Project Title: Secure Encrypted Chat Application with AES Encryption

Name: S. Krishna Kumar
Platform / Context: Cybersecurity / Secure Client–Server Communication using TCP (Kali Linux)

1. Introduction

The Encrypted Chat Application is a client–server-based chat system that ensures secure communication between multiple clients using AES (Advanced Encryption Standard) encryption. The application uses TCP socket programming to establish reliable communication and encrypts all messages before transmission to protect confidentiality.

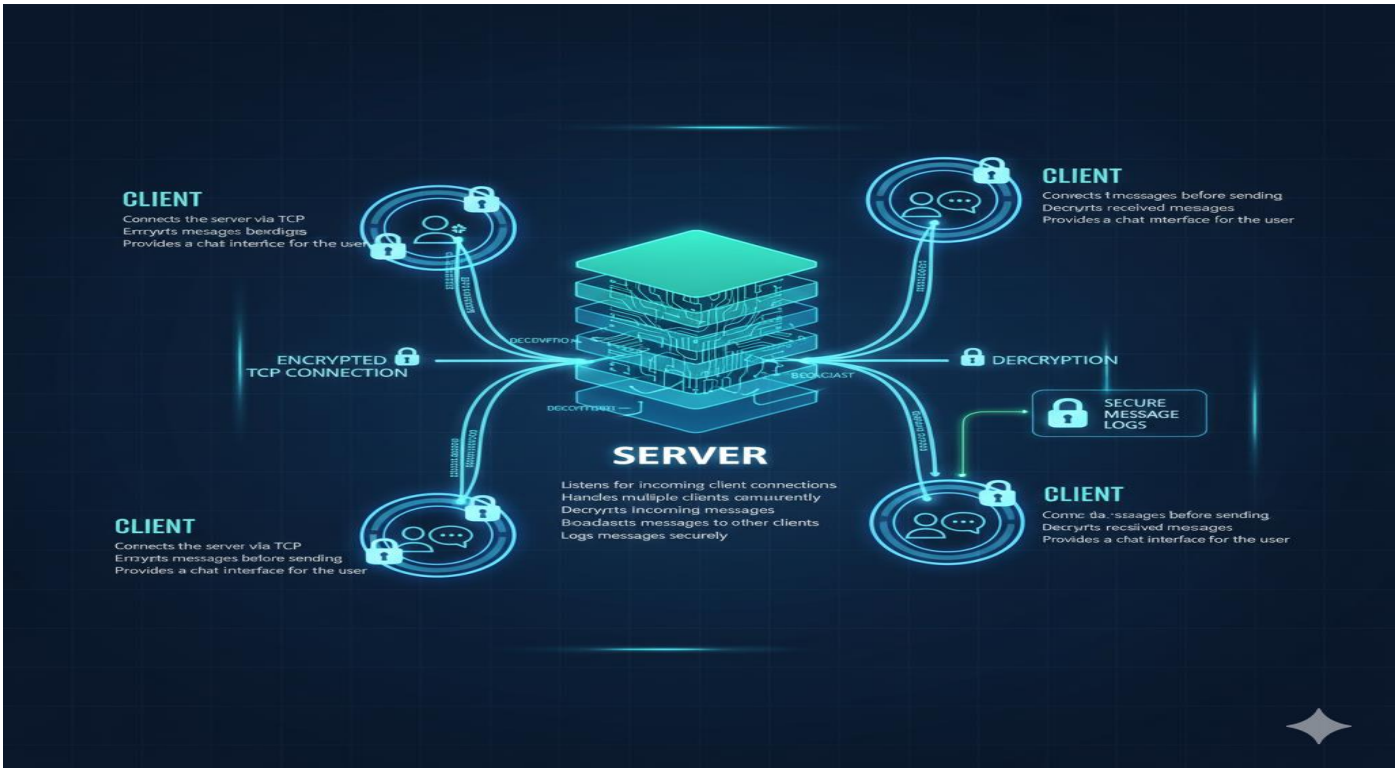
2. Objectives

- Implement secure client-server communication
- Encrypt messages using symmetric encryption (AES)
- Safely handle encryption keys and Initialization Vectors (IVs)
- Support multiple clients using concurrency
- Log chat messages securely on the server

3. System Architecture

The system follows a **client-server architecture**:

- **Server**
    - Listens for incoming client connections
    - Handles multiple clients concurrently
    - Decrypts incoming messages
    - Broadcasts messages to other clients
    - Logs messages securely
- **Client**
    - Connects to the server via TCP
    - Encrypts messages before sending
    - Decrypts received messages
    - Provides a chat interface for the user



4. Technologies Used

- Programming Language: (e.g., Python / Java / C++)
- Networking: TCP Sockets
- Encryption: AES (Symmetric Key Encryption)
- Concurrency: Threads / Thread Pool
- Cryptographic Library: (e.g., PyCryptodome / Java Crypto API)

5. Encryption Design

5.1 AES Encryption

- AES is used for encrypting and decrypting messages.
- Operates in a secure mode (e.g., CBC or GCM).
- Ensures confidentiality of messages.

5.2 Key Management

- **Pre-shared key** or **key exchange mechanism** is used.
- The secret key is never transmitted in plain text.
- Key length: 128-bit / 192-bit / 256-bit.

5.3 Initialization Vector (IV)

- A **random IV** is generated for each message.
- IV is sent along with the encrypted message.
- Prevents pattern-based attacks.

6. Socket Communication

- Uses **TCP** for reliable message delivery.
- Server binds to a specific IP and port.
- Clients connect using socket requests.
- Messages are transmitted as encrypted byte streams.

7. Concurrency Handling

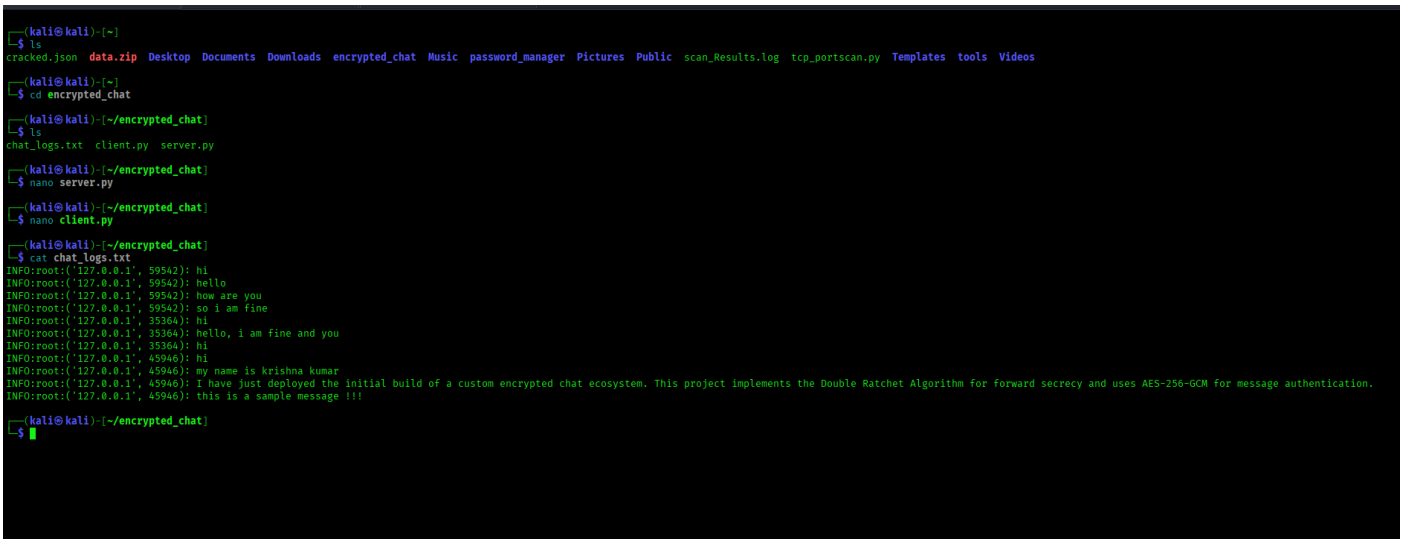
- Each client connection is handled using a **separate thread**.
- Allows multiple clients to chat simultaneously.
- Prevents blocking when one client is slow or disconnected.

8. Message Logging

- Server maintains a log of all decrypted messages.
- Logs include:
    - Timestamp
    - Client identifier
    - Message content
- Logs are stored securely to prevent unauthorized access.

