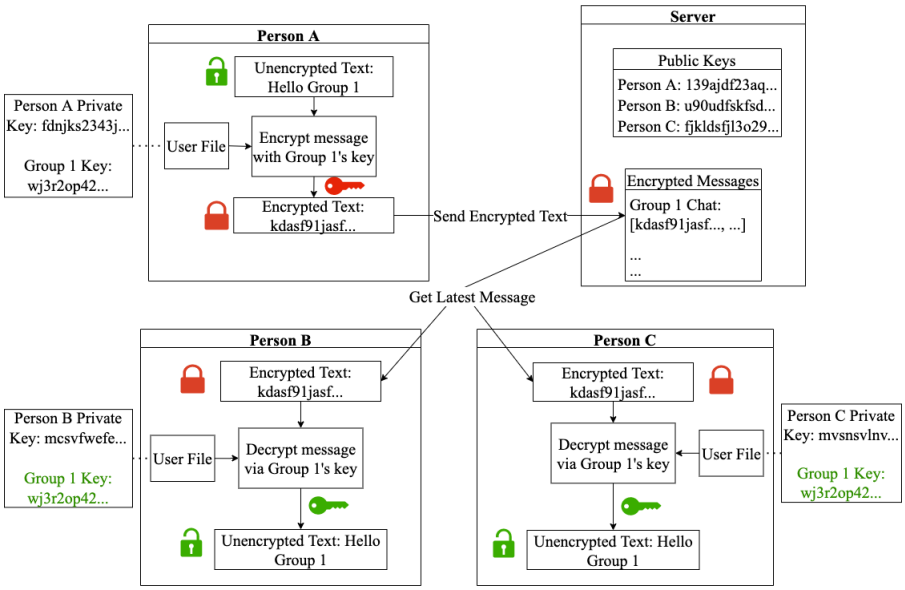

Fig. 1. Data Flow of Encrypted Messages in a Group Chat

serves as the frontend framework, Node.js for the backend server, Apollo GraphQL as the variant of GraphQL for sending data from the backend to the frontend, and PostgreSQL for the database.

4.1 System Flow

When a user wants to creates new messaging group, an AES group key is generated by the group creator. This AES key is encrypted using the other group members' public keys before being fanned out to the other group chat members. The encrypted group AES key is then decrypted using each user's private key, which finishes the process for creating a group chat. When a user wants to send a message to a group chat, a similar process occurs. However, the AES group key is instead used to encrypt and decrypt messages that are sent. Figure 1 illustrates the process of obtaining the encrypted message content from the server and decrypting it using the AES group key available to each user. Person A sends a message "Hello Group 1" in its plaintext form. This message is encrypted using the Group 1 Key (the AES key) and is transmitted and stored on the server as encrypted text. Person B and Person C, who are also members in Group 1, then will obtain this encrypted text from the server when they load this group chat. The message content will be decrypted using the same Group 1 AES key, and displayed in its plaintext form. At all points in this process, only those with access to the AES key (Group 1 Key) can view messages as plaintext.

4.2 Implementation

The following sections outlines the implementation details for E2-Chat. An explanation on how GraphQL is used in E2-Chat is described in *Data Manipulations and Queries*.

4.2.1 Data Manipulation and Queries. In order to send data from the server to the client, GraphQL queries were generated. An examples of a GraphQL query that was created is:

```
groupsByUser(username: String!): [GroupOutWithPrivateKey]
```

This query expects the username as a string and will return an object of type GroupOutWithPrivateKey. This type consists of the following data (formatted in GraphQL):

```
type GroupOutWithPrivateKey {
    id: Int!
    users: [UserOut!]
    name: String!
    privateKey: String!
}
```

This query will return all of the groups that a user belongs to and will only consist of following data: the group id, the users (formatted as a JSON object with the username), the name of the group and the AES key to decrypt messages.

