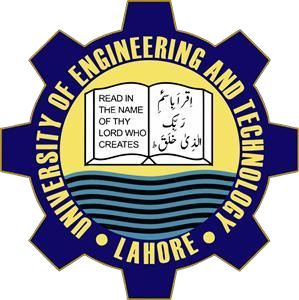
**Project** **Report**



**Submitted by:**

Malik Abdul Wahab 2023-CS-659

Abdullah Ayaz 2023-CS-696

**Submitted to:**

Ma’am Zoha

# Department of Computer Science

# University of Engineering and Technology Lahore, New Campus

# Encrypto Chat:

# A Secure Real-Time Chat Application - Project Report

**1. Introduction**

Encrypto Chat is a secure, real-time chat application designed to provide users with encrypted messaging and file-sharing capabilities. Built using Flask, SocketIO, and SQLite, the application leverages advanced cryptographic techniques (AES and RSA) to ensure the confidentiality and integrity of communications. The project aims to deliver a user-friendly interface with robust security features, making it suitable for private and secure conversations.

This report provides an overview of the project, its architecture, achievements, analysis of its strengths and limitations, a detailed output section describing specific actions and their expected outcomes, and a dedicated section for output screenshots to showcase the application's functionality.

**2. Project Overview**

**2.1 Objective**

The primary objective of Encrypto Chat is to create a secure communication platform that:

* Encrypts messages and files using AES and RSA algorithms.
* Supports real-time messaging via WebSocket (SocketIO).
* Provides a modern, visually appealing user interface using Tailwind CSS and custom animations.
* Stores user data and messages securely in an SQLite database.
* Allows users to search for other users, initiate chats, and manage chat history.

**2.2 Technologies Used**

* **Backend**: Flask (Python web framework), Flask-SocketIO (real-time communication).
* **Database**: SQLite (lightweight relational database).
* **Cryptography**: PyCrypto (AES, RSA encryption/decryption).
* **Frontend**: HTML, Tailwind CSS, JavaScript, SocketIO client.
* **Security**: Passlib (password hashing), RSA key pairs for user authentication and encryption.
* **Others**: Base64 (encoding/decoding), datetime (timestamp management).

**2.3 Key Features**

* **User Authentication**: Secure signup and login with hashed passwords (sha256\_crypt).
* **Encrypted Messaging**: Messages can be encrypted using AES or RSA, with keys managed securely.
* **File Sharing**: Users can upload and download encrypted files.
* **Real-Time Communication**: SocketIO enables instant message delivery and notifications.
* **Chat Management**: Users can delete chat history and switch between encryption ciphers (AES/RSA).
* **Responsive UI**: Glassmorphism design with neon effects and particle animations for an engaging experience.

**3. System Architecture**

**3.1 File Structure**

The project consists of the following key files:

* **app.py**: Core Flask application handling routes, SocketIO events, and encryption logic.
* **crypto.py**: Cipher class implementing AES, DES, and RSA encryption/decryption.
* **database.py**: Utility script to inspect SQLite database tables and data.
* **main.html**: Landing page with links to signup and login.
* **signup.html**: User registration page.
* **login.html**: User login page.
* **dashboard.html**: User dashboard with options to search users or logout.
* **search.html**: Interface to search for other users by username.
* **chat.html**: Chat interface for messaging and file sharing.

**3.2 Workflow**

1. **User Registration**: Users sign up with a username and password, generating RSA key pairs stored in the database.
2. **Authentication**: Passwords are hashed and verified using sha256\_crypt during login.
3. **Chat Initiation**: Users search for others, select a recipient, and start a chat session.
4. **Message Encryption**: Messages are encrypted using AES (with a random key encrypted by RSA) or RSA directly.
5. **Real-Time Messaging**: SocketIO handles message delivery, cipher switching, and chat deletion events.
6. **File Sharing**: Files are encrypted with AES, stored in the database, and downloadable by the recipient.
7. **Database Management**: SQLite stores user credentials, public/private keys, and encrypted messages/files.

**3.3 Database Schema**

* **users**: Stores user information (id, username, hashed password, public\_key, private\_key).
* **messages**: Stores message details (id, sender\_id, receiver\_id, cipher, encrypted\_message, iv, encrypted\_key, file\_name, file\_type, is\_file, timestamp).

**4. Achievements**

**4.1 Security Implementation**

* Successfully implemented AES and RSA encryption for messages and files, ensuring end-to-end security.
* RSA key pairs are generated per user, with private keys stored securely in the database.
* Passwords are hashed using sha256\_crypt, preventing plaintext exposure.