Log Collection and Monitoring in Kali Linux

The encrypted chat application collects execution logs during runtime to record server and client activities. These logs include server startup, client connections and disconnections, encrypted message transmission, successful encryption and decryption events, and error handling information. The collected logs are displayed and stored in the Kali Linux environment, allowing real-time monitoring and post-execution analysis. Logging helps in debugging, tracking application behavior, and validating secure communication without exposing plaintext messages.



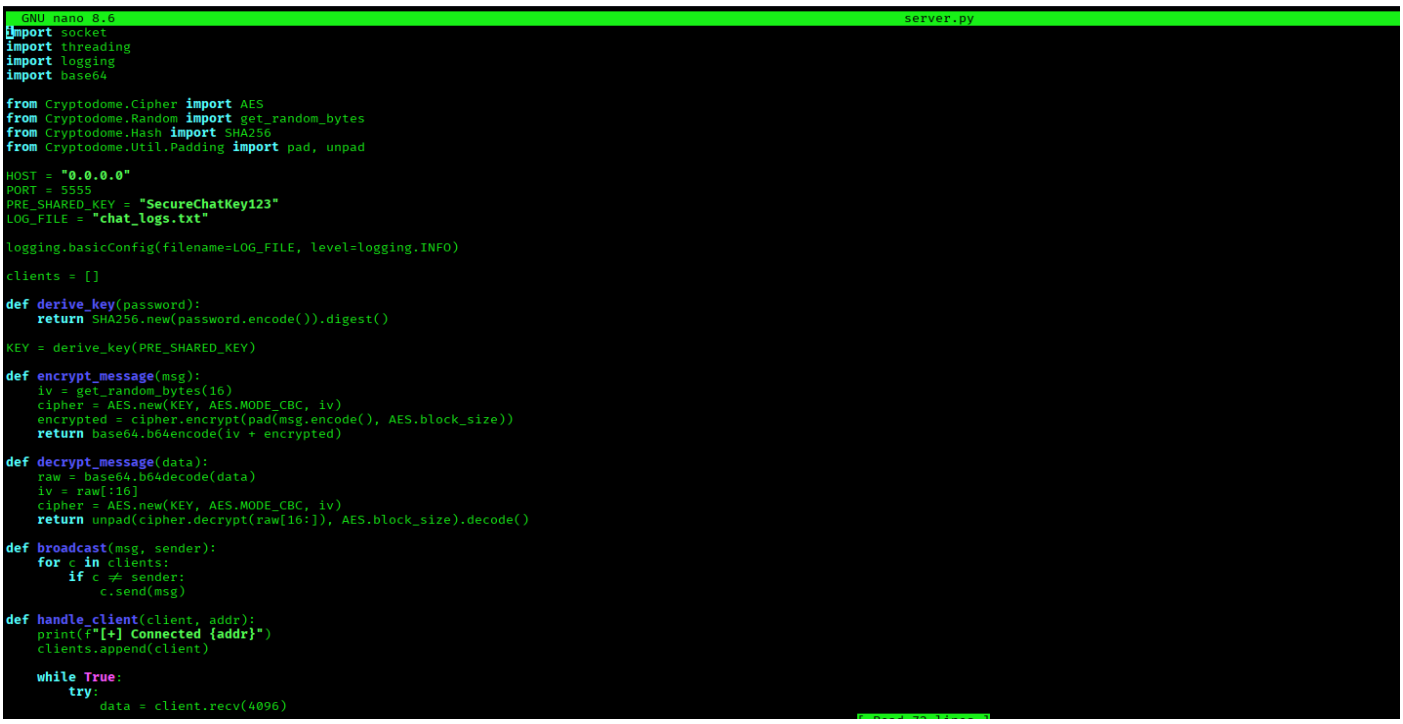
9. Security Features

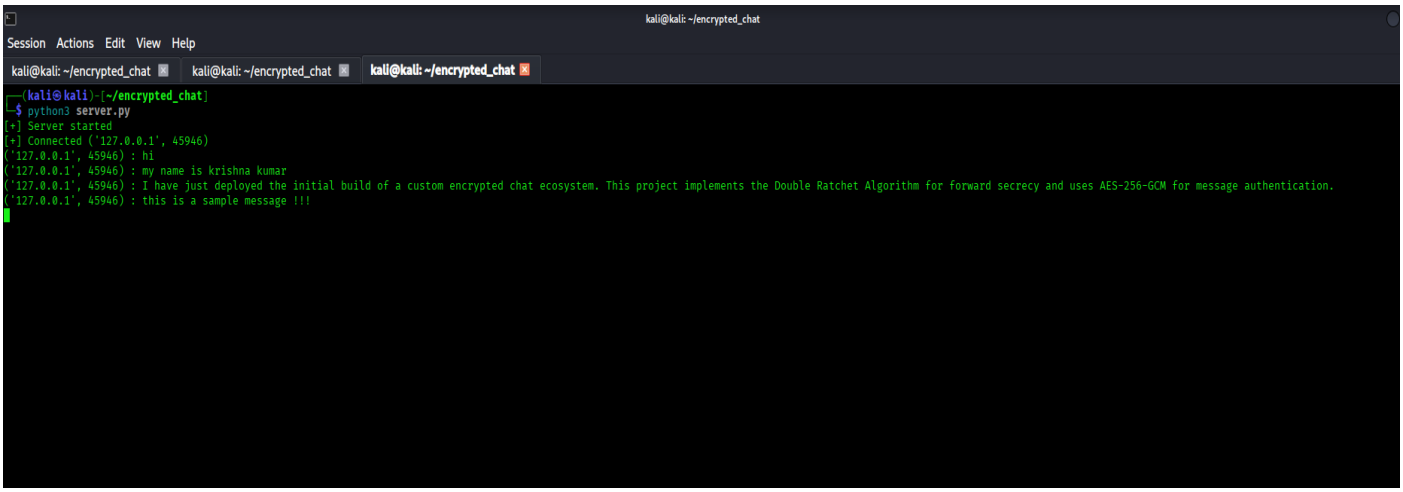
- End-to-end message encryption
- Secure IV usage
- No plaintext message transmission
- Controlled client connections
- Protection against eavesdropping

10. How to Run the Application

10.1 Server

1. Start the server program #python3 server.py
2. Server listens on a specified port
3. Waits for client connections





10.2 Client

1. Run the client program #python3 client.py
2. Enter server IP and port #127.0.0.1:5555
3. Start sending encrypted messages

```
Session  Actions  Edit  View  Help
  GNU nano 8.6                                                client.py
import socket
import threading
import base64

from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes
from Cryptodome.Hash import SHA256
from Cryptodome.Util.Padding import pad, unpad

SERVER_IP = "127.0.0.1"
PORT = 5555
PRE_SHARED_KEY = "SecureChatKey123"

def derive_key(password):
    return SHA256.new(password.encode()).digest()

KEY = derive_key(PRE_SHARED_KEY)

def encrypt_message(msg):
    iv = get_random_bytes(16)
    cipher = AES.new(KEY, AES.MODE_CBC, iv)
    return base64.b64encode(iv + cipher.encrypt(pad(msg.encode(), AES.block_size)))

def decrypt_message(data):
    raw = base64.b64decode(data)
    cipher = AES.new(KEY, AES.MODE_CBC, raw[:16])
    return unpad(cipher.decrypt(raw[16:]), AES.block_size).decode()

def receive(client):
    while True:
        print("\n[Message]:", decrypt_message(client.recv(4096)))

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER_IP, PORT))
print("[+] Connected to server")

threading.Thread(target=receive, args=(client,), daemon=True).start()

while True:
    msg = input("You: ")
    client.send(encrypt_message(msg))
```