In order to modify or change data, GraphQL mutations are used. One such example is the createMessage mutation (shown below), which accepts the encrypted content as a string, the group id, the sender and the content type (text or file).

```
createMessage(content: String! group: Int!, sender:
     String!, cType: String!): Message
```

Once the resolver (the associated server-side code) executes, the mutation will return a message object that consists of the encrypted content, the group id, the sender's username, the content type and the time the message was created. To receive data streams, GraphQL subscriptions were implemented. Subscriptions allow the data to be received at any given time when someone publishes to the channel

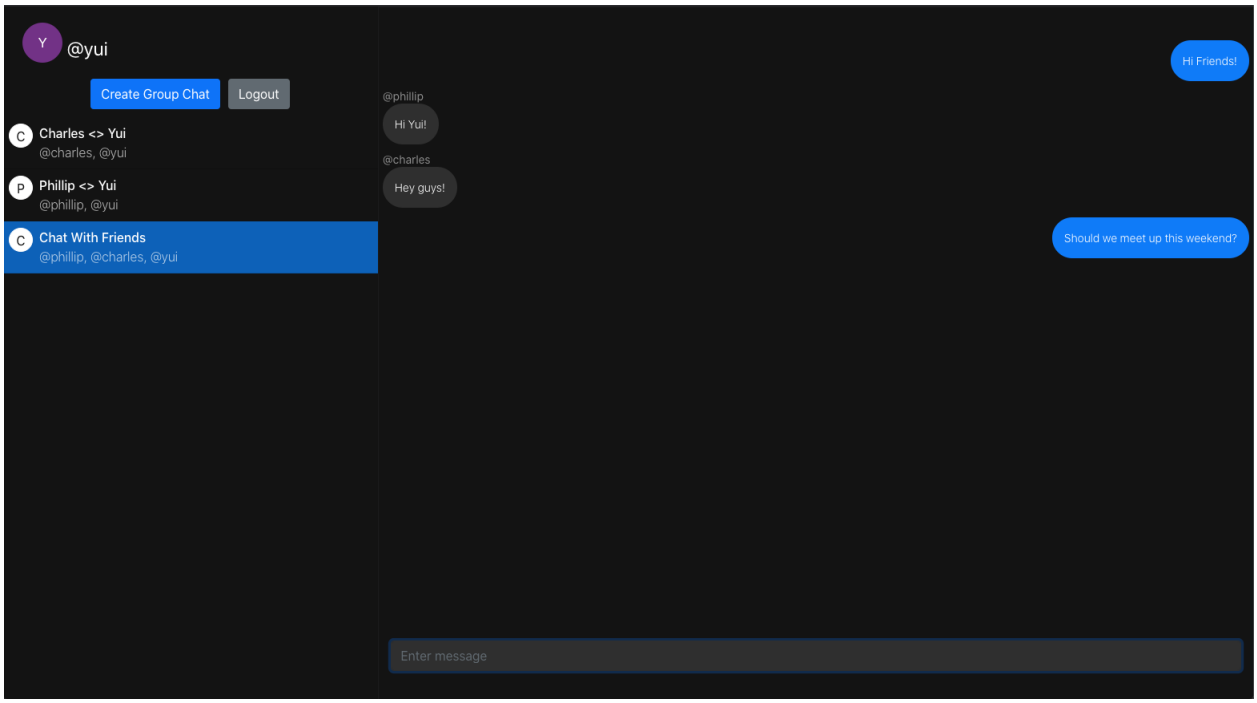

Fig. 2. Group Chat With Users @phillip, @charles and @yui.

that the subscriber is listening to. An example subscription is the one that waits for new messages to be created, shown below:

```
newMessage(gid: Int!): Message
```

This subscription essentially subscribes the user to listen for all new messages that are sent associated with the group id.

4.2.2 Registration. Registration requires the user to input a username and password. An RSA key pair is then generated client side using a built-in browser cryptography library. The username and the public key, from the generated key pair, is sent to the server via a GraphQL mutation. The private key, the hashed password, and the username is stored in the browser local storage. Local storage is chosen over cookies because it never expires, and the max size of local storage is 5MB, whereas cookie max size is 4096 bytes. This entire local storage content is encrypted and sent to the server as a JSON, referred to as the "user file", storing all of the user's relevant keys, their username, and their hashed password. Sending this "user file" is also accomplished using a GraphQL mutation.

