A Report On

Cryptographic Secure Messaging System Using AES

Submitted by

Name	Entry Number
ARINDAM SAL	2024JCS2041
AYAN SHIL	2024JCS2048

1. Introduction & Goal of the Assignment

This assignment implements a secure messaging application that ensures confidentiality, integrity, message freshness, and username privacy using Advanced Encryption Standard (AES) encryption in CBC mode. We use password-derived keys to secure client-server communication, enabling safe message storage and retrieval by encrypting usernames and messages.

The following figure demonstrates the flow of messages and key generation. The sender uses a password to derive a key using PBKDF2, which, combined with a random Initialization Vector (IV), encrypts both the username and message content before sending it to the receiver.

Representation:

- **Sender (A) ➔ Receiver (B)**
    - **Key Derivation (PBKDF2):** Converts password to key
    - **AES-CBC Mode:** Encryp10ts messages with confidentiality
    - **IV (Initialization Vector):** Unique per message for replay protection

2. Implementation Features

This section highlights the implementation features, with code snippets demonstrating each feature's fulfilment of assignment requirements.

2.1 Password-Derived Encryption Key (Using PBKDF2)

- The password is converted into a secure key using PBKDF2. This function generates a key using a salt to add entropy.

```
def derive_key(password, salt):
    return PBKDF2(password, salt, dkLen=AES_KEY_SIZE)
```

2.2 AES Encryption with Random IV for Confidentiality

- AES encryption is performed in CBC mode, with a unique 16-byte IV for each message, ensuring each encrypted output remains distinct.

```
iv = get_random_bytes(AES_BLOCK_SIZE)
cipher = AES.new(key, AES.MODE_CBC, iv)
```

2.3 Integrity and Replay Prevention with Random IVs

- A unique, random IV is generated for each message, preventing replay attacks and ensuring identical messages have unique encrypted outputs.

```
iv = get_random_bytes(AES_BLOCK_SIZE)
```

2.4 Socket-Based Client-Server Communication

- The client and server communicate over a secure socket, transferring encrypted messages and usernames.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

    s.connect((SERVER_IP, SERVER_PORT))

    s.sendall(str(data_packet).encode())
```

2.5 Database Storage(SQLite3) for Encrypted Messages

- Each message is securely stored in an SQLite database, along with encrypted usernames, IVs, and salts.
- Additional admin features:
    - Admin can delete individual messages or clear the entire database.
    - Admin can decrypt messages if the correct password is provided.

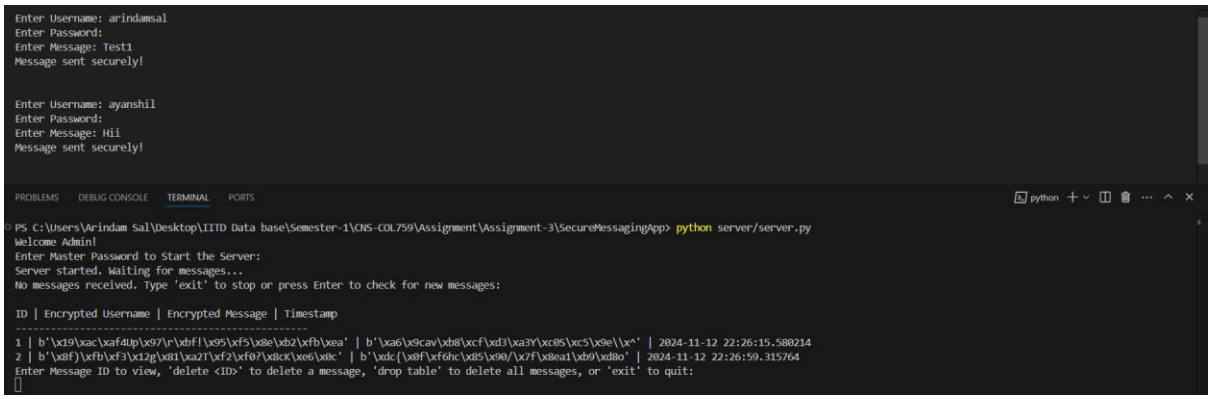
```
def store_message(user_id, message, iv_user, iv_message, salt_user, salt_message):

    conn = sqlite3.connect("messages.db")

