Computer Systems Security (B9IS103)

Secure Communications and collaboration Design

Programming and analysis

Group Members: Victor Obi - 10616426
Ankur Viradiya - 10638916

Supervisor: Paul Laird

1. INTRODUCTION

The "MessageApp" is a robust and privacy-centric platform developed using Flask and the `cryptography` library. Designed with security as a top priority, the app ensures end-to-end encryption for user messages, safeguarding sensitive communications from unauthorized access. Leveraging RSA key pairs and AES encryption, the app offers seamless and secure key exchange between users, enabling them to send and receive messages confidentially. The public keys are securely stored in the database, while private keys are accessed solely during user login, reinforcing data protection. With HTTPS encryption and CSRF protection, the app guarantees a safe and user-friendly experience, making it an ideal choice for private communication needs.

1.1 FUNCTIONAL REQUIREMENTS:

I. User Registration and Authentication:
   - Users should be able to register with a unique username and password.
   - The system should authenticate users during login using their credentials.

II. Message Encryption and Decryption:
   - The system should encrypt messages using AES encryption with CFB mode and PKCS#7 padding.
   - Encrypted messages should be decrypted by the intended recipient using the correct key.

III. End-to-End Encryption:
   - The system should ensure end-to-end encryption, where the message remains encrypted from sender to receiver and is decrypted only at the receiver's end (Ermoshina, K., Musiani, F. and Halpin, H., 2016).

IV. Secure Key Exchange:
   - The system should implement a secure key exchange using RSA to securely generate and exchange encryption keys between the sender and receiver.

V. Message Integrity:
   - The system should ensure the integrity of transmitted messages, ensuring they have not been tampered with during transmission.

VI. User Dashboard:
   - Users should have access to a dashboard displaying received messages from other users.

1.2 NONFUNCTIONAL REQUIREMENTS:

VII. Security:
- The system should prioritize security and employ strong encryption algorithms and secure key exchange protocols to protect sensitive data.

VIII. Performance:
- Encryption and decryption processes should be efficient to minimize latency and response times during message transmission.

IX. Scalability:
- The system should be able to handle an increasing number of users and messages without significant performance degradation.

X. Reliability:
- The system should be highly reliable, ensuring that messages are encrypted and decrypted accurately and consistently.

XI. Data Privacy:
- The system should adhere to data privacy regulations and protect user data from unauthorized access or disclosure.

2. Encryption Techniques

Data privacy and security challenges in digital communication include data breaches, cyberattacks, interception, man-in-the-middle attacks, phishing, data retention, and lack of end-to-end encryption. These challenges raise concerns about user privacy, unauthorized access to sensitive information, and misuse of personal data. To address these issues, messaging apps are adopting strong encryption techniques and data protection measures to ensure user trust and confidence in secure communication (Salomon, D., 2003).

Here we are using two encryption techniques:

2.1 AES

AES encryption ensures security and privacy by employing a secret key known only to the sender and receiver, making unauthorized access or data modifications challenging. Its NIST certification and widespread adoption as the standard for encrypting sensitive data and other entities demonstrate its robustness. AES encryption is renowned for its high level of security and efficiency, as it remains unbroken by any known attacks[3].

AES encryption works by using a secret key to transform plaintext data into ciphertext through a series of rounds. Each round involves byte substitution, row shifting, column mixing, and key addition. AES is highly secure and ensures that unauthorized parties cannot decrypt the ciphertext without the correct secret key.

2.2 RSA

The RSA cryptographic algorithm is a widely used public key encryption method that ensures data communication security. It employs two fundamental processes. Firstly, using a public key, it encrypts plaintext data into an unrecognizable ciphertext, making it virtually impossible to retrieve the original data without the encryption password. Secondly, with a private key, RSA decrypts the ciphertext back into its original form. RSA is now extensively employed in web browsers, email programs, mobile phones, virtual private networks, and secure shells. Previously restricted by patents and export laws, RSA is now more accessible due to the expiration of patents and relaxation of US export regulations (Kota, C.M. and Aissi, C., 2022).

3. Transmission Flow

1. **User Dashboard:**

   The sender composes the message in the user interface (UI) of the MessageApp, specifying the recipient and the message content.

2. **Message Encryption by Sender:**

   The app generates a random symmetric encryption key specifically for this message. This key will be used for encrypting the message using the AES encryption algorithm. The message is encrypted using the AES algorithm and the generated symmetric key, resulting in an ciphertext.