4.2.3 Login and Logout Process. Login requires the user to input their username and password. A GraphQL query is executed using the username to obtain the related "user file" JSON. This encrypted user information is decrypted using the hashed password, which is entered during login, as the key for AES decryption. If the resulting data is valid, the user login is validated and the "user file" content is loaded back into the browser local storage. Logging out will result in the local storage being cleared.

4.2.4 Groups. When a new group is created, an AES group key is generated by the group creator, and a GraphQL mutation initializes the group on the server. The group AES key, which is encrypted by each group member's public key, is then fanned out to the members of the group. The other users in the group receive the encrypted group AES key though a GraphQL subscription, which monitors when users are added to groups. Once a user receives the encrypted AES key, it is decrypted using the user's private key. The unencrypted AES key is then stored in browser local storage, and is used to encrypt subsequent messages sent to the chat. Finally, the metadata for the user is updated with the newly encrypted contents and pushed to the server.

4.2.5 Chat Feed. When a user opens a group, the last 50 messages for the group are obtained using a GraphQL query. This query gets a message object for each message that was sent, which includes the encrypted message content and metadata such as the message timestamp, the sender of the message, and the message content type, such as text or media. Then, the encrypted message content is decrypted on the frontend, using the AES key associated with this group which was loaded into the browser local storage either on login or group creation. This decrypted message content is ultimately what is displayed on the screen as plaintext. If a user scrolls past the currently loaded messages, a loading indicator appears. A new GraphQL query is then executed which uses pagination to get the next batch of 50 recent messages.

A GraphQL subscription is created that listens to new messages for each group. When a new message is received from this subscription, the content is decrypted and displayed at the bottom of the

SQL 1 - 3

id	content	cType	createdAt	updatedAt	groupId	userId
1	U2FsdGVkX18rAXmU2RUSZdrwukjynCTMD7nX2c5F/4VsmuQnsFVgoRAIKSxac612+zlnWySxbLQSJB9g7fVO1g==	text	2020-11-03 04:43:15.160 +00:00	2020-11-03 04:43:15.160 +00:00	1	2
2	U2FsdGVkX18WHRbaprOkxuskvVEGs5clz7nvNXVxZM+hndOYsfH4OAerDAebkIG8	text	2020-11-03 04:43:16.283 +00:00	2020-11-03 04:43:16.283 +00:00	1	2
3	U2FsdGVkX19E20yun9nhwWk4ruBvwXdB0GgPRv+WuqleYQnD6/ONfp0DM1FUgRl2	text	2020-11-03 04:43:17.259 +00:00	2020-11-03 04:43:17.259 +00:00	1	2
4	U2FsdGVkX1/X8QrK1W6/DdpHopP7FWoP4t6p+fdmW9dBl4m2URkai123mfQCz84q	text	2020-11-03 04:43:18.443 +00:00	2020-11-03 04:43:18.443 +00:00	1	2
5	U2FsdGVkX1+olKNgvcAb86rNsd0vSbJnsB0IDPIZK3d3yBZWrNWEo0eN9hME8HOE	text	2020-11-03 04:43:19.632 +00:00	2020-11-03 04:43:19.632 +00:00	1	2
6	U2FsdGVkX19F5lIlptugM7y1hrNWhCsUhT7a+HyURrtdDewA8h62x5Y43L2yUWCPVBig7gN+l4K0C/XZ8mwjkg==	text	2020-11-03 04:43:21.294 +00:00	2020-11-03 04:43:21.294 +00:00	1	2
7	U2FsdGVkX19JSkQTapnEiEDcRbV9jOCcymqVxnk4OQw3YUSyboTdAYk66jfX/PdkaLryJPFTsDGl5BBVQQh9zg==	text	2020-11-03 04:43:23.046 +00:00	2020-11-03 04:43:23.046 +00:00	1	2
8	U2FsdGVkX18v5a7J63dBnKXMsw14bH+zulvAjXnq9B2YiJw2Z1yrPurqeyyxwuzpfAzW6K6n9xeLzAzC+oe8+w==	text	2020-11-03 04:43:24.388 +00:00	2020-11-03 04:43:24.388 +00:00	1	2
9	U2FsdGVkX1+nXix1lxK2erfteNJRGRtU4hauEMg34aUhJUbpeVfi3GpfxoYH5czm	text	2020-11-03 04:46:24.397 +00:00	2020-11-03 04:46:24.397 +00:00	2	2
10	U2FsdGVkX1/oaRspQNf6FwzyGljLLuy6aw61qHfSuHj4nmfXLPuH9fuuMiSaEN68	text	2020-11-03 04:46:25.331 +00:00	2020-11-03 04:46:25.331 +00:00	2	2