    # ... SQL insertion of encrypted message data ...
```

3. Security Objectives

This section describes how the implementation achieves the assignment's security objectives.

3.1 Confidentiality

- The encryption ensures that only intended recipients with the correct password can decrypt and read messages.
- **Screenshot:** Encrypted message in the database.

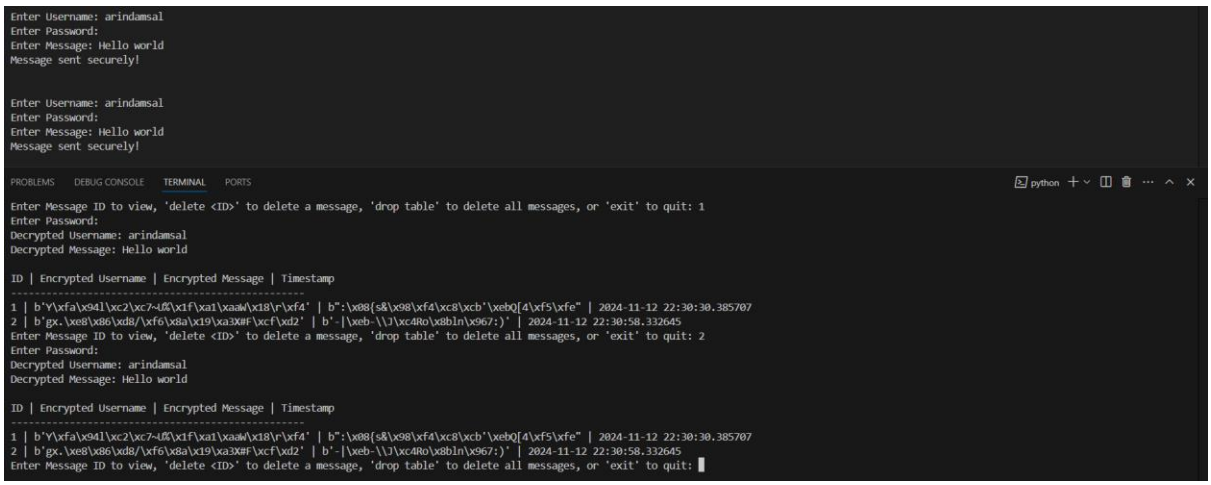


3.2 Integrity

- The AES-CBC mode ensures that tampered messages fail to decrypt correctly, preserving the integrity of each message.

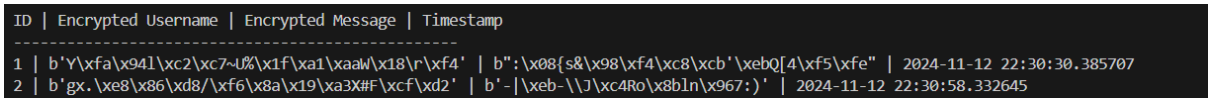
3.3 Message Freshness and Replay Attack Prevention

- Each message has a unique IV, ensuring identical messages appear differently in encrypted form, preventing replay attacks.



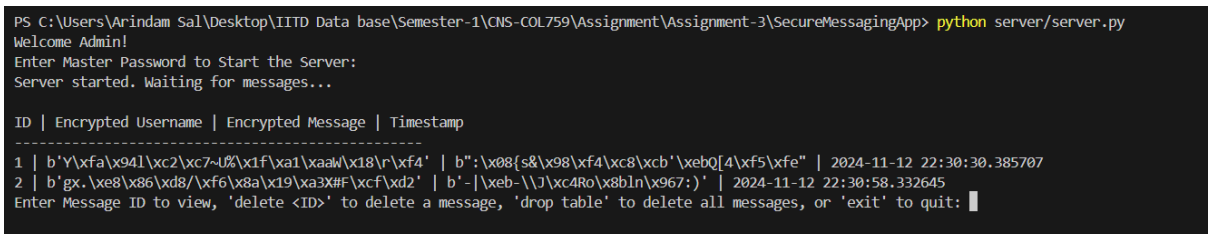
3.4 Username Privacy

- Usernames are encrypted separately and stored with a fixed length (16 bytes max), ensuring privacy.
- **Screenshot:** Encrypted usernames stored in the database



3.5 Extra Layer of Security

- A master password protects sensitive operations on the server, like message deletion or database reset, adding an extra layer of security.
- **Screenshot:** Prompt for master password before performing admin actions



4. Environment Setup & Running the Application

Details on dependencies and the commands to set up and run the application.

4.1. Dependencies

- **Pythonpackages:** Need to Install required packages(pycryptodome==3.14.1, sqlite3 etc) at both client and server sides:

```
pip install -r client/requirements.txt

pip install -r server/requirements.txt
```

4.2. Commands

- **Running the Server:** python server/serevr.py
- **Running the client:** python client/client.py

4.3. Master password: admin007

5. Testing Strategy

Each test case below verifies different aspects of the application.

5.1. Basic Message Encryption and Decryption

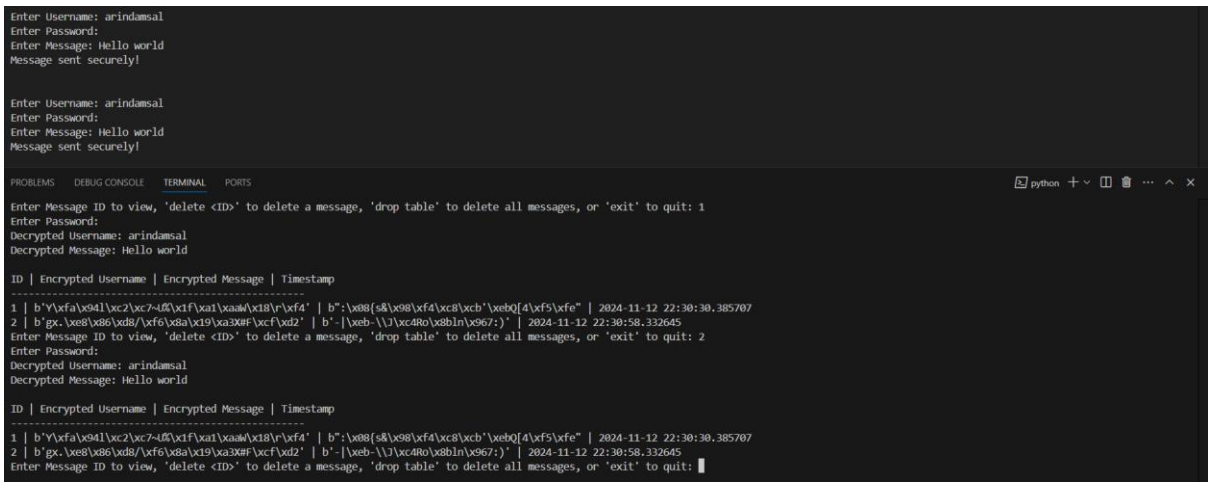
- Ensures the message encrypts and decrypts successfully.
- **Screenshot:** Encrypted and decrypted messages match.

5.2. Maximum Username Length (16 Bytes)

- Verifies that usernames with exactly 16 bytes are encrypted and stored correctly.
- **Screenshot:** Showing storage of 16-byte username.

5.3. Replay Attack Prevention

- Ensures unique IV for each identical message, preventing replays.
- **Screenshot:** Encrypted output differs for identical messages.



5.4. Invalid Password for Decryption

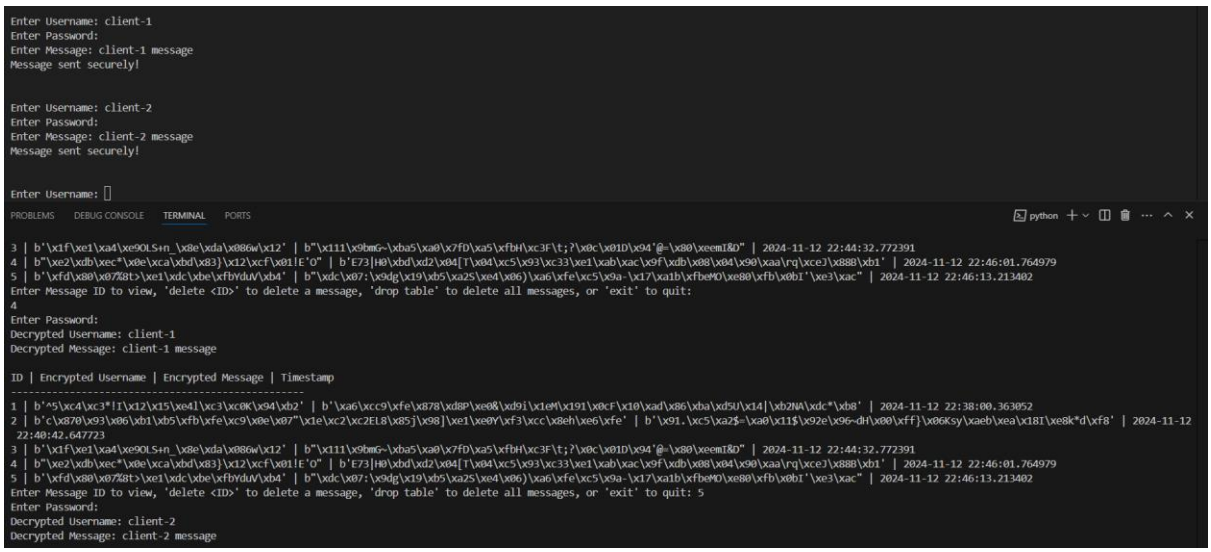
- Checks that decryption fails with an incorrect password.
- **Screenshot:** Failed decryption message.

5.5. Special Characters in Username and Message

- Confirms that non-ASCII characters encrypt and decrypt correctly.
- **Screenshot:** Display of special characters after decryption.

5.6. Multiple Clients with the Same Password

- Checks that multiple clients can send messages securely.
- **Screenshot:** Shows encrypted messages from different clients.