3. **Key Exchange:**

   To protect the symmetric encryption key, the app retrieves the recipient's RSA public key from the database. The RSA public key is used for encryption and is known to all users. The sender encrypts the symmetric encryption key (generated in the previous step) using the recipient's RSA public key. This ensures that only the recipient, possessing the corresponding RSA private key, can decrypt the symmetric key.

4. **Receiver Retrieval of Encrypted Message:**

   The receiver logs in to the MessageApp and accesses their dashboard. The app queries the database to retrieve all the encrypted messages that are meant for the receiver.

5. **Decryption by Receiver:**

   For each encrypted message received, the receiver's dashboard extracts the encrypted symmetric key from the database. The app uses the receiver's RSA private key which was stored in the session after login to decrypt the symmetric key. With the decrypted symmetric key, the receiver decrypts the message using the AES algorithm, obtaining the original plaintext message.

6. **Displaying Decrypted Message to Receiver:**

   The decrypted message plaintext is displayed in the user interface of the receiver's dashboard, allowing them to read and understand the content of the message securely.

4. Work Flow

MessageApp is a chat-based web application that helps to connect two users and they can exchange the message. While creating this application we have implemented RSA algorithm.

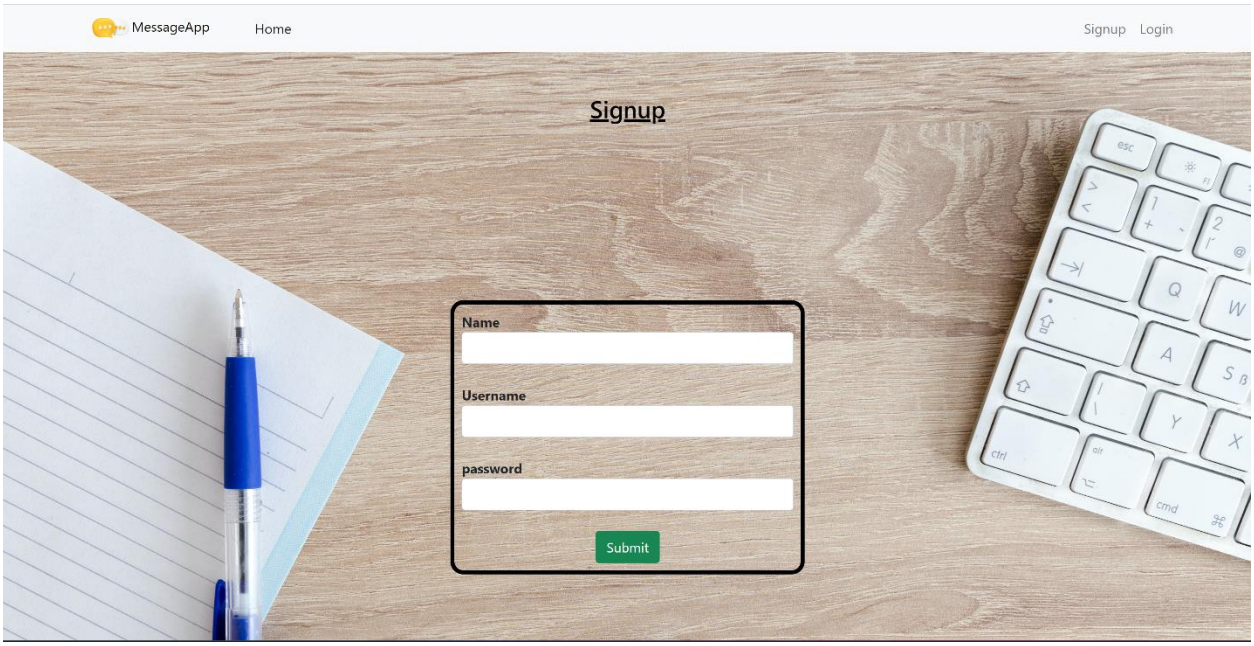

Figure 1. Signup page for users

Firstly, users needs to signup for using this MessageApp providing their details.

