CS594 - Project Report

End-to-End Encrypted Messaging Service - The Signal Protocol

Members:
Snigdha Ghosh Dastidar (656727529)
Rohan Karle Sudarshan(670950064)
Gunashree Channakeshava (650091403)
Chandhu Bhumireddy (675217882)
Abhiram Vasudev (662558284)

GitHub Link: https://github.com/GunashreeC/signalprotocol

Project Overview

Our project aims to develop a secure messaging service based on the Signal Protocol, utilizing its robust encryption techniques to ensure end-to-end security for communication. The system comprises three main components: the sender, receiver, and a relay server. The relay server plays a pivotal role in managing long-term keys and facilitating secure message exchange between users. Our implementation focuses on ensuring seamless and secure communication while adhering to the principles of the Signal Protocol.

Implementation Details

Protocol Implementation:

At the core of our project lies the implementation of two key features of the Signal Protocol: X3DH (Extended Triple Diffie-Hellman) and the Double Ratcheting Algorithm. X3DH facilitates the initial key exchange, establishing a secure connection between parties, while the Double Ratcheting Algorithm ensures forward secrecy by encrypting each message with a new key.

System Components:

The relay server serves as the backbone of our messaging service, managing long-term keys and mediating message exchanges securely. The sender initiates communication by fetching prekey bundles from the server, utilizing X3DH for the initial message and the Double Ratcheting Algorithm for subsequent messages. The receiver publishes initial and prekeys to the server and manages keys for maintaining secure communication through double ratcheting.

Application Features:
Our application features a user-friendly interface enabling login, contact listing, and a chat window for message exchange. Real-time communication is facilitated via WebSocket connection, ensuring seamless interaction between users. Secure links to the server enable initialization, encryption, and decryption of messages, along with storage and retrieval of chat data, ensuring the confidentiality and integrity of communication.

Technical Stack:
Our technical stack includes a web application employing HTTPS for secure communication, with the backend implemented in JavaScript. We used HTTP server as it focuses on client-to-client communication. We used local MySQL storage as extracting and inserting data into the table is easier using PHP.
Real-time communication is enabled through WebSocket, while client and server-side components encompass UI/UX design, database management, messaging engine, and API implementation.

Key Exchange:
Key exchange is facilitated through the X3DH protocol, establishing a shared secret key between parties without simultaneous online presence. This protocol lays the foundation for secure communication, enabling subsequent message encryption and decryption.

Ratcheting Mechanism:
The ratcheting mechanism ensures forward secrecy by continuously updating shared secrets and message keys. The Diffie-Hellman Ratchet and Symmetric-Key Ratchet, using the concept of KDF, work in tandem to derive new keys for each message, preventing compromise even if a single key is compromised.

Message Encryption and Decryption:
Messages are encrypted and decrypted using message keys derived from the symmetric-key ratchet. Each message contains a header with the current ratchet public key and message count, facilitating synchronization and state management between sender and receiver.

Practical Implementation Steps

1. Choosing a Suitable Cryptographic Library: Utilized a well-supported cryptographic library capable of handling Elliptic Curve cryptography, AES encryption, and HMAC. Used “libsodium” and “crypto” javascript modules to implement the protocols.

2. Implement X3DH Protocol: Coded logic for the X3DH protocol to establish initial shared secrets and securely seed the double ratchet mechanism. Generate key pairs, manage public key distributions securely, and ensure safe storage of private keys using SQL database. We implemented the Curve22519 elliptical curve to generate long term keys, a pair of private key and its public key. Generated a set of prekeys as well that is used for initial key exchange. The receiver fetches the prekey bundle and performs ECDH (Elliptical Curve Diffie-Hellman) key exchange using their own private and selected prekey's public key.

3. Implement Double Ratchet Algorithm: Developed Diffie-Hellman logic and symmetric-key ratchets, managing key derivation, message key generation, and state synchronization. Implemented the KDF chain concept by constructing HMAC and HKDF functions. After the initial Diffie-Hellman key exchange, the shared secret is derived using SHA256 - KDF to generate symmetric keys for encryption and authentication and chain keys for forward secrecy.

5. Integrate Encryption and Decryption: Encrypt outgoing messages with current message keys and decrypt incoming messages using stored or derived keys, ensuring synchronization and state management. Each session has its own fresh set of symmetric keys, AES. The outgoing messages are authenticated using MAC using SHA 256. Upon receiving the message, the receiver decrypts it using their current symmetric key and verifies it with MAC. After it is successfully decrypted, the recipient advances their own ratchet state.

