📘 PROJECT REPORT
Encrypted Chat Application Using AES Encryption

1. Introduction

The purpose of this project is to develop a secure encrypted chat application using Python. The application ensures that any message sent between clients is protected using AES (Advanced Encryption Standard) encryption.

This project demonstrates the concepts of:

Socket programming

Symmetric key encryption

Secure message transmission

Multi-client communication

Real-time encrypted chatting

This completes the requirement of building a secure communication system as part of the SyntecXHub internship tasks.

2. Objective

The main objectives of this project are:

1.Build a server-client chat application.

2.Encrypt every message using AES encryption before sending it.

3.Support multiple clients communicating securely at the same time.

4.Run the application in a Jupyter Notebook environment with a user-friendly interface.

5.Demonstrate encrypted communication using real-time message transmission.

3. Technologies Used
Programming Language:-Python
Encryption:-AES (AES.MODE_EAX)
Networking:-Python Socket Library
Multi-client Support:-Python Threading
Environment:-Jupyter Notebook

Libraries:-PyCryptodome

4. System Architecture
4.1 Components

Server

Listens for incoming client connections

Receives encrypted messages

Decrypts using AES

Broadcasts re-encrypted messages to all clients

Client

Connects to the server

Encrypts message using AES

Sends encrypted message

Receives and decrypts messages from other clients

4.2 Working Flow

Server starts and waits for clients.

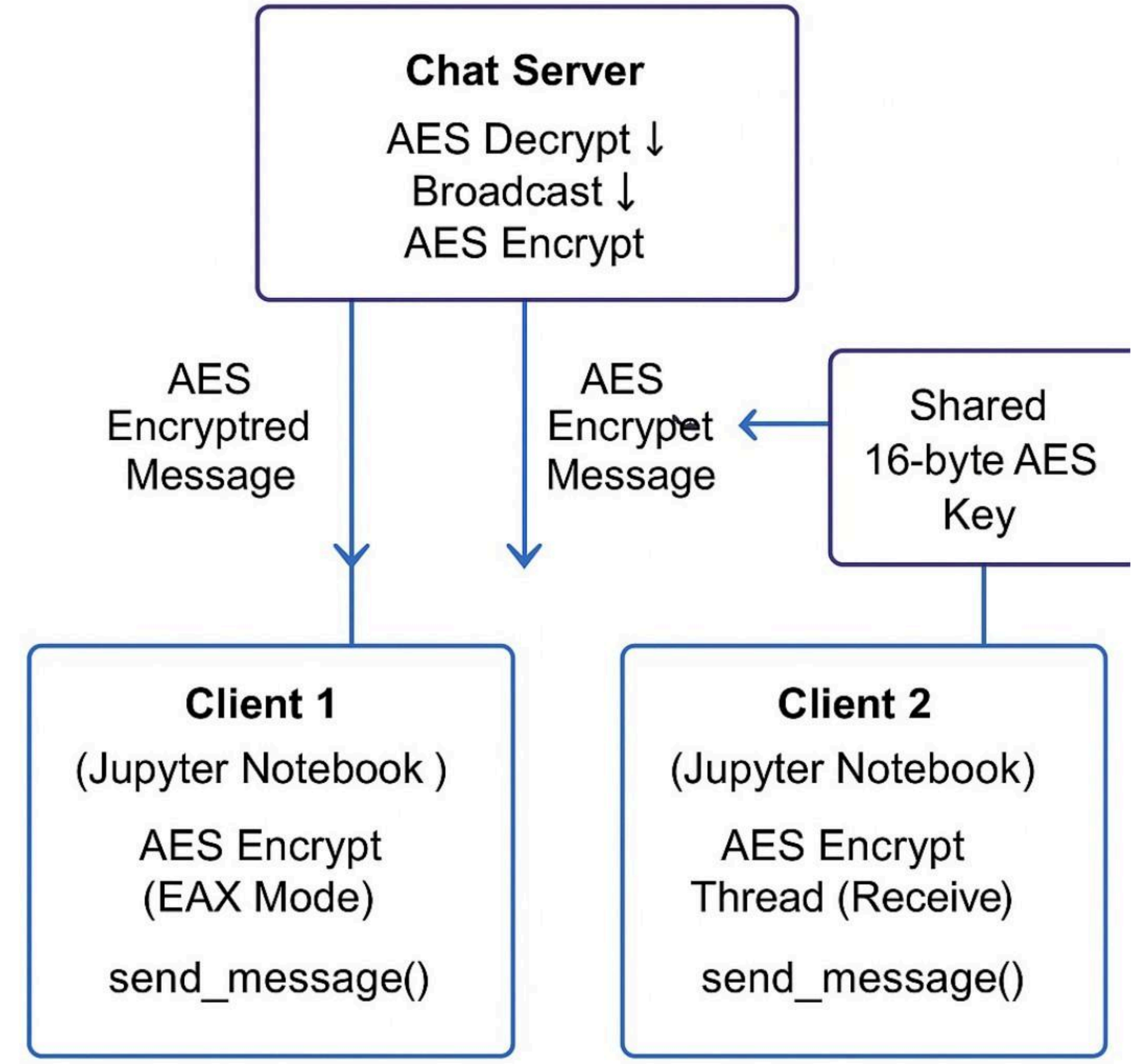
Clients connect to the server.