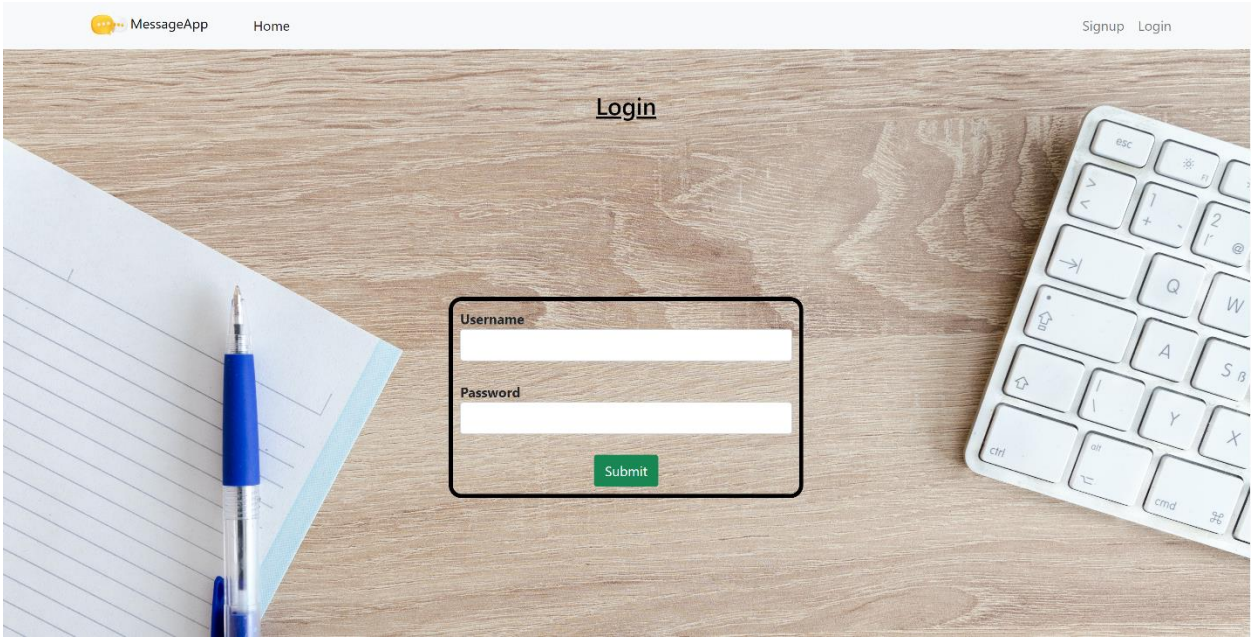

Figure 2. Login page

In this login page, registered user can login to this web app by using their login credentials.

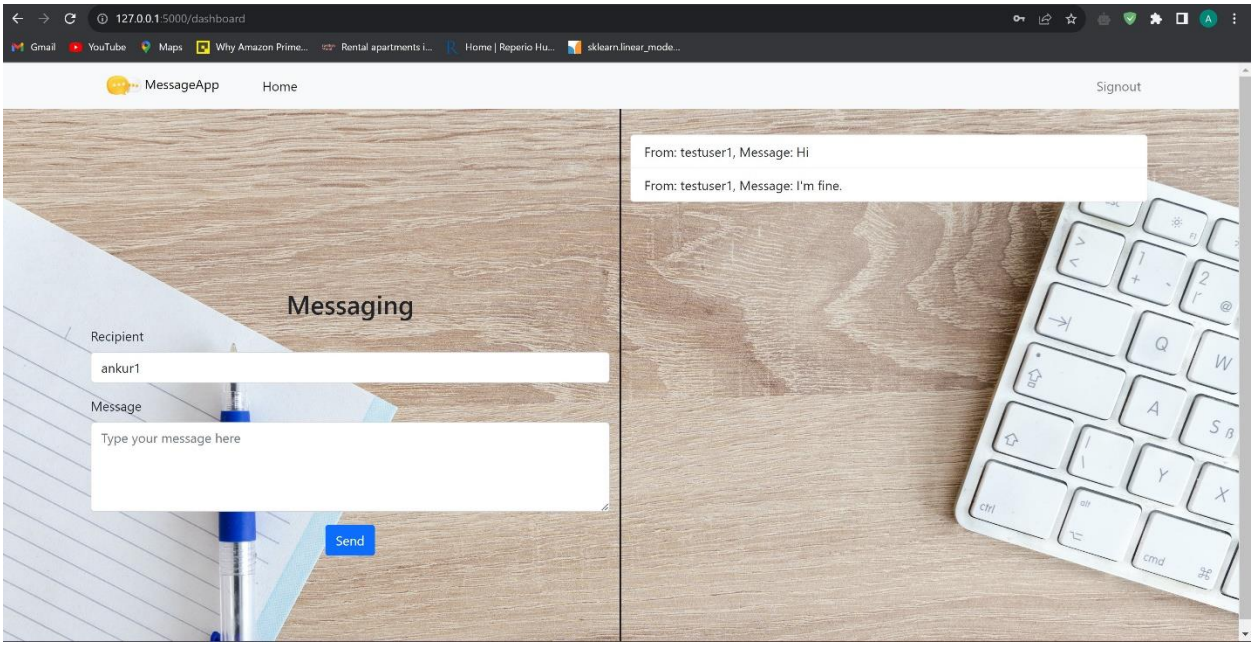

Figure 3. Sending message by sender1

Here, Sender1 will login to the system and then he will select recipient and write message for to the sender two. Whenever he send the msg, after successful message transmission, alert box will popped up.