Fig. 3. Screenshot of Encrypted Content in SQL Database.

feed. If the chat feed already encompasses the allocated window, the new message displayed causes the window to automatically scroll to the bottom of the feed. Figure 2 shows the general view of the Chat Feed with messages between three users. Technical challenges for implementing the chat feed involved adding the pagination that enabled loading chats more efficiently.

4.2.6 Chat Input. The chat input is the main way for a user to interact with groups. Once the user finishes typing their message and sends the message to the group, the message is encrypted with the group's AES key, which is located in localstorage. A GraphQL mutation is made with the encrypted message contents, which then creates a new object in the database. Finally, the message contents are published to the other subscribers of the group chat.

5 RESULTS

E2-Chat was successfully implemented using the described tech stack. E2-Chat is available at this link (https://rithik.me/Capstone). All individuals that are new to E2-Chat can register for an account.

In Figure 2, the user signed up for E2-Chat with the username "Yui". The user is then able to create a new chat room by inputting the usernames of the people Yui wanted to message. Yui entered the usernames of her friends, "Charles" and "Phillip" to be added to the group chat. Yui sends a message to the group chat, and Phillip and Charles reply back to the message. These capabilities demonstrate that E2-Chat is a functional messaging service for many users.

5.1 Security

The encrypted messages in our database are shown by Figure 3, presented above. The message content originally sent in plaintext is only stored in the database in its encrypted form. All instances of decryption keys are only located on each user's physical device, meaning the server has no way of decrypting these messages contained in the database. Furthermore, since all the connections between the frontend and the server are also through TLS, the actual message in transit is also encrypted on top of the actual content encryption. Therefore, since plaintext messages are only visible on each end user's device, the service is E2EE. If an attacker were to be part of an attack that tried intercept messages in transit between users and the server, the attacker would only be able to obtain the TLS-encrypted message object, which is further secured by the additional AES encryption which we implemented. If an attacker were to compromise our database or server, they would get access to the AES-encrypted message content (shown in Figure 3).

5.2 Deployment

In order for this project to be available for users, E2-Chat was deployed in two separate parts: the React.js frontend and the Node.js backend. The React.js frontend is compiled using the built in react-scripts library into a static website. It is then deployed on to GitHub Pages where the project is viewable to the public. The Node.js backend is hosted on Heroku, where it can be accessed by the frontend. The PostgreSQL database is also hosted on Heroku and is directly linked to the backend via secrets that rotate continuously to ensure that this sensitive data is not leaked.

5.3 User Interface

The primary user interface aspects include the login and registration page, the group chat feed, the message feed and the chat input. The login and registration page allows users to register for an account by specifying a unique username and a password. They can reenter this information to login and view their group chats and messages. The group chat feed displays a list of group chats in which the user is a member. Users have the option to create a new group chat, which generates a pop-up where they can select a name for the chat and specify usernames of users who they would like to include as members. The message feed component showcases all of the messages that have been sent in a particular group. Initially, the message feed will load 50 messages and, as the user scrolls up to view older messages, the feed will automatically update with these newer messages. The chat input consists of simple text input field that allows the users to type message content that they would like to send to the group. Messages are only sent when the "enter" key is pressed. All components were built using the open-source Bootstrap framework to ensure consistency of design.

