CREATING A SECURE MOBILE MESSAGING APP

Ali Aljahmi

CIS-446-002

Project Overview

This project is a secure chat application built using Python. It allows two users to communicate over a network using encrypted messages. One side runs a server, and the other runs a client. The app uses AES encryption provided by Python's cryptography library, specifically through a tool called Fernet. The goal of the project is to demonstrate how encryption can be used to secure communication and protect privacy.

What is AES

AES stands for Advanced Encryption Standard. It is a symmetric encryption algorithm. This means the same secret key is used to both encrypt and decrypt messages. AES is fast and secure, which is why it is used widely in modern applications such as messaging, file storage, and web traffic.

What is RSA

RSA is an asymmetric encryption algorithm. It uses two keys instead of one. A public key is used to encrypt data, and a private key is used to decrypt it. RSA is very useful when two users want to share a secret key over a network. It is slower than AES and is better used for securely exchanging small pieces of data like encryption keys rather than entire conversations.

Why AES Was Chosen Instead of RSA

This project uses AES because it is faster and more suitable for encrypting full messages in a chat app. Since the same users are talking throughout the session, sharing the key once at the beginning is enough. RSA would be more appropriate if we needed to exchange the AES key securely or work with unknown users. AES provides strong security with better speed, making it the better choice for this app.

How the Code Works

Starting with the Cryptography Library

The cryptography library in Python provides easy tools for secure encryption. Fernet is one of those tools. It uses AES encryption under the hood and ensures the messages are also authenticated, which means it checks that the message was not changed. Fernet takes care of all the complex encryption steps, so we can focus on writing the chat logic.

How the Server Works

1. The server starts by creating a Fernet key using:
2. key = Fernet.generate_key()

This key is printed out so the user can copy and paste it into the client code.

3. The server sets up a network socket and listens for one incoming connection.
4. Once a client connects, the server launches a new thread. This thread listens to incoming encrypted messages, decrypts them using the Fernet key, and prints them.
5. At the same time, the server waits for the user to type a message. It encrypts the message and sends it to the client.
6. This process continues, with both sides sending and receiving encrypted messages until one side stops the program.

How the Client Works

1. The client script includes the same Fernet key that was printed by the server. It uses this key to create a Fernet object.
2. The client connects to the server's socket.
3. Just like the server, the client starts a new thread. This thread listens to messages from the server, decrypts them, and prints them.

4. While that thread is running, the main part of the program waits for the user to type a message, encrypts it using Fernet, and sends it to the server.
5. The client and server keep running until the user closes the program.

Threat Models and Potential Vulnerabilities

Threats Addressed

- Eavesdropping is prevented. Messages are encrypted, so someone watching the network cannot read them.
- Man in the middle attacks are avoided if the secret key is kept safe.
- If the server is compromised but the key is not, the attacker still cannot read past messages.

Potential Vulnerabilities

- The key is shared manually. If it is sent through an unsafe channel, an attacker could use it to read or send messages.
- There is no identity verification. Anyone with the key can impersonate either side.
- The same key is used for the whole conversation. If it is stolen, all past messages are exposed.
- Replay attacks are possible. An attacker could record a message and send it later. Using message timestamps would help stop this.

Conclusion

This project demonstrates how to create a simple but secure chat application using Python and AES encryption. It explains the use of Fernet for encryption and shows how the server and client communicate securely. Although this version uses a basic manual key sharing method, it still protects the privacy of messages. Future updates could include RSA for key exchange, identity checks, and rotating keys to make the system more secure and reliable.