**4.2 Real-Time Functionality**

* Integrated Flask-SocketIO for seamless real-time messaging and notifications.
* Implemented events for message sending, cipher switching, and chat deletion, enhancing user experience.

**4.3 User Interface**

* Designed a visually appealing UI with glassmorphism, neon effects, and particle animations.
* Responsive design ensures compatibility across devices.
* Intuitive chat interface with clear distinction between encrypted and decrypted messages.

**4.4 File Sharing**

* Enabled secure file uploads and downloads with AES encryption.
* Handled file metadata (name, type) and provided download links for recipients.

**4.5 Scalability**

* Modular code structure with a separate Cipher class for encryption logic.
* SQLite database is lightweight and suitable for small to medium-scale deployments.

**5. Analysis**

**5.1 Strengths**

* **Security**: Robust encryption (AES/RSA) and password hashing ensure data confidentiality.
* **Real-Time Communication**: SocketIO provides instant messaging and event handling.
* **User Experience**: Modern UI with animations and clear feedback enhances usability.
* **Modularity**: Separation of concerns (e.g., crypto.py for encryption) makes the codebase maintainable.
* **File Sharing**: Secure file transfer adds significant value to the application.

**5.2 Limitations**

* **Database Storage**: Storing private keys in the database poses a security risk if the database is compromised.
* **Scalability**: SQLite is not ideal for large-scale applications with high concurrency.
* **Encryption Overhead**: RSA encryption for large messages can be computationally expensive.
* **Error Handling**: Some edge cases (e.g., missing keys or invalid files) may cause decryption errors.
* **Lack of DES Usage**: The crypto.py file includes DES encryption, but it is not utilized in the application, indicating unused code.

**5.3 Potential Improvements**

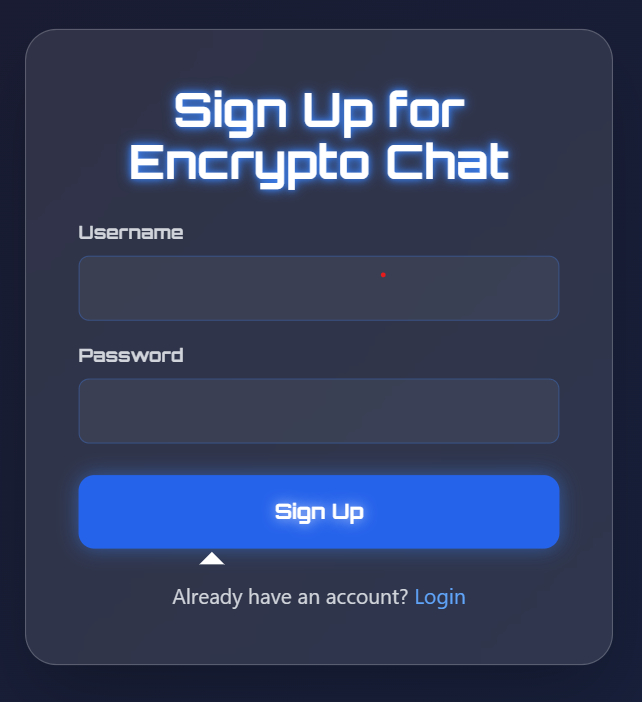
* **Secure Key Storage**: Use a dedicated key management system or client-side storage for private keys.
* **Database Upgrade**: Migrate to PostgreSQL or MySQL for better scalability and concurrency.
* **Hybrid Encryption Optimization**: Use AES for all messages and RSA only for key exchange to reduce overhead.
* **Enhanced Error Handling**: Implement robust error messages and fallback mechanisms for decryption failures.
* **Remove Unused Code**: Eliminate DES encryption from crypto.py or integrate it into the application.
* **Additional Features**: Enhance functionality by adding group chats, message editing, or read receipts.

**6. Output Section**

This section describes specific user actions in the Encrypto Chat application, their expected outcomes, and placeholders for screenshots to visually demonstrate the results. Each action includes a description of what happens when the action is performed and where you can insert a screenshot to illustrate the output.