6. Handle Messages: Implemented logic to handle out-of-order messages and lost messages, ensuring robustness and reliability in communication. Using JavaScript built-in libraries such as async and await functions, we established an asynchronous system for the application.

Through diligent implementation of these steps and adherence to the principles of the Signal Protocol, our messaging service provides a secure platform for confidential communication, safeguarding user privacy and data integrity.

Frontend and Database Implementation

For the frontend implementation, we opted for React.js, a popular JavaScript library for building user interfaces. React's component-based architecture allowed us to create modular and reusable UI components, enhancing code organization and maintainability. Leveraging React's virtual DOM, we achieved efficient rendering performance, ensuring smooth user interactions even in complex applications. Features such as login, contact listing, and chat windows were developed as separate React components, promoting code reusability and scalability. Additionally, we utilized WebSocket for real-time communication, enabling seamless message exchange between users without the need for page refresh.

As for the database implementation, we chose MySQL for its reliability, scalability, and robust feature set. MySQL's support for ACID properties ensured data integrity, crucial for storing sensitive user information and message data securely. We designed the database schema to accommodate user authentication data, contact lists, chat messages, and prekey bundles, optimizing for efficient data retrieval and storage. Utilizing MySQL's indexing capabilities, we ensured fast query execution, facilitating smooth user experience even with large datasets. Integrating MySQL with our backend operations streamlined data management, enabling seamless interaction between the frontend and backend components of our messaging service.

Results and Screenshots

The app typically starts with a user creating an account, where they provide necessary information such as a username. Upon registration, the app generates unique keys for the user using the X3DH (Extended Triple Diffie-Hellman) protocol, ensuring secure communication. Once the account is created, the user can log in using their credentials, which authenticates them against the database and allows access to the app's features.

After logging in, users can create chat rooms for group communication. They can define the room's name, adding members to these rooms involves searching for the preexisting users from the app's database using the username, who have already created accounts and generated their own encryption keys. The chat messages within these rooms are encrypted using the Double Ratchet Algorithm, guaranteeing end-to-end encryption and security. When a user is offline, messages intended for them are stored and delivered upon their return to the app. The user's status is displayed with a red dot to signify their offline status, and once they are online, their profile is marked with a green dot, indicating their availability for communication. This status update ensures efficient communication and helps users know when their contacts are reachable.

username	identityKeyPair	ephemeralKey_Pub	ephemeralKey_Private	signedPreKey
snigdha	82,42,191,112,177,170,236,55,202,50,149,103,230,26...	33,158,100,118,246,83,88,124,45,201,205,147,126,36...	185,70,246,47,233,245,176,24,182,151,71,255,16,118...	142,229,223,5
gunashree	9,188,232,211,213,224,49,221,4,118,214,153,225,66,...	202,13,100,139,81,237,25,249,179,71,26,48,88,114,8...	164,193,142,117,244,32,90,250,191,84,109,52,30,194...	35,147,2,3,203
chandhu	30,127,66,173,161,164,10,53,240,28,26,202,241,163,...	150,132,220,227,139,253,208,239,150,241,32,23,121,...	180,201,59,183,199,21,234,179,78,161,248,93,219,16...	231,31,234,22
rohan	117,98,9,143,5,112,3,60,5,221,142,234,244,69,148,1...	120,250,132,26,195,222,115,82,168,163,73,49,235,9,...	246,35,245,225,176,130,248,14,9,76,188,181,42,2,91...	222,117,130,2
abhiram	242,182,185,195,63,11,207,45,37,116,54,244,24,76,1...	73,228,121,211,155,36,200,12,132,8,158,190,103,113...	215,232,206,182,93,62,23,42,96,39,128,122,205,176,...	120,246,189,3
new user	238,192,160,87,38,106,253,177,168,95,129,183,123,6...	233,64,51,228,255,95,243,79,101,51,115,151,87,154,...	42,227,66,243,183,194,166,250,0,160,251,127,202,94...	112,62,37,38,1

Fig 1. Server Database when the users are created during X3DH.