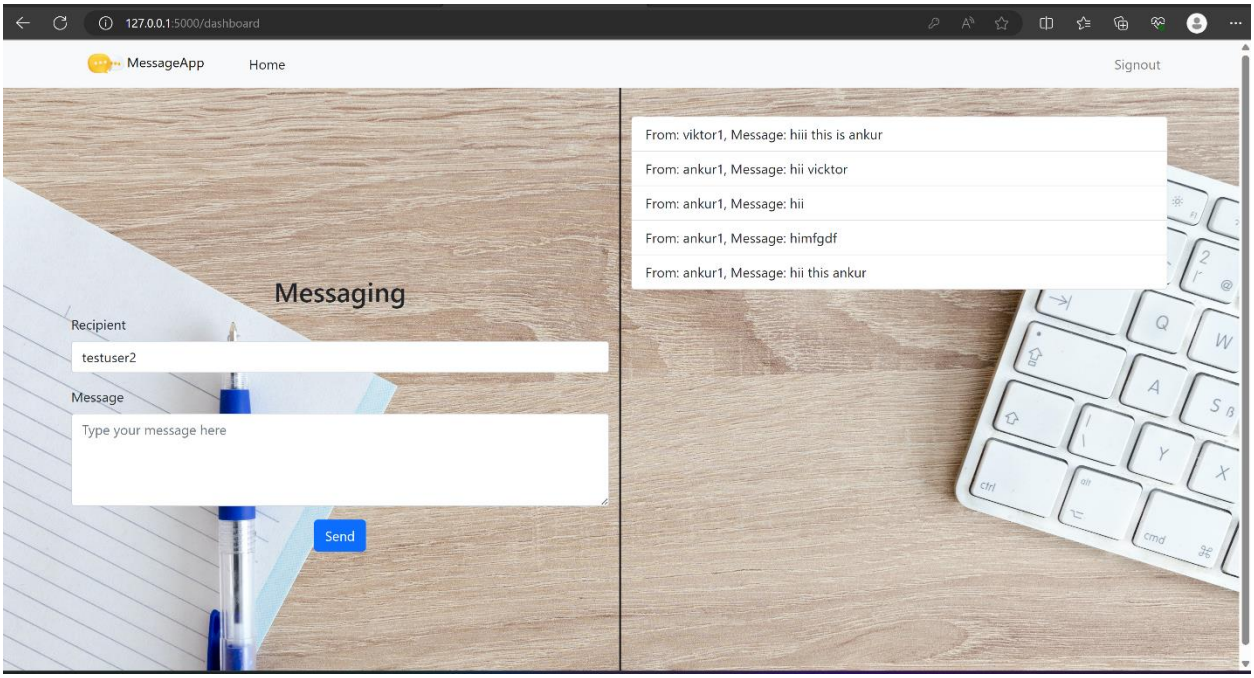

Figure 4. Receiving message by sender2

At the receiving side, sender2 will login to his/her app and in dashboard, messages will show in dashboard.

5. Key Features

5.1 Importing Libraries and Modules

The Flask application starts by importing necessary libraries and modules. These include Flask for building the web application, MySQLdb for database connectivity, and Flask-WTF for CSRF protection.

5.2 Database Configuration and Connection

Database credentials are securely stored in a dictionary named db_config. The code establishes a connection to the MySQL database using the MySQLdb library.

5.3 User Data Functions

The code provides several functions to work with user data in the database. These functions handle user registration, user login, and password hashing for enhanced security.

5.4 Cryptographic Functions

Various cryptographic functions are implemented for secure key exchange and message encryption/decryption. These include generating RSA key pairs for secure communication, generating a random symmetric encryption key for each message, and encrypting and decrypting messages using AES encryption for symmetric encryption.

5.5 Flask App Configuration

The Flask app is configured with a secret key, session type, and an upload folder. CSRF protection is enabled using the Flask-WTF extension, ensuring a higher level of security against cross-site request forgery attacks.

5.6 Routes

These several routes to handle different functionalities of the MessageApp:

1. Homepage ('/'): Renders the index.html template.

2. User Registration ('/signup'): Handles user registration.

3. User Login ('/login'): Manages user login.

4. User Dashboard ('/dashboard'): Displays other users and allows sending messages.

5. Sending Messages ('/send'): Implements the secure key exchange mechanism using RSA and AES encryption for message transmission.

