1. **What we did**

`e2ee-im` is an end-to-end encrypted command line interface instant messaging application. The communication protocols behind `e2ee-im` uses end-to-end encryption (E2EE), which ensures only the users or clients that are directly involved in a specific communications stream are able to read or decode messages. To achieve this, our system uses elliptic curve cryptography (ECC), Elliptic Curve Digital Signing Algorithm (ECDSA), Elliptic Curve Diffie-Hellman Key Exchange (ECDH), Advanced Encryption Standard (AES), and many other cryptographic methods and techniques to securely send encrypted messages and ensure the privacy of our users.

2. **What are the components of the project**

The first major component of this application is the client-server authentication protocol. The application, when being executed by a user for the first time, generates a private key and the corresponding public key using elliptic curve cryptography (secp256k1) which is stored locally on the client's system. Next, the client system will then attempt to establish a connection with the websocket server on the backend. When prompted to establish a connection by some client, the websocket server first executes a socket.io middleware method that does some preliminary authentication of the connection client before the client is actually allowed to establish the connection. For new users that are connecting to the node.js websocket server for the first time, the middleware method creates an entry in the postgres database that records the user's username, public key, and the time of creation. On the other hand, for returning users, the middleware function fetches the client's information and ensures that the client's current public key matches the public key that is stored on the server. After a websocket connection is established between the client and server, the server generates a random token which it sends to the client. The client takes this token and creates a signature of the token using the elliptic curve digital signing algorithm (ECDSA) using its private key. The server then verifies the signature created by the client by using the public key of the user that is stored in its database. If this verification step passes, the client and server have successfully completed the authentication process, otherwise, the server forcefully disconnects the client and terminates the connection.

The second major component of our system that protects the privacy of our users is the Elliptic Curve Diffie-Hellman key exchange (ECDH) protocol. For our implementation, the computations behind the ECDH protocol is largely handled on the client side. Given that the server has designated one user as a hypothetical Alice and Bob, Alice will initiate the protocol by sending her unique CUID and public key to Bob. After receiving Alice's message that is passed through the server, Bob will ask the server if these two values match up to the CUID and public key pair held by the server, as in our implementation it is assumed that the server is the authority. After having Alice's CUID and public key, Bob then sends his CUID and public key with a signature, which uses his private key to sign the string that contains both of their CUIDs and public keys. When Alice receives this message, she first verifies the signature to check Bob's identity and whether he's using public and private keys properly. Then she also needs to send a signature signing the same message with her private key and send that to Bob for him to verify her identity. Finally after Alice and Bob have established trust that each other is who they say that they are and that each user holds the correct private key, they each privately use the associativity and closure properties of ECC (as each client's public key is created by multiplying their private key with the predefined generator point G) to calculate a shared symmetric key.

The third and last major component of our system is the encryption of client to client communications using the Advanced Encryption Standard (AES). AES is a symmetric block cipher established by the U.S. National Institute of Standards and Technology, and is a symmetric-key encryption algorithm, meaning the same key that is used to encrypt a message is also used to decrypt a message. Thus, we use the symmetric key calculated using ECDH, which both clients have access to to both encrypt and decrypt the messages and communications between clients.

As previously stated, our system is composed of a client and a server. The client is written using Python and does the majority of the calculations for performing ECDH. To support operations by and between the client and server we have implemented four Python classes: Secp256r1(ECC), Point, Inf(infinity) and a Client class. The class Secp256r1, which is a specific predefined elliptical curve, handles all operations that require elliptic curve math by overloading the necessary operators. The following two classes, Point and Inf, are helper classes for the Secp256r1 class. The class Point instantiates a point on the curve defined by Secp256r1. The class Inf represents the point of infinity on the curve defined by Secp256r1. Finally, the Client class acts as a helper class to the true client by abstracting methods required for exchanges defined in the ECDH protocol.

3. What are guarantees to the users of e2ee-im

The first component of our system, which is the client-server authentication protocol, prevents malicious users from spoofing usernames by requiring all users to authenticate using ECDSA. For instance, our system ensures that a client with the username "Bob" is the same client "Bob" the next time they connect with the server. This is because only Bob's client, which has access to the private key, is able to create a signature of the token provided by the server, that is verifiable by the server using the corresponding public key stored by the server's database. A client that is pretending to be Bob would not be able to generate a valid signature as they do not have access to the private key that corresponds to the public key stored by the server that is used to verify the signatures. It is also important to note that the token provided by the server is randomly generated each time by a cryptographically secure pseudo-random number generator

(CSPRNG) to ensure that the signature is different for every session. The server essentially
The second component of our system, the ECDH protocol that generates a shared symmetric key between two clients, ensures that two clients can securely exchange keys over a public network. The resulting symmetric key is only known by the two clients involved during ECDH. The server in this case only acts as an intermediary that facilitates the communication between the two clients.

E2ee-im has some serious strengths as a program, the chats sent using the program are properly encrypted using ecc and all random values are generated using a csprng. All messages sent between two clients can only be decrypted by this pair of clients that shared their own symmetric key, notably the server has a copy of this key and no honest but curious man in the middle attacks could obtain this symmetric key. The identity of each client is verified through signatures which also affirms they are using private key and public key properly.

Given the project nature of e2ee-im and the compressed timeline of its development there are improvements that can be made. One of the glaring issues with our implementation is that the server has a list of people’s CUID who have connected to the server. This could be problematic because if a user's CUID is found on their local computer by an adversary it would allow for the adversary to prove that they have connected to the server. Another issue with the implementation of e2ee-im is that the server knows when a given person is online. The server knowing when a user is online is problematic for the same reason that the server having a list of people who have connected to the server is. Finally e2ee-im offers no help to the users in keeping their sensitive information safe, such as their private and symmetric keys. An inherent issue with a symmetric key system is that if either users symmetric key is stolen then messages from both parties can be decrypted.