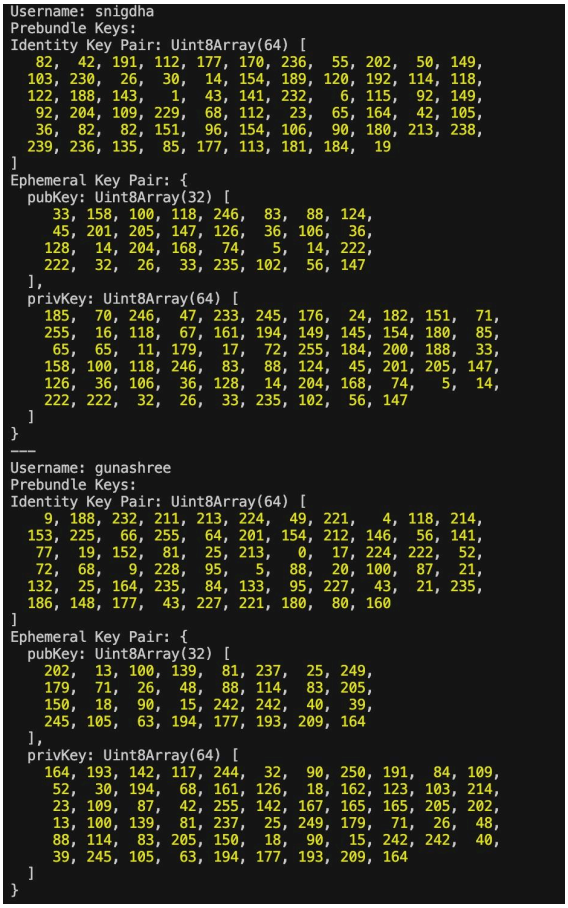


Fig 2. Stored in server db when the user registers for the first time

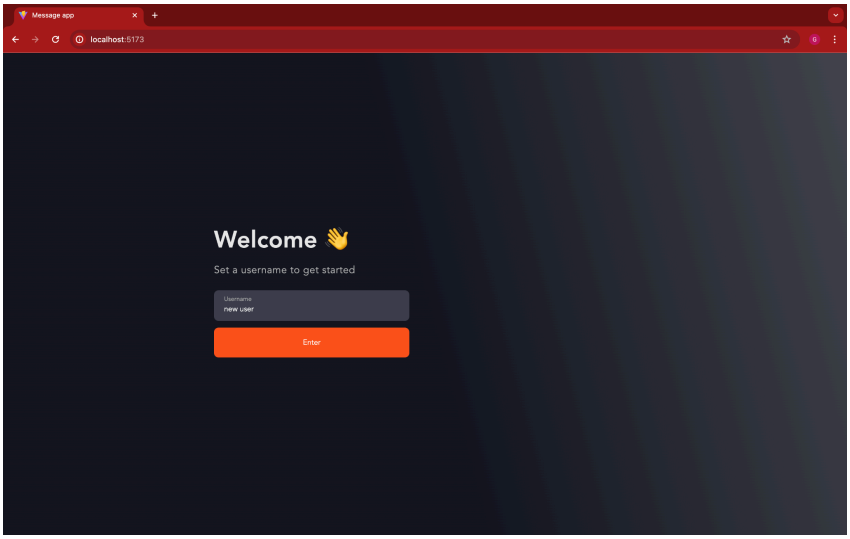


Fig3. This is the welcome page of the project and when the user signs up for the first time.