Each client sends messages using AES encryption.

Server decrypts and reads the message.

Server broadcasts encrypted messages to all other clients.

Clients receive and decrypt messages in real-time.



5. Encryption Details
AES Encryption

AES (Advanced Encryption Standard) is a widely used symmetric encryption algorithm.
This project uses:

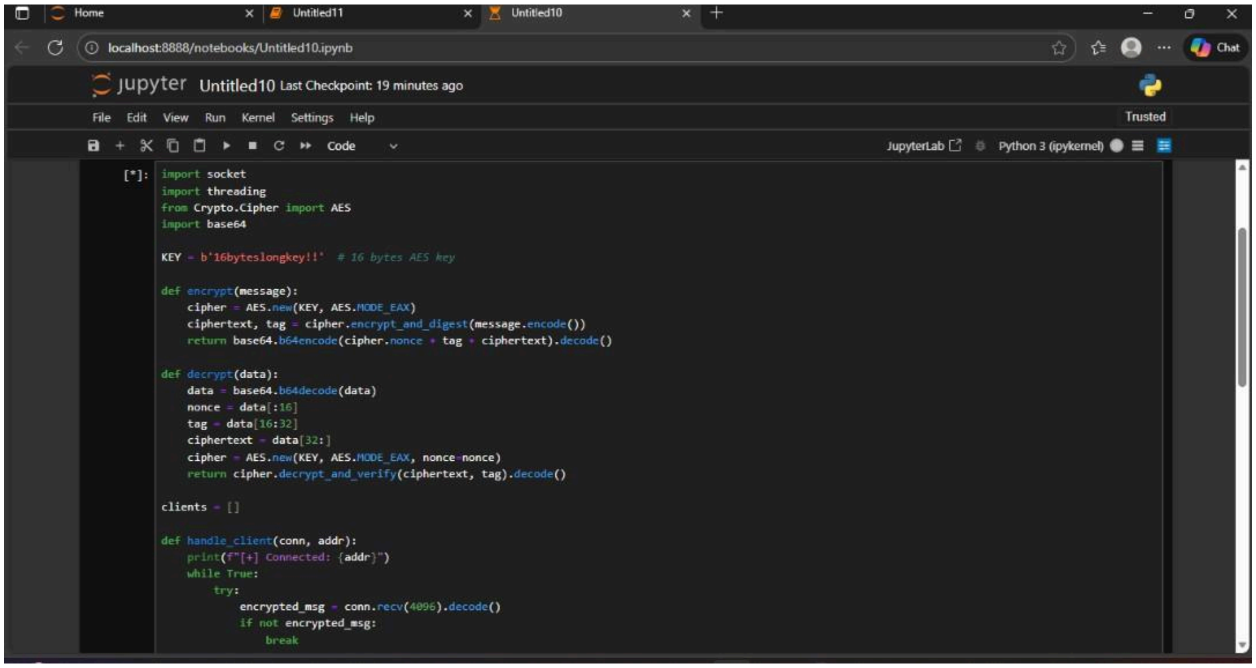
•AES-EAX mode (provides authentication + encryption)