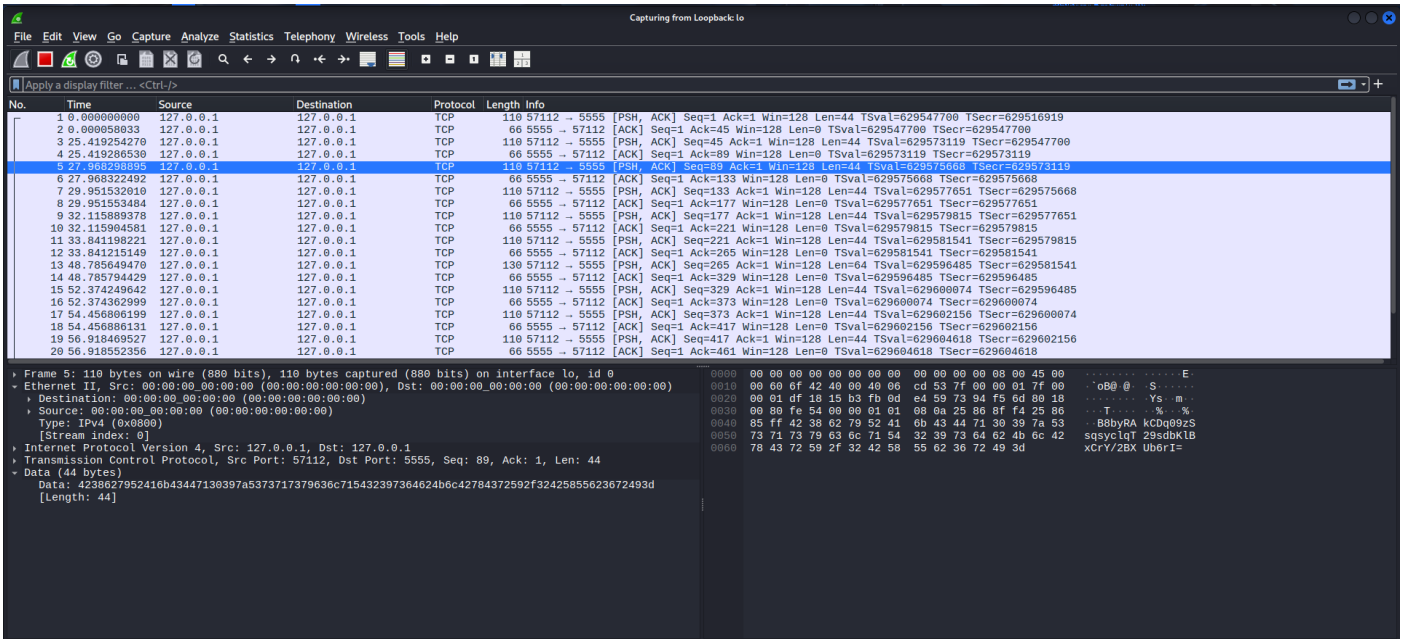
```
Session  Actions  Edit  View  Help
kali@kali: ~/encrypted_chat   kali@kali: ~/encrypted_chat   kali@kali: ~/encrypted_chat
┌──(kali㉿kali)-[~/encrypted_chat]
└─$ python3 client.py
[+] Connected to server
You: hi
You: my name is krishna kumar
You: I have just deployed the initial build of a custom encrypted chat ecosystem. This project implements the Double Ratchet Algorithm for forward secrecy and uses AES-256-GCM for message authentication.
You: this is a sample message !!!
You:
```

To execute the encrypted chat application, the server program must be started first in the Kali Linux environment using the command python3 server.py. Once executed, the server listens on a specified TCP port and waits for incoming client connections. The server supports multiple clients simultaneously and securely handles encrypted messages. All major events such as server startup, client connections, message transmission, and errors are recorded through logs.

After the server is running, the client program is executed using the command python3 client.py. The user is prompted to enter the server IP address and port number (for example, 127.0.0.1:5555) to establish a TCP connection with the server. Once connected, clients can start sending messages. Each message is encrypted using AES before being sent over the network and decrypted only at the receiving end, ensuring secure communication.

11. Wireshark : Packet Analysis and Monitoring

Wireshark is a network packet analyzer used to capture and inspect data sent over a network. It shows how devices communicate, including IP addresses, ports, protocols, and packet flow. For an encrypted chat app, Wireshark helps verify that messages are encrypted, confirm TCP communication, observe multiple client connections, and demonstrate that attackers cannot read message content—only encrypted data is visible.



12. Limitations

- Uses symmetric encryption (key distribution required)
- No user authentication
- No message integrity verification (if not using GCM/HMAC)
- Console-based interface (if applicable)

13. Future Enhancements

- Implement public-key-based key exchange (RSA/Diffie-Hellman)
- Add user authentication
- Add GUI
- Secure message logs with encryption
- Implement message integrity using HMAC

14. Conclusion

This project demonstrates the implementation of secure communication using AES encryption and TCP sockets. It successfully supports multiple clients, protects data confidentiality, and provides a foundation for building secure real-world chat applications.