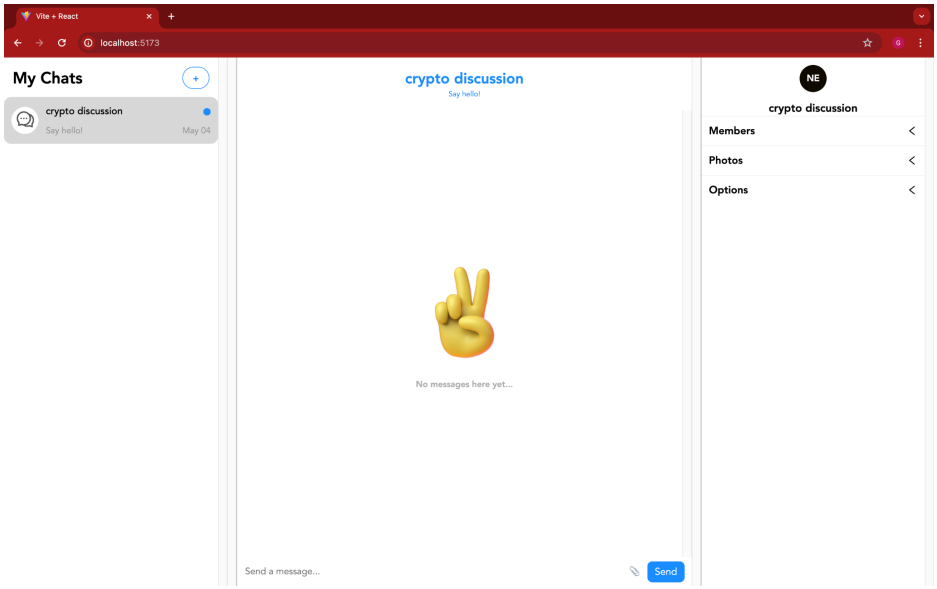


Fig 4. This is the chat page when the user creates a new chat room

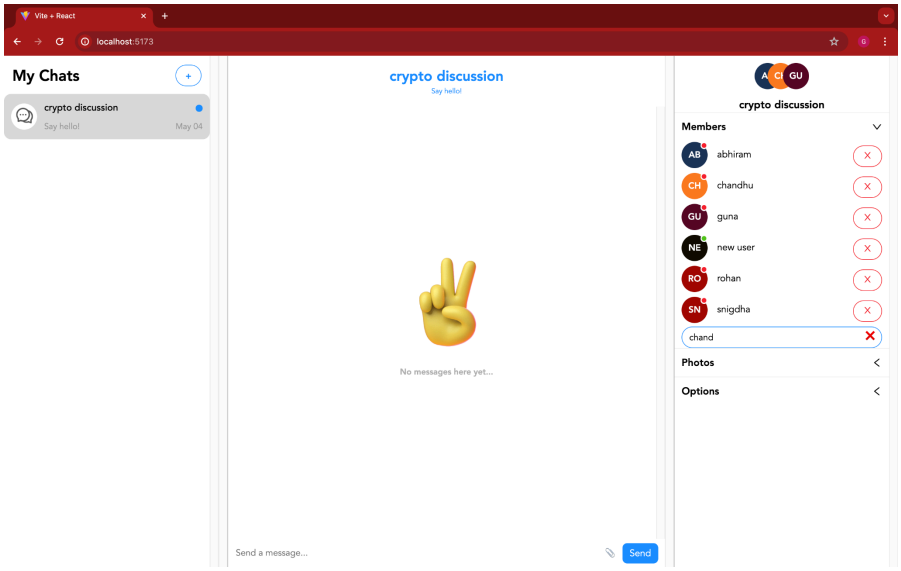


Fig 5. This is the chat page when the user adds members to the chat room.

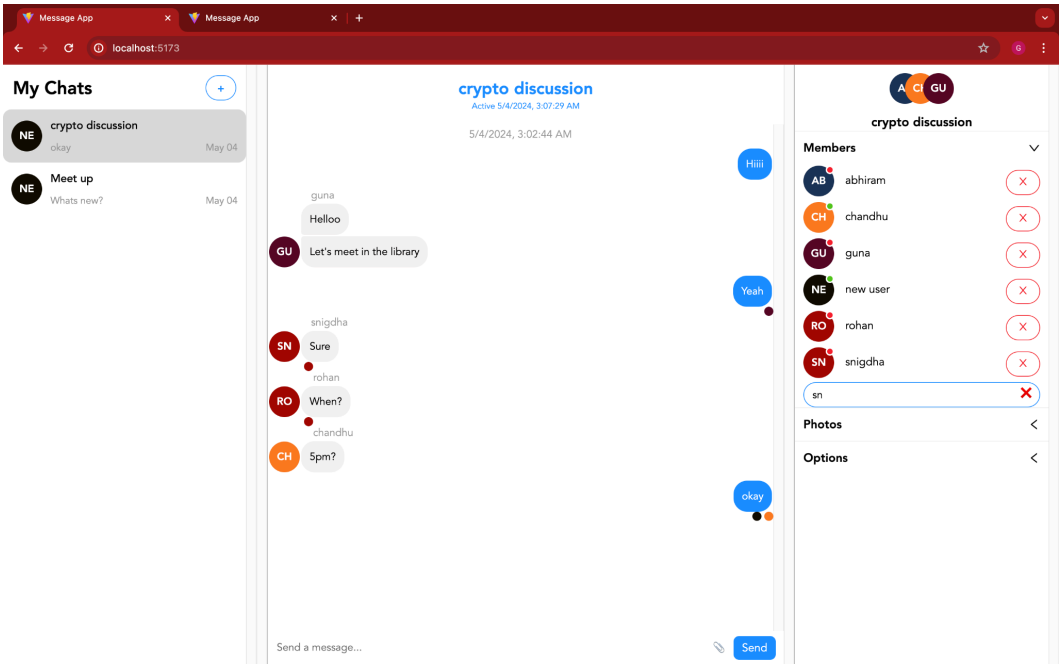
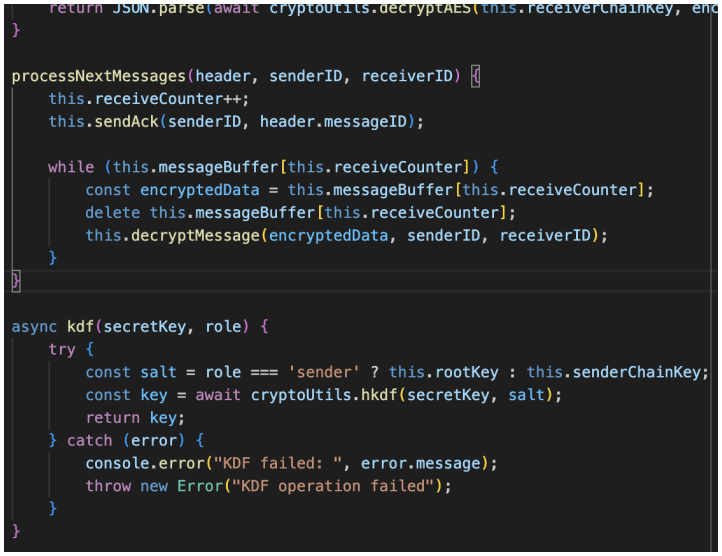