6. Receiving Messages ('/receive'): Retrieves and decrypts messages for the logged-in user.

7. User Logout ('/logout'): Handles user logout.

5.7 Secure Key Exchange Mechanism

The messaging app implements a robust secure key exchange mechanism in the /send route, ensuring the confidentiality and authenticity of messages:

1. The sender generates a random symmetric encryption key (key) for each message using the generate_random_key() function.

2. The message (message) is symmetrically encrypted using AES with the symmetric encryption key (key) using the encrypt_message() function.

3. The sender retrieves the recipient's RSA public key from the database and encrypts the symmetric encryption key (key) with it. This ensures that only the recipient can decrypt the key.

4. The encrypted message (encrypted_data) and the encrypted symmetric key (encrypted_symmetric_key) are securely stored in the database for delivery.

5.8 Message Decryption

In the /receive route, the messaging app retrieves and decrypts messages for the logged-in user:

1. The route retrieves all messages addressed to the current user from the database.

2. For each message, the route retrieves the encrypted message (ciphertext) and the encrypted symmetric key (encrypted_symmetric_key).

3. The user's RSA private key is loaded from the session and used to decrypt the encrypted_symmetric_key using RSA decryption.

4. The symmetric key (symmetric_key) obtained from RSA decryption is then used to decrypt the ciphertext using AES decryption.

5. The decrypted message is added to a list along with the sender's username.

6. The list of decrypted messages is then returned as a JSON response to the client.

6. Security Measures

6.1 Password Hashing

The code employs the bcrypt library to ensure robust security by hashing user passwords before storing them in the database. Hashing passwords is a crucial practice that adds an extra layer of protection, as it prevents the original passwords from being stored in plain text. By doing so, the

code makes it significantly more challenging for potential attackers to retrieve the actual passwords, even if they manage to gain access to the database. This approach enhances the overall security of the application and safeguards the sensitive user information effectively.

6.2 CSRF Protection

The application incorporates Cross-Site Request Forgery (CSRF) protection through the use of the Flask-WTF extension. This security measure plays a crucial role in preventing malicious attackers from executing unauthorized requests on behalf of authenticated users. By implementing CSRF protection, the application ensures that only legitimate and authorized requests are allowed, effectively mitigating the risk of CSRF attacks and bolstering the overall security of the system (Czeskis, A., Moshchuk, A., Kohno, T. and Wang, H.J., 2013).

6.3 Database Input Validation

The code implements input validation and sanitization as part of its defense against SQL injection attacks. This security measure is achieved by employing parameterized queries with placeholders whenever interacting with the database. By using this approach, the code ensures that user inputs are thoroughly validated and sanitized before being used in database queries. This effectively prevents potential SQL injection attacks, making the application more resilient against malicious attempts to manipulate the database through user inputs.

6.4 User Input Validation

The code includes validation and sanitization of user input to prevent various types of attacks, such as XSS (Cross-Site Scripting) attacks. For example, the code checks that the username contains only alphanumeric characters and underscores to prevent injection of harmful characters.

6.5 SSL/TLS Encryption

The application uses SSL/TLS encryption to secure communication between the server and clients. When configuring a web server for HTTPS, we need both the cert.pem and key.pem files to enable SSL/TLS encryption and provide a secure connection between the server and clients.

7 Conclusion

The secure messaging app built with Flask incorporates strong cryptographic operations, database security, and CSRF protection to ensure a safe and private communication platform. By implementing a secure key exchange mechanism, the app guarantees that messages are transmitted confidentially and only accessible to their intended recipients.

8 References

1. Ermoshina, K., Musiani, F. and Halpin, H., 2016. End-to-end encrypted messaging protocols: An overview. In *Internet Science: Third International Conference, INSCI 2016, Florence, Italy, September 12-14, 2016, Proceedings 3* (pp. 244- 254). Springer International Publishing.

2. Salomon, D., 2003. *Data privacy and security: encryption and information hiding*. Springer Science & Business Media.

3. ProPrivacy.com. (n.d.). *AES Encryption | Everything you need to know about AES*. [online] Available at: https://proprivacy.com/guides/aes-encryption#is-aes-encryption-the-best-type-of-encryption [Accessed 2 Aug. 2023].

4. Kota, C.M. and Aissi, C., 2022, March. Implementation of the RSA algorithm and its cryptanalysis. In *2002 GSW*.

5. Czeskis, A., Moshchuk, A., Kohno, T. and Wang, H.J., 2013, May. Lightweight server support for browser-based CSRF protection. In *Proceedings of the 22nd international conference on World Wide Web* (pp. 273-284).