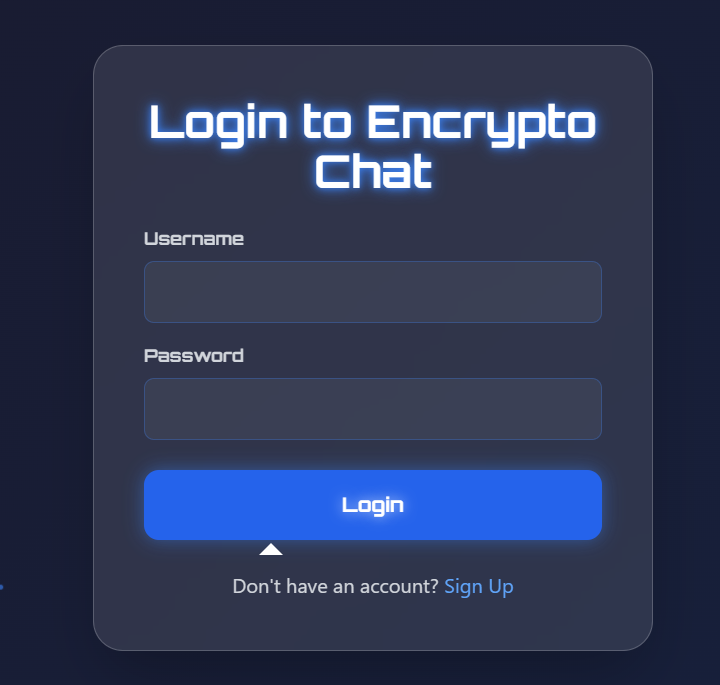
**6.1 User Signup**

* **Action**: A user navigates to the signup page (/signup) and submits a username and password.
* **Expected Outcome**:
  + If the username is unique, a new user is created with hashed password and RSA key pair, and the user is redirected to the login page.
  + If the username already exists, an error message is displayed on the signup page.
* **Screenshot**:

****

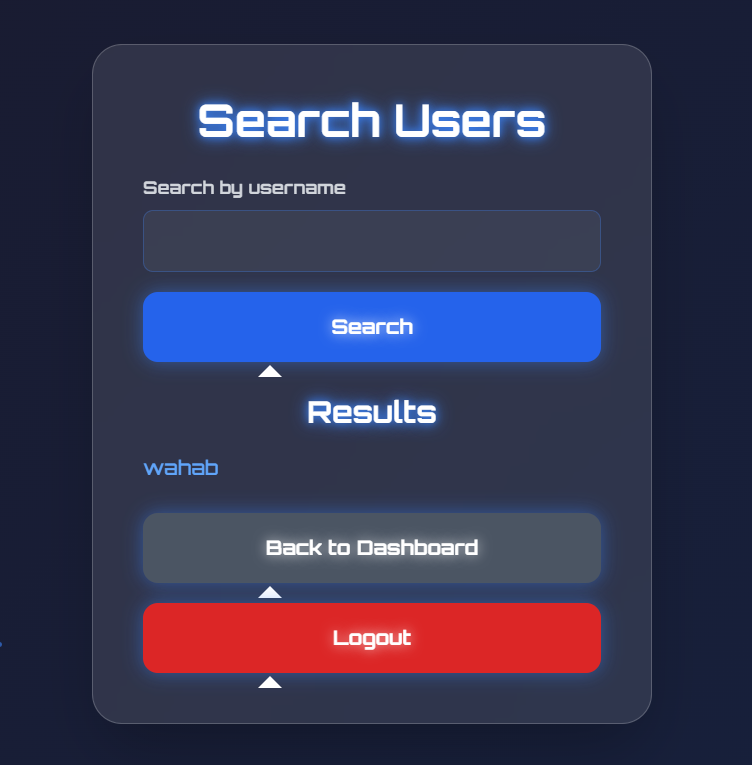
**6.2 User Login**

* **Action**: A user navigates to the login page (/login) and submits their username and password.
* **Expected Outcome**:
  + If credentials are correct, the user is logged in, a session is created, and they are redirected to the dashboard.
  + If credentials are invalid, an error message is displayed on the login page.
* **Screenshot**:

****

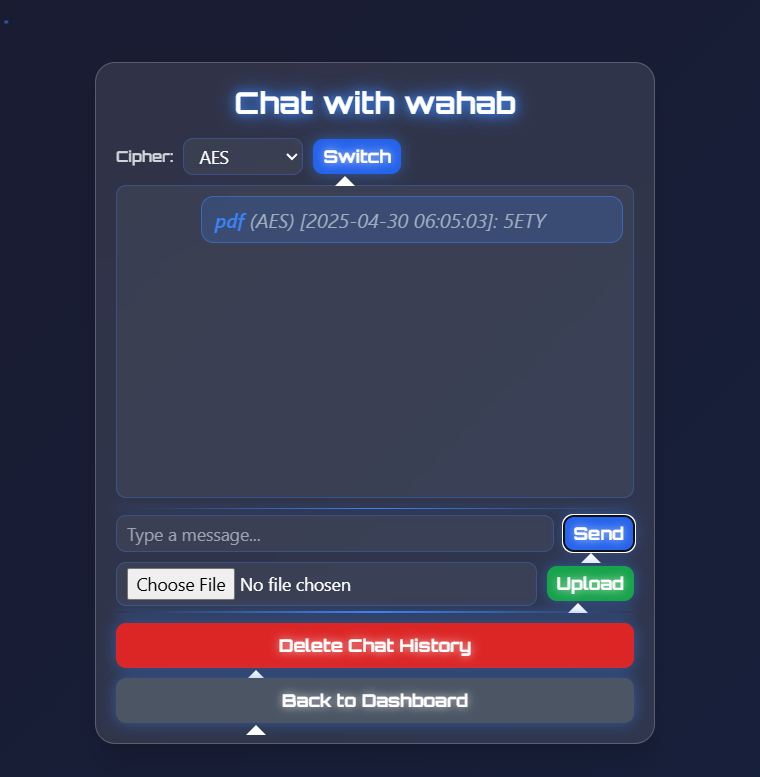
**6.3 Searching for Users**

* **Action**: A logged-in user navigates to the search page (/search) and enters a username to search for other users.
* **Expected Outcome**:
  + A list of matching usernames is displayed with links to start a chat.
  + If no matches are found, the search results section remains empty.
* **Screenshot:**



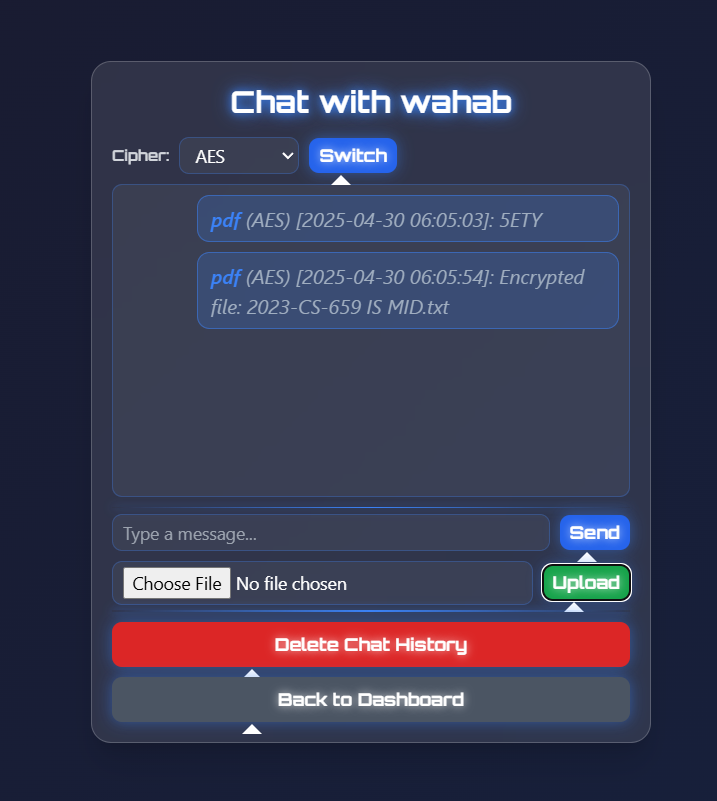
**6.4 Sending a Message**

* **Action**: In the chat interface (/chat/), a user types a message, selects a cipher (AES or RSA), and clicks "Send."
* **Expected Outcome**:
  + The message is encrypted (using the selected cipher), stored in the database, and sent to the recipient via SocketIO.
  + The sender sees the encrypted message in the chat window.
  + The recipient sees the decrypted message in real-time.
* **Screenshot**:



**6.5 Uploading a File**

* **Action**: In the chat interface, a user selects a file and clicks "Upload."
* **Expected Outcome**:
  + The file is encrypted with AES, stored in the database, and a notification is sent to the recipient.
  + The sender sees an "Encrypted file: " message.
  + The recipient sees a clickable link to download the file.
  + An alert confirms successful upload ("File uploaded successfully!").
* **Screenshot**:



**6.6 Downloading a File**

* **Action**: In the chat interface, a recipient clicks the file download link for a received file.
* **Expected Outcome**:
  + The file is decrypted using the recipient's private key and downloaded with the original filename and type.
  + If unauthorized or decryption fails, an error message is returned.
* **Screenshot:**