5.7. Empty Message

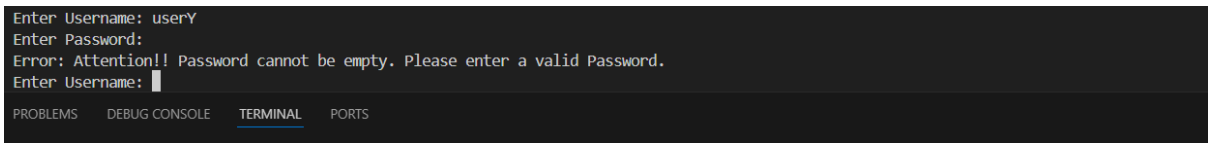
- Ensures that empty messages are encrypted and stored without errors.
- **Screenshot:** Show encrypted empty message.

5.8. Username Exceeds 16 Bytes

- Verifies that usernames longer than 16 bytes are rejected.
- **Screenshot:** Error message for username length limit.

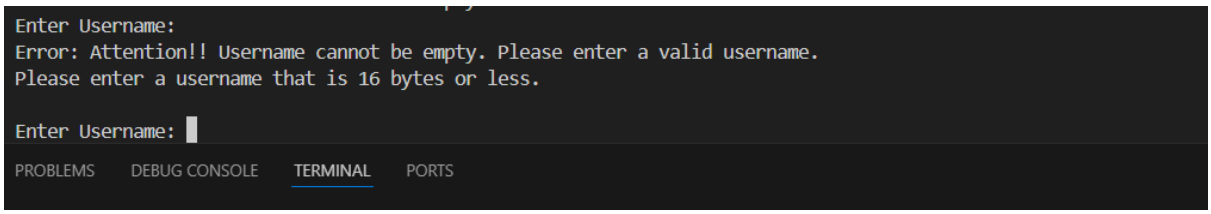
5.9. Empty Password

- Checks that an empty password triggers an error.
- **Screenshot:** Error message for empty password.



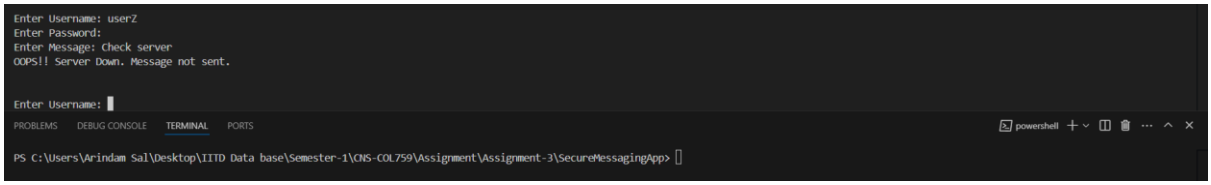
6. Empty Username

- Verifies that empty usernames are rejected.
- **Screenshot:** Error message for empty username.



7. Server is Down

- Client fails to send message
- **Screenshot:** Handling when the server is down.



6. Future Scope & Conclusion

6.1. Conclusion: This assignment effectively demonstrates secure message encryption, with additional features to enhance confidentiality, integrity, and replay protection in a simulated client-server setup. By ensuring encrypted storage of messages and adding a master password for server operations, this application lays a foundation for real-world secure communication systems.

6.2. Future Scope: Future improvements could include:

- Implementing a more robust key exchange protocol.
- Enhancing password security with multi-factor authentication.
- Adding message forwarding or deletion timestamps for better traceability.