•16-byte key shared between server and clients

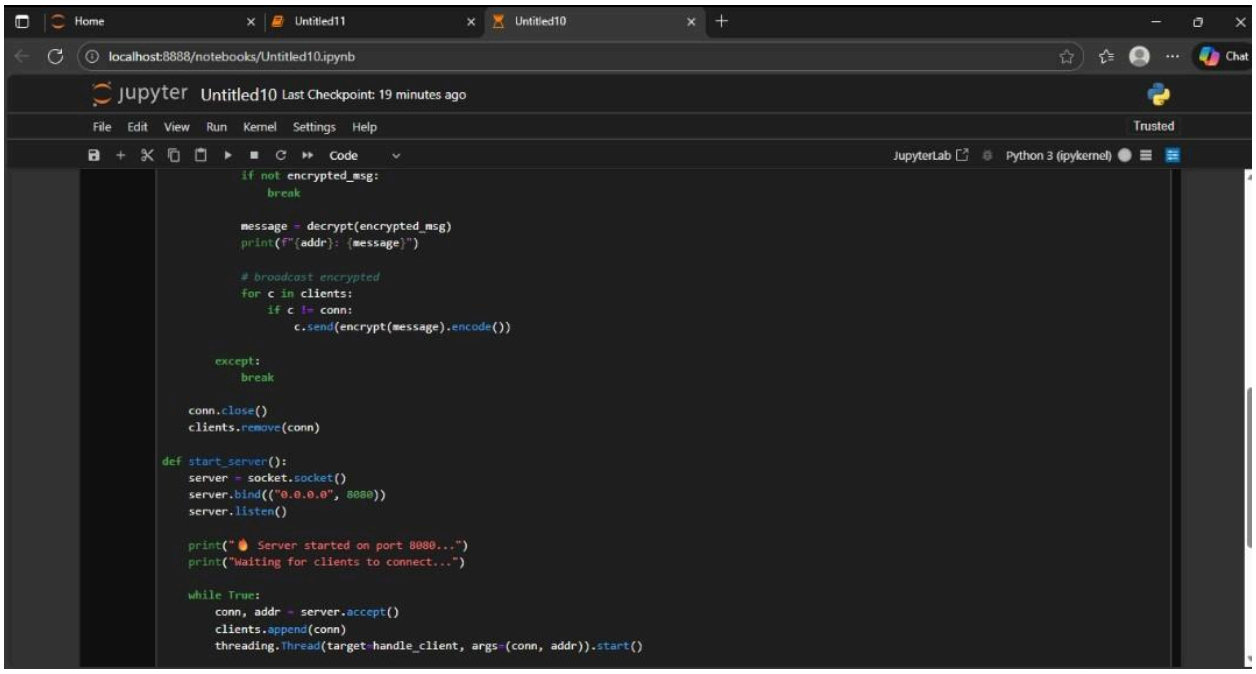
•Secure nonce generated for every message