Fig 6. This is the chat page when the user chats in the group. The green and the red dot on the user profile signify that the user is offline or online. Red signifies offline and green signifies online.

Code Snippets

```
module.exports = {
    getRandomBytes: function (size) {
        return crypto.randomBytes(size);
    },
    createKeyPair: async function () {
        await sodium.ready;
        const { publicKey, privateKey } = sodium.crypto_sign_keypair();
        return {
            pubKey: publicKey,
            privKey: privateKey
        };
    },
    generateHash: async function (data) {
        await sodium.ready;
        return sodium.crypto_generichash(32, data);
    },
    Ed25519Sign: async function (privKey, message) {
        await sodium.ready;
        const signature = sodium.crypto_sign_detached(message, privKey);
        return signature;
    },
    Ed25519Verify: async function (pubKey, msg, sig) {
```

Fig 1. X3dh functions that creates keypairs and signed prekeys



```
    return JSON.parse(await cryptoUtils.decryptAES(this.receiverChainKey, enc
}

processNextMessages(header, senderID, receiverID) {
    this.receiveCounter++;
    this.sendAck(senderID, header.messageID);

    while (this.messageBuffer[this.receiveCounter]) {
        const encryptedData = this.messageBuffer[this.receiveCounter];
        delete this.messageBuffer[this.receiveCounter];
        this.decryptMessage(encryptedData, senderID, receiverID);
    }
}

async kdf(secretKey, role) {
    try {
        const salt = role === 'sender' ? this.rootKey : this.senderChainKey;
        const key = await cryptoUtils.hkdf(secretKey, salt);
        return key;
    } catch (error) {
        console.error("KDF failed: ", error.message);
        throw new Error("KDF operation failed");
    }
}
```

Fig 2. Double Rachet functions

Conclusion

In conclusion, our project successfully leverages the robust security features of the Signal Protocol to deliver a secure messaging application that prioritizes user privacy through end-to-end encryption. By meticulously implementing key security protocols such as X3DH for initial key exchange and the Double Ratcheting Algorithm for message encryption, we have developed a messaging service that ensures the confidentiality and integrity of user communications. Throughout the project, our focus remained on providing a seamless and secure platform for users to interact, safeguarding their privacy in the digital realm.

We would perform a separate page to register a new user to enhance this project. We would analyze further to find ways that this implementation might be attacked. With the completion of this project, we have demonstrated the importance of privacy in digital communication and the efficacy of utilizing advanced encryption techniques to uphold it. Moving forward, we anticipate that our messaging application will serve as a testament to the significance of incorporating robust security measures into modern communication platforms. As technology continues to evolve, we remain committed to advancing the field of secure messaging and contributing to the ongoing discourse surrounding privacy in the digital age.

References

https://github.com/narayanpai1/Signal-Protocol-Implementation/blob/master/messenger.js
https://signal.org/docs/specifications/x3dh/
https://github.com/signalapp/libsignal-protocol-javascript/tree/master/src
https://github.com/harveyconnor/curve25519-js/tree/master/src
https://github.com/PeculiarVentures/2key-ratchet/tree/master/src/protocol
https://flownet.com/ron/code/djbec.js
https://chatengine.io/
https://signal.org/docs/specifications/doubleratchet/