****

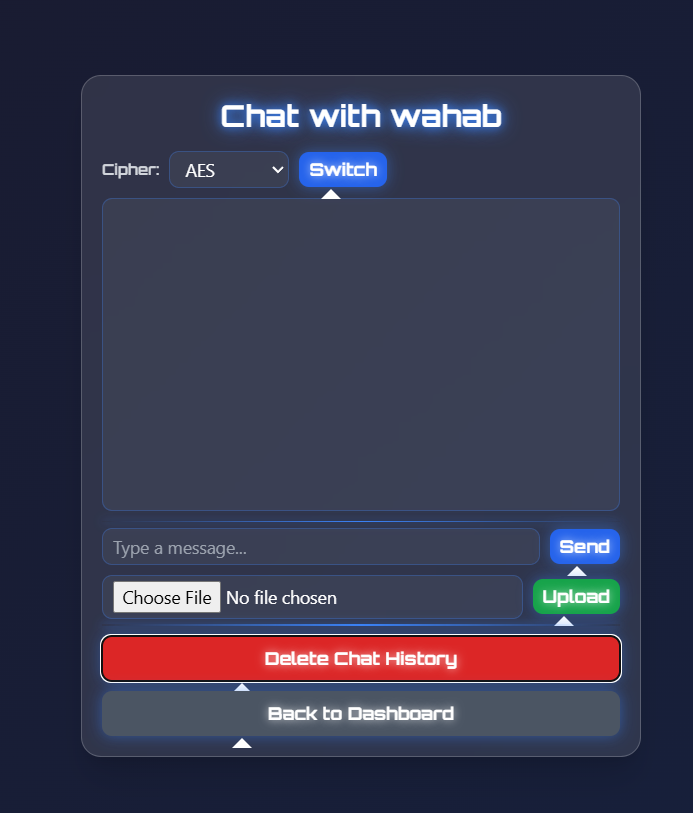
**6.7 Switching Cipher**

* **Action**: In the chat interface, a user selects a different cipher (AES or RSA) and clicks "Switch."
* **Expected Outcome**:
  + The cipher is updated for subsequent messages, and an alert confirms the change ("Cipher switched to ").
  + Both users in the chat receive a notification via SocketIO.
* **Screenshot**:



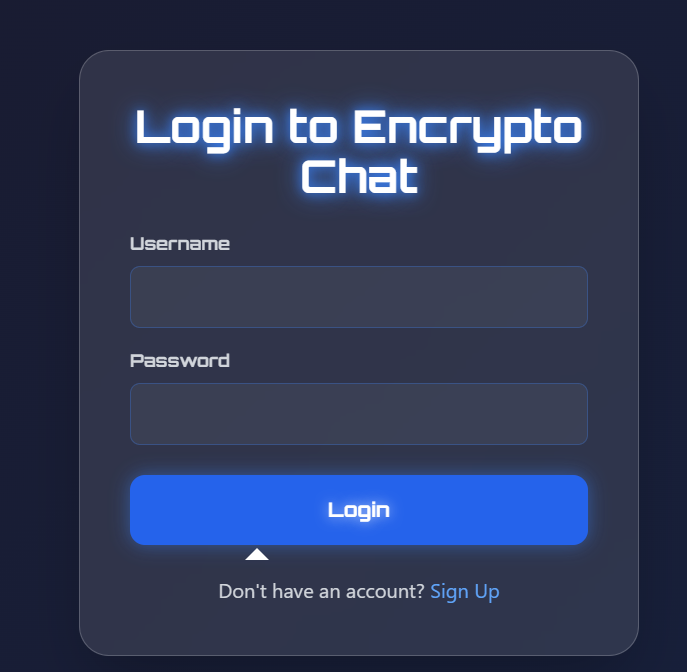
**6.8 Deleting Chat History**

* **Action**: In the chat interface, a user clicks "Delete Chat History" and confirms the action.
* **Expected Outcome**:
  + All messages between the user and the recipient are deleted from the database.
  + The chat window is cleared for both users, and an alert confirms the action ("Chat history deleted successfully").
* **Screenshot Placeholder**:



**6.9 Logging Out**

* **Action**: A user clicks "Logout" from the dashboard or other pages.
* **Expected Outcome**:
  + The user's session is cleared, and they are redirected to the login page.
* **Screenshot**:



**7. Conclusion**

Encrypto Chat successfully implements a secure, real-time chat application with robust encryption, user-friendly design, and file-sharing capabilities. The project demonstrates the effective use of Flask, SocketIO, and cryptographic libraries to deliver a functional and secure communication platform. While there are areas for improvement, such as key storage and scalability, the application meets its core objectives and provides a solid foundation for future enhancements.