COL759:Cryptography & Computer Security

Assignment Prepared By:

Abhinav Singh : 2021CS50746

Pushpraj : 2021CS50596

Guide :

Prof. Ashok K Bhateja

Dept. of Computer Science and Engineering

Indian Institute of Technology, Delhi

New Delhi , India

Contents

1 Project Approach and Code Flow

This project implements a secure client-server messaging application that ensures confidentiality and integrity by using encryption and secure key derivation. The system employs AES encryption in CBC mode, HMAC for integrity, and a unique salt for each session to prevent replay attacks. The code is organized into four main files:

1. client.py
2. crypto_utils.py
3. database_utils.py
4. server.py

Each file is responsible for a specific aspect of the application's functionality, and together, they create a secure end-to-end encrypted messaging environment.

1.1 Client file

The client.py file is responsible for establishing a connection with the server, deriving a session key, encrypting the message, and sending it to the server.

- **Server Connection**:The client initiates a connection to the server using a socket on a specified IP and port .
- **Key Derivation**: Upon connection, the server sends a unique salt to the client. The client then derives a cryptographic key from the shared password and the received salt using the PBKDF2 key derivation function in crypto_utils.py.
- **Encryption and Message Transmission**:
  - The client accepts user input for both the username and the message.

  - These inputs are padded, encrypted, and combined with a randomly generated initialization vector (IV).
  - The encrypted payload, including the IV, is then sent to the server.

- **Receiving Server Response**: After sending a message, the client waits for a response from the server to confirm receipt, providing an interactive experience.

1.2 Crypto_utils file

The crypto_utils.py file provides utility functions for cryptographic operations, including key derivation, padding, encryption, and decryption.

- **Key Derivation**:The derive_key function uses PBKDF2HMAC with SHA-256 to create a secure key from the shared password and server-provided salt.
- **Padding and Unpadding**:The pad and unpad functions ensure that the plaintext length aligns with AES block size requirements.
- **AES Encryption and Decryption:**:
  - aes_encrypt encrypts plaintext with AES in CBC mode, using the provided IV and derived key.
  - aes_decrypt decrypts ciphertext with AES in CBC mode and removes padding.
- **Message Assembly**: final_message assembles the encrypted message by encrypting both the username and message separately, adding an IV, and returning a concatenated result. break_message extracts the IV, encrypted username, and encrypted message from the received payload, making it easier for the server to decrypt each component.

1.3 Server file

The server.py file handles the server-side operations, including accepting client connections, securely processing incoming messages, and logging them for future use.

- Server Setup: The server initializes the network socket, binds it to the specified address and port, and starts listening for incoming client connections.
- Handling Client Connections:For each new client connection, the server initiates a unique encrypted session. It starts by generating a random salt value, which is sent to the client to establish a session-specific encryption key.
- Message Reception and Decryption:The server receives encrypted messages from the client. Each message is parsed to separate the initialization vector (IV), the encrypted username, and the encrypted message content. Using cryptographic utilities, the server decrypts both the username and message components to retrieve the original plaintext information.
- Database Storage and Response:

  After decrypting the message, the server logs the encrypted data in a database to ensure secure and persistent storage. Each message is recorded with the corresponding encrypted username, message, IV, and a timestamp. The server then sends a confirmation response back to the client, indicating successful message receipt and processing.

1.4 Database_utils file

The database_utils.py file is dedicated to database management, ensuring reliable and secure storage of encrypted messages for future verification or retrieval.

- Database Setup: The file includes functionality to initialize the database by creating a table specifically designed to store encrypted messages. The table holds fields for encrypted usernames, messages, IVs, and timestamps, and it is created if it does not already exist.
- Message Storage: This file manages the insertion of each encrypted message entry into the database. Each stored record includes the encrypted username, message content, IV, and a timestamp to maintain accurate logging.

1.5 Code Flow Summarized

1. **Client Initialization**: The client connects to the server and receives a session-specific salt. It derives an encryption key and prompts the user for a username and message.
2. **Message Encryption**: Using the derived key and a generated IV, the client encrypts the username and message, assembles them into a single encrypted payload, and sends it to the server.
3. **Server Reception and Decryption**: The server receives the encrypted payload, extracts the IV, and decrypts the username and message. It logs the message in the database for future reference.
4. **Confirmation**: The server responds to the client, confirming message receipt and providing feedback to the client.

2 Important Points to Note

- The application developed fulfills security objectives like Confidentiality, Integrity, Message Freshness and Replay Attack Prevention and Username Privacy.
- The client and server share a pre-defined password to generate a unique encryption key for each session. This ensures that only the in-

tended recipient (the server) can decrypt the messages. By implementing E2E encryption with session-specific keys, the system maintains confidentiality and prevents unauthorized access to message contents, even if they are intercepted.

- AES encryption is used in Cipher Block Chaining (CBC) mode, which ensures message confidentiality and prevent patterns from being recognizable.
- The use of a unique IV for each message helps prevent replay attacks by ensuring that identical messages encrypt to different ciphertexts.
- The design incorporates HMAC(Hash-Based Message Authentication Code) to detect tampering by validating message authenticity and ensures integrity.
- Details about number of bytes which we pad can be found in padded byte.
- Whenever user sends a message , server acknowledges that. So this way integrity is maintained in both directions.
- Design also handles the case when 'salt' is tampered.