6. PROGRAM CODE

Server

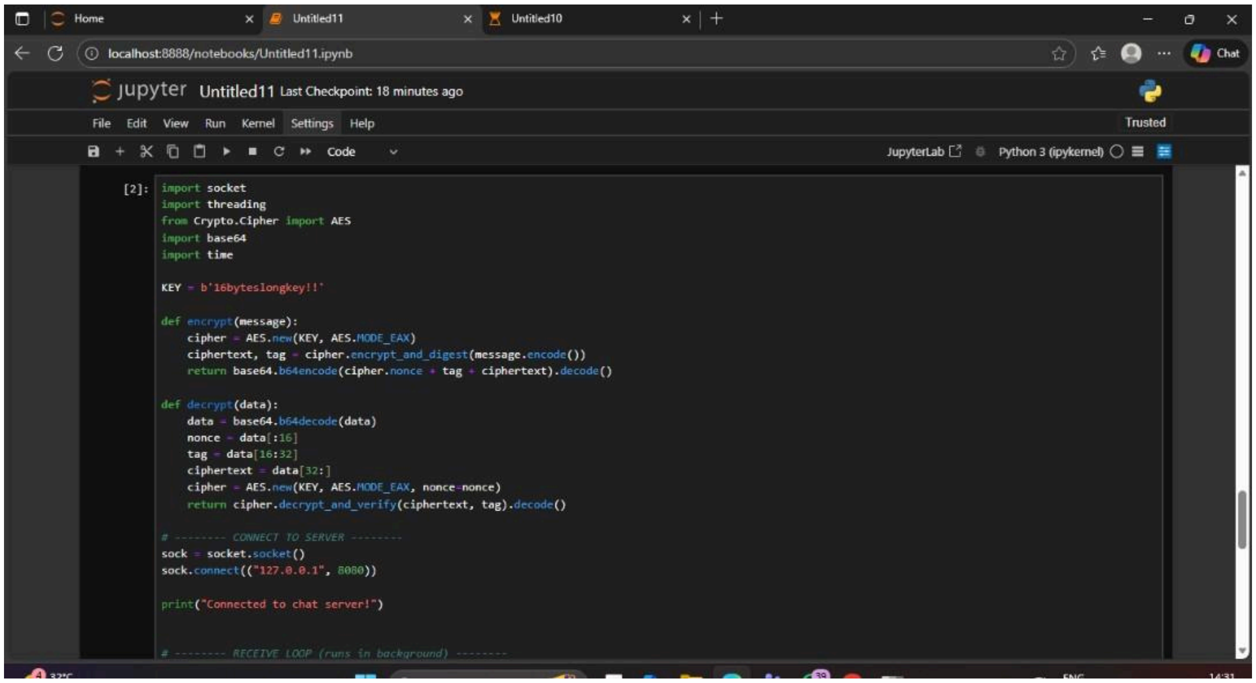


```
import socket
import threading
from Crypto.Cipher import AES
import base64

KEY = b'16byteslongkey!!'  # 16 bytes AES key

def encrypt(message):
    cipher = AES.new(KEY, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(message.encode())
    return base64.b64encode(cipher.nonce + tag + ciphertext).decode()

def decrypt(data):
    data = base64.b64decode(data)
    nonce = data[:16]
    tag = data[16:32]
    ciphertext = data[32:]
    cipher = AES.new(KEY, AES.MODE_EAX, nonce=nonce)
    return cipher.decrypt_and_verify(ciphertext, tag).decode()

clients = []

def handle_client(conn, addr):
    print(f"[+] Connected: {addr}")
    while True:
        try:
            encrypted_msg = conn.recv(4096).decode()
            if not encrypted_msg:
                break
```

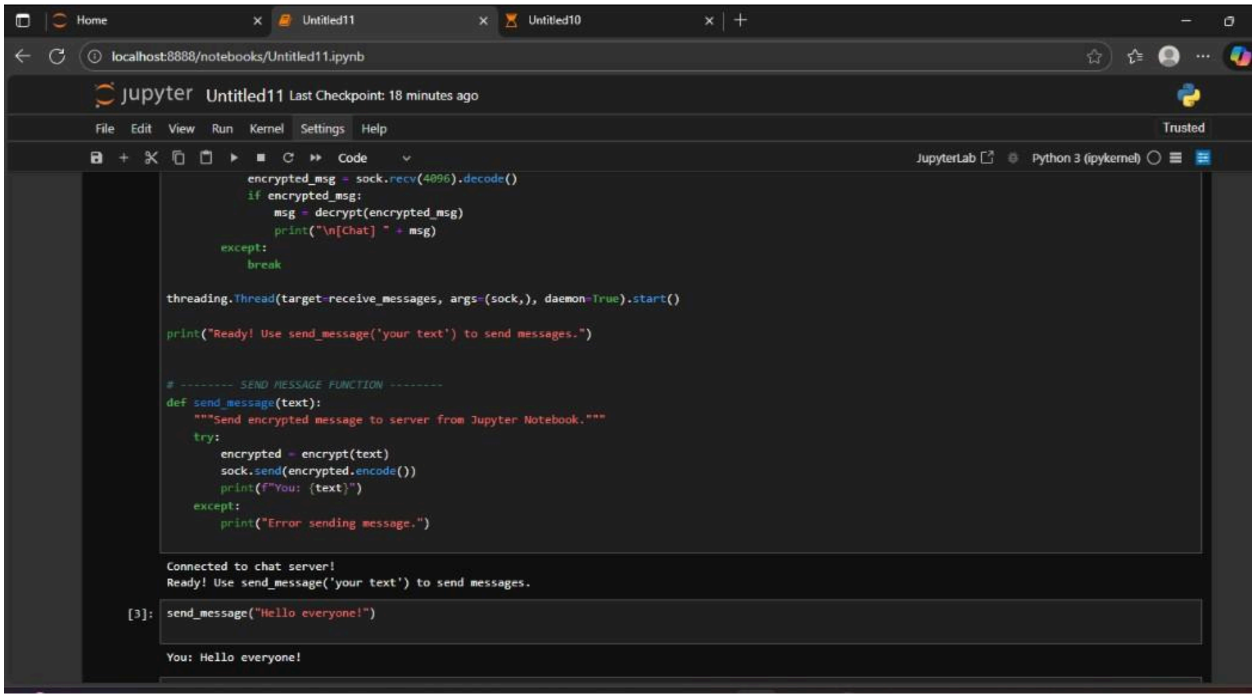


```
            if not encrypted_msg:
                break

            message = decrypt(encrypted_msg)
            print(f"{addr}: {message}")

            # broadcast encrypted
            for c in clients:
                if c != conn:
                    c.send(encrypt(message).encode())

        except:
            break

    conn.close()
    clients.remove(conn)

def start_server():
    server = socket.socket()
    server.bind(("0.0.0.0", 8080))
    server.listen()

    print("🔥 Server started on port 8080...")
    print("Waiting for clients to connect...")

    while True:
        conn, addr = server.accept()
        clients.append(conn)
        threading.Thread(target=handle_client, args=(conn, addr)).start()
```

Client

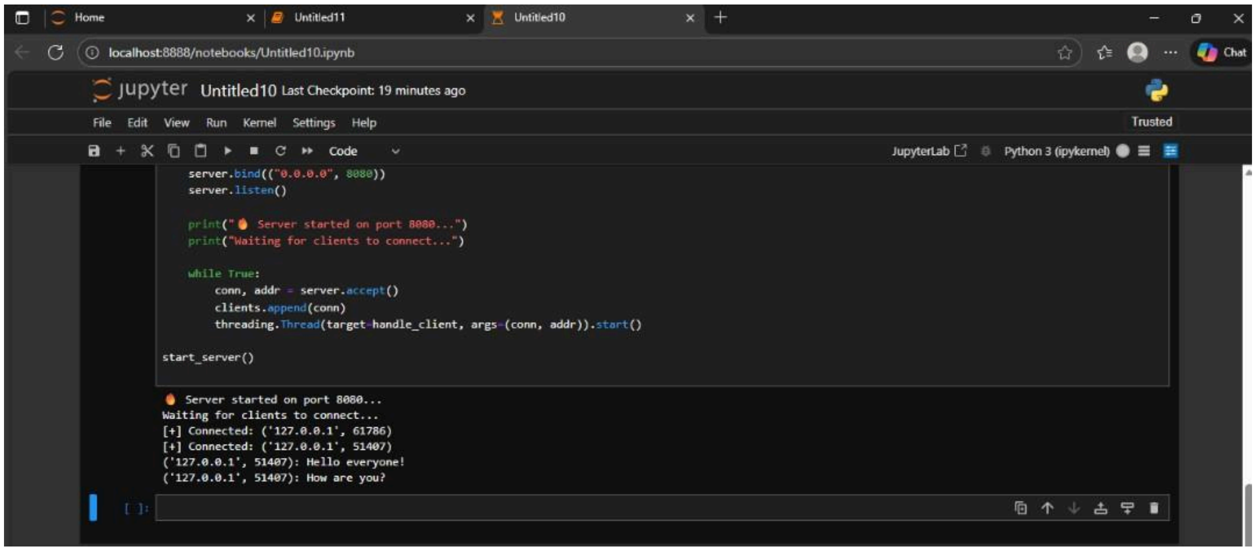


```
[2]: import socket
     import threading
     from Crypto.Cipher import AES
     import base64
     import time

     KEY = b'16byteslongkey!!'

     def encrypt(message):
         cipher = AES.new(KEY, AES.MODE_EAX)
         ciphertext, tag = cipher.encrypt_and_digest(message.encode())
         return base64.b64encode(cipher.nonce + tag + ciphertext).decode()

     def decrypt(data):
         data = base64.b64decode(data)
         nonce = data[:16]
         tag = data[16:32]
         ciphertext = data[32:]
         cipher = AES.new(KEY, AES.MODE_EAX, nonce=nonce)
         return cipher.decrypt_and_verify(ciphertext, tag).decode()

     # -------- CONNECT TO SERVER --------
     sock = socket.socket()
     sock.connect(("127.0.0.1", 8080))

     print("Connected to chat server!")


     # -------- RECEIVE LOOP (runs in background) --------
```



```
                 encrypted_msg = sock.recv(4096).decode()
                 if encrypted_msg:
                     msg = decrypt(encrypted_msg)
                     print("\n[Chat] " + msg)
             except:
                 break

     threading.Thread(target=receive_messages, args=(sock,), daemon=True).start()

     print("Ready! Use send_message('your text') to send messages.")


     # -------- SEND MESSAGE FUNCTION --------
     def send_message(text):
         """Send encrypted message to server from Jupyter Notebook."""
         try:
             encrypted = encrypt(text)
             sock.send(encrypted.encode())
             print(f"You: {text}")
         except:
             print("Error sending message.")

     Connected to chat server!
     Ready! Use send_message('your text') to send messages.

[3]: send_message("Hello everyone!")

     You: Hello everyone!
```

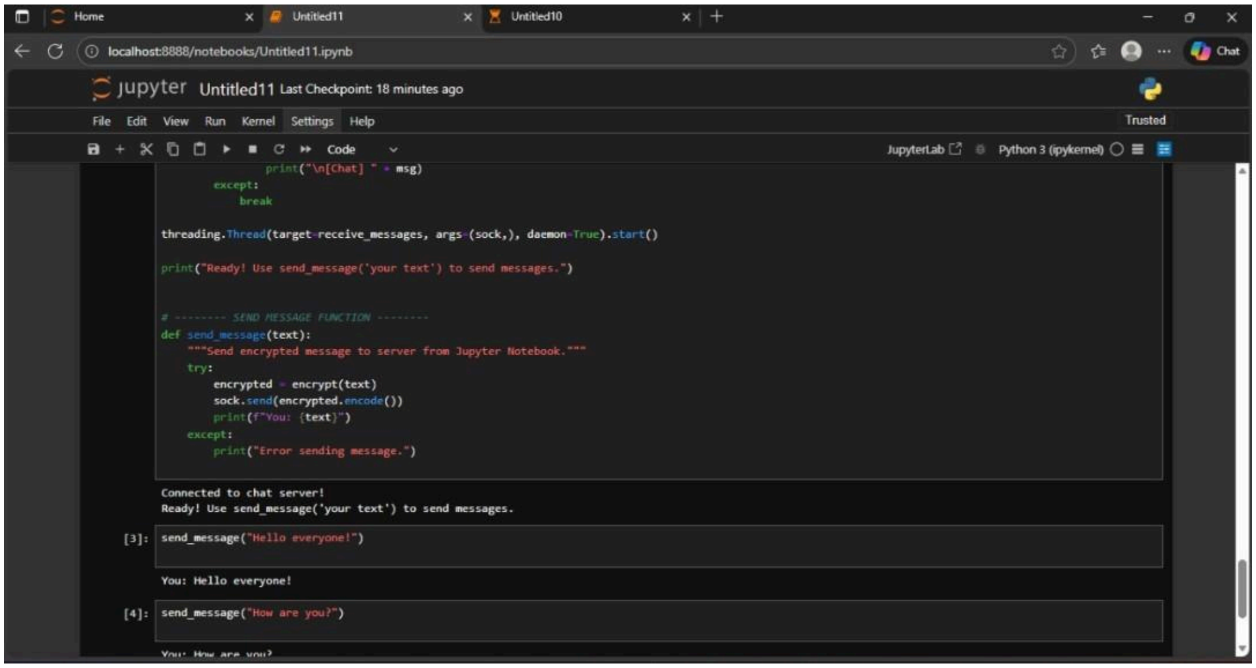
7. OUTPUT
Server



```
    server.bind(("0.0.0.0", 8080))
    server.listen()

    print("🔥 Server started on port 8080...")
    print("Waiting for clients to connect...")

    while True:
        conn, addr = server.accept()
        clients.append(conn)
        threading.Thread(target=handle_client, args=(conn, addr)).start()

start_server()
```

```
🔥 Server started on port 8080...
Waiting for clients to connect...
[+] Connected: ('127.0.0.1', 61786)
[+] Connected: ('127.0.0.1', 51407)
('127.0.0.1', 51407): Hello everyone!
('127.0.0.1', 51407): How are you?
```

Client



```
                print("\n[Chat] " + msg)
        except:
            break

threading.Thread(target=receive_messages, args=(sock,), daemon=True).start()

print("Ready! Use send_message('your text') to send messages.")


# -------- SEND MESSAGE FUNCTION --------
def send_message(text):
    """Send encrypted message to server from Jupyter Notebook."""
    try:
        encrypted = encrypt(text)
        sock.send(encrypted.encode())
        print(f"You: {text}")
    except:
        print("Error sending message.")
```

```
Connected to chat server!
Ready! Use send_message('your text') to send messages.
```

```
[3]: send_message("Hello everyone!")
```

```
You: Hello everyone!
```

```
[4]: send_message("How are you?")
```

```
You: How are you?
```

8. Security Considerations

•AES encryption ensures confidentiality of messages.

•EAX mode provides message authentication and prevents tampering.

•The AES key must remain confidential.

•Replay attacks prevented by unique nonce per message.

•For production systems, each client should have a unique key or use RSA-based key exchange.

9. Conclusion

This project successfully demonstrates how to create a secure, encrypted chat application using Python.
The system uses AES encryption to protect communication between clients and supports multiple clients simultaneously.

•The project helped in learning:

•Real-time secure communication

•Use of AES encryption

•Socket programming

•Thread handling

Implementing encryption inside Jupyter Notebook

This fulfills the requirement of the SyntexHub encrypted chat application project.

10. Future Enhancements

Possible improvements include:

✔ AES-256 encryption
✔ RSA-based key exchange
✔ GUI-based chat interface
✔ Encrypted message logs
✔ End-to-end encryption (Signal Protocol)
✔ Android app version.