6 CONCLUSION

In this paper, the design and implementation for E2-Chat, a purely web-based E2EE messaging service, was outlined. We verified that the messages sent through E2-Chat were E2EE, and that the basic functionalities for a messaging service operated correctly. The E2EE requirement for E2-Chat was validated by verifying that the message contents of a user's messages are encrypted on the server, and

that plaintext messages are only visible on each end user's device. The general messaging functionalities of E2-Chat were verified by having users sign up and message each other using E2-Chat. Additionally, a smartphone was not needed to sign up and use E2-Chat. The completion of this project showcases how a web-based E2EE messaging service can be successfully designed and implemented, which would enable individuals that do not own a smartphone to be able to use an E2EE messaging service. Continued technological innovations such as E2-Chat are needed to maintain user privacy in a world where governments and third party actors are constantly seeking ways to read one's messages.

7 FUTURE WORK

Future work on E2-Chat will include allowing for various file transfer, such as media including images, and other files such as PDF and text files. To send files via E2-Chat, files will be converted into a base64 string and uploaded to the web browser. The use of base64 strings will simulate the file as a text message and will be transmitted to the rest of the group members in the same manner as normal text-based messages. When these messages are received by the recipients in the group chat, the messages will be decoded using the group's key and then will be decoded from base64 into the appropriate file format. The decrypted file can then be downloaded by all of the users in the group to view.

Improvement can also be made to the user interface. Currently, new messages which are received cause the window to refresh the frame, first displaying the top of the feed and simulating the actual scroll to the bottom to display the new message. However, this can be inconvenient as every new message will cause this action, and viewing older messages during new message retrieval will also execute this action. During testing, feedback was also received which indicated some preferred the dark theme of the interface, while others indicated that they would have preferred a light mode theme. Thus, future work will include a toggle feature to allow users to seamlessly switch between these two themes.

ACKNOWLEDGMENTS

Thank you to our technical advisors, Professor Yixin Sun and Professor Madhur Behl, for assistance throughout this project.

REFERENCES

[1] Brooke Auxier, Lee Rainie, Monica Anderson, Andrew Perrin, Madhu Kumar, and Erica Turner. 2019. *Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information.* Retrieved Nov. 8, 2020 from https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/

[2] Andy Greenberg. 2020. *Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information.* Retrieved Nov. 8, 2020 from https://www.wired.com/story/facebook-messenger-end-to-end-encryption-default/

[3] Business Insider. 2016. *Apple is taking on Facebook with an iMessage app store.* Retrieved Nov. 8, 2020 from https://www.businessinsider.com/apple-is-taking-on-facebook-with-an-imessage-app-store-2016-9

[4] Dong-Cheol Jeon, LIUHAOYANG, and Heejoung Hwang. 2019. Design of Hybrid Application Based on GraphQL for Efficient Query for PHR. *2019 International Conference on Information and Communication Technology Convergence* 50, 1 (oct 2019). https://doi.org/10.1109/ICTC46691.2019.8940003

[5] Greg Kumparak. 2014. *Apple Explains Exactly How Secure iMessage Really Is.* Retrieved Nov. 8, 2020 from https://social.techcrunch.com/2014/02/27/apple-explains-exactly-how-secure-imessage-really-is/

[6] Kristina Libby. 2019. *How SMS Works—and Why You Shouldn't Use It Anymore.* Retrieved Nov. 8, 2020 from https://www.popularmechanics.com/technology/security/a29789903/what-is-sms/

[7] Punita Meelu and Rajni Meelu. 2012. Implementation of Public Key Cryptographic System: RSA. *International Journal of Information Technology and Knowledge Management* 5, 2 (Dec. 2012), 239–242.

[8] Jon Porter. 2020. *WhatsApp now has 2 billion users.* Retrieved Nov. 8, 2020 from https://www.theverge.com/2020/2/12/21134652/whatsapp-2-billion-monthly-active-users-encryption-facebook

[9] Rebecca J. Rosen. 2011. *How Your Private Emails Can Be Used Against You in Court.* Retrieved Nov. 8, 2020 from https://www.theatlantic.com/technology/archive/2011/07/how-your-private-emails-can-be-used-against-you-in-court/241505/

[10] Laura Silver. 2019. *Smartphone Ownership Is Growing Rapidly Around the World, but Not Always Equally.* Retrieved Nov. 8, 2020 from https://www.pewresearch.org/global/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equally/

[11] WhatsApp. 2016. *WhatsApp Encryption Overview.* Retrieved Nov. 8, 2020 from https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf