Secure Chat App Project

Introduction

This document provides a comprehensive overview of a secure chat application built with end-to-end encryption. The project demonstrates how modern cryptographic principles can be integrated into a web-based chat platform to ensure private and secure communication. It leverages a Python backend for real-time data handling and a pure JavaScript frontend for all cryptographic operations, ensuring that the server never has access to the plaintext messages.

Abstract

The Secure Chat App is a private, real-time messaging application that guarantees end-to-end encryption (E2EE) using public-key and symmetric-key cryptography. It uses RSA for secure key exchange and AES for fast message encryption. The application is designed so that messages are only decrypted on the client side, ensuring privacy and confidentiality. Optional features, such as message rephrasing and summarization, are integrated using the Gemini API.

Tools Used

- **Python:** The core backend language.
- **Flask-SocketIO:** A Python library that enables real-time, bi-directional communication between the server and clients.
- **RSA/Web Crypto API:** Used for public-key cryptography to securely exchange session keys. RSA is used for its asymmetric encryption capabilities.
- **AES/Web Crypto API:** Used for symmetric-key cryptography. This is a much faster method for encrypting the actual message content.
- **HTML, CSS (Tailwind CSS), & JavaScript:** The foundational technologies for the user interface and client-side logic.
- **Gemini API:** An external service used to power LLM-based features like message rephrasing and conversation summarization.

Steps Involved in Building the Project

1. **Backend Setup:** A Flask-SocketIO server (`app.py`) is created to manage user connections, route messages, and handle key distribution. The server initializes a list of default users (Alice, Bob, Charlie, Diana) and generates a unique RSA key pair for each.
2. **User Authentication & Key Exchange:** When a user joins the chat, they are assigned a session ID. Their newly generated public key is sent to the server and broadcast to all other connected users.
3. **Initial Message Encryption:** When a user sends their first message to a recipient, they first generate a one-time **AES session key**. They encrypt this AES key using the recipient's **public RSA key**, ensuring only the recipient can decrypt it. This encrypted AES key is sent along with the message.

4. **Message Encryption & Decryption:** The actual message content is encrypted using the newly generated AES session key. This encrypted message is then sent to the server. The server forwards the message to the intended recipient without ever decrypting it.
5. **Client-Side Decryption:** When the recipient receives the message, their client-side JavaScript first uses their **private RSA key** to decrypt the AES session key. Once they have the session key, they can use it to decrypt the message content. For all subsequent messages in the conversation, the same AES session key is reused, eliminating the need to re-exchange it.
6. **Real-Time Messaging:** Flask-SocketIO ensures that all these events happen in real-time, providing a seamless chat experience.

Conclusion

This project successfully demonstrates a practical implementation of end-to-end encryption for a chat application, using a combination of public-key and symmetric-key cryptography. By handling all encryption and decryption on the client side, the app ensures that the messages remain private from the server and any potential eavesdroppers. The architecture provides a solid foundation for a truly secure communication platform.