Ex. No.3-Secured Client Server Communications

Name: **SUJAN CHOWDARY JUJJAVARAPU (RA2412033010001)**

Implements both TCP and UDP servers and clients that communicate using the Caesar Cipher for message encryption. The server listens for incoming messages, decrypts them, processes the content, and sends back an encrypted response.

Key Components

1. **Caesar Cipher Functions**
   - `encrypt(text, shift)`: Encrypts the input text by shifting characters.
   - `decrypt(text, shift)`: Decrypts the input text using the inverse of the encryption shift.
2. **TCP Server**
   - Listens on localhost at port 12345.
   - Accepts connections, receives encrypted messages, decrypts them, and sends back an encrypted response.
3. **TCP Client**
   - Connects to the TCP server, sends an encrypted message, and receives the server's response.
4. **UDP Server**
   - Listens on localhost at port 12346.
   - Receives encrypted messages, decrypts them, and sends back an encrypted response to the sender.
5. **UDP Client**
   - Sends an encrypted message to the UDP server and receives the response.

Usage

- Users choose the desired component (TCP/UDP server/client) to run.
- Each component handles message encryption and decryption automatically.

Code:

```
import socket
import time

# Caesar Cipher utility functions
def encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            ascii_offset = 65 if char.isupper() else 97
            result += chr((ord(char) - ascii_offset + shift) % 26 + ascii_offset)
        else:
            result += char
    return result

def decrypt(text, shift):
    return encrypt(text, -shift)

# TCP Server
def tcp_server():
    host = '127.0.0.1'
    port = 12345
    shift = 3

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)

    print(f"TCP Server listening on {host}:{port}")

    while True:
        conn, addr = server_socket.accept()
        print(f"Connected by {addr}")

        encrypted_data = conn.recv(1024).decode()
        decrypted_data = decrypt(encrypted_data, shift)
```

```
        print(f"Received (encrypted): {encrypted_data}")
        print(f"Received (decrypted): {decrypted_data}")

        response = f"Server received: {decrypted_data}"
        encrypted_response = encrypt(response, shift)
        conn.send(encrypted_response.encode())

        conn.close()

# TCP Client
def tcp_client():
    host = '127.0.0.1'
    port = 12345
    shift = 3

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))

    message = "Hello, encrypted TCP server!"
    encrypted_message = encrypt(message, shift)
    print(f"Sending (original): {message}")
    print(f"Sending (encrypted): {encrypted_message}")
    client_socket.send(encrypted_message.encode())

    encrypted_response = client_socket.recv(1024).decode()
    decrypted_response = decrypt(encrypted_response, shift)
    print(f"Received from server (encrypted): {encrypted_response}")
    print(f"Received from server (decrypted): {decrypted_response}")

    client_socket.close()

# UDP Server
def udp_server():
    host = '127.0.0.1'
    port = 12346
    shift = 3

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((host, port))

    print(f"UDP Server listening on {host}:{port}")
```

```
    while True:
        encrypted_data, addr = server_socket.recvfrom(4096)
        encrypted_data = encrypted_data.decode()
        decrypted_data = decrypt(encrypted_data, shift)
        print(f"Received from {addr}")
        print(f"Encrypted: {encrypted_data}")
        print(f"Decrypted: {decrypted_data}")

        response = f"Server received: {decrypted_data}"
        encrypted_response = encrypt(response, shift)
        server_socket.sendto(encrypted_response.encode(), addr)

# UDP Client
def udp_client():
    server_address = ('127.0.0.1', 12346)
    shift = 3

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    try:
        message = "Hello, encrypted UDP Server!"
        encrypted_message = encrypt(message, shift)
        print(f"Sending (original): {message}")
        print(f"Sending (encrypted): {encrypted_message}")
        client_socket.sendto(encrypted_message.encode(), server_address)

        print("Waiting for response...")
        encrypted_data, server = client_socket.recvfrom(4096)
        encrypted_data = encrypted_data.decode()
        decrypted_data = decrypt(encrypted_data, shift)
        print(f"Received from {server}")
        print(f"Encrypted: {encrypted_data}")
        print(f"Decrypted: {decrypted_data}")

    finally:
        print("Closing socket")
        client_socket.close()

# Main function to run the desired component
def main():
    print("Choose a component to run:")
    print("1. TCP Server")
```

```
    print("2. TCP Client")
    print("3. UDP Server")
    print("4. UDP Client")
    choice = input("Enter your choice (1-4): ")

    if choice == '1':
        tcp_server()
    elif choice == '2':
        tcp_client()
    elif choice == '3':
        udp_server()
    elif choice == '4':
        udp_client()
    else:
        print("Invalid choice. Please run the script again and select a number between
1 and 4.")

if __name__ == "__main__":
    main()
```

Output:

TCP Server

```
Last login: Sun Oct  6 09:38:41 on console
(base) jsujanchowdary@Js-MacBook-Pro ~ % ls
Applications            Movies                  learning.dSYM
Cloud-Lab-Exps          Music                   pyhton_practice.py
Desktop                 Pictures                test.ipynb
Documents               Public                  welcome-to-docker
Downloads               learning
Library                 learning.cpp
(base) jsujanchowdary@Js-MacBook-Pro ~ % cd Downloads
(base) jsujanchowdary@Js-MacBook-Pro Downloads % ls
Docker-Container RA2412033010001 .pdf   main.py
Ollama.app                              screen-capture (1).webm
Resume.pdf                              screen-capture (2).webm
Srmseal.png                             screen-capture.webm
Visual Studio Code.app
(base) jsujanchowdary@Js-MacBook-Pro Downloads % python3 main.py
Choose a component to run:
1. TCP Server
2. TCP Client
3. UDP Server
4. UDP Client
Enter your choice (1-4): 1
TCP Server listening on 127.0.0.1:12345
Connected by ('127.0.0.1', 50214)
Received (encrypted): Khoor, hqfubswhg WFS vhuyhu!
Received (decrypted): Hello, encrypted TCP server!
```

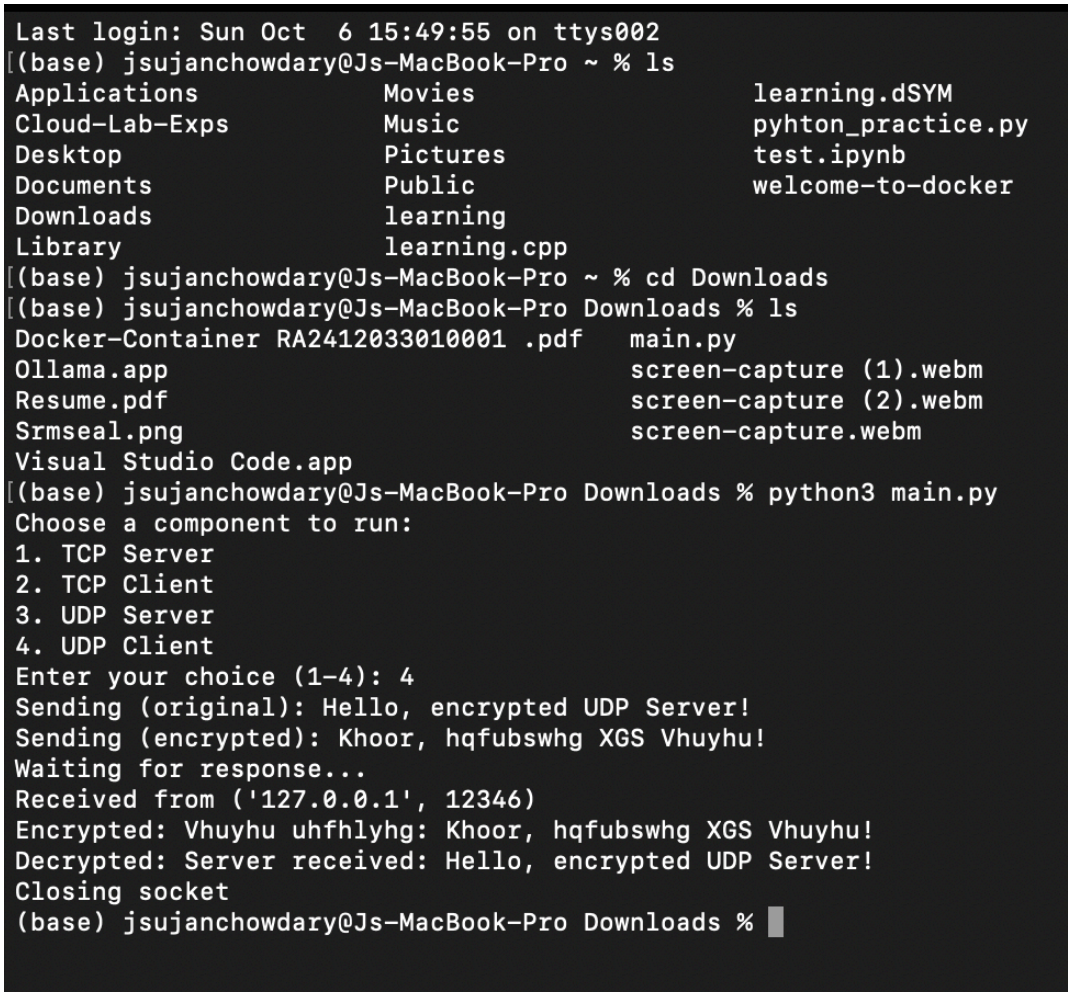
TCP Client

```
Last login: Sun Oct  6 15:49:42 on ttys000
(base) jsujanchowdary@Js-MacBook-Pro ~ % ls
Applications            Movies                  learning.dSYM
Cloud-Lab-Exps          Music                   pyhton_practice.py
Desktop                 Pictures                test.ipynb
Documents               Public                  welcome-to-docker
Downloads               learning
Library                 learning.cpp
(base) jsujanchowdary@Js-MacBook-Pro ~ % cd Downloads
(base) jsujanchowdary@Js-MacBook-Pro Downloads % ls
Docker-Container RA2412033010001 .pdf   main.py
Ollama.app                              screen-capture (1).webm
Resume.pdf                              screen-capture (2).webm
Srmseal.png                             screen-capture.webm
Visual Studio Code.app
(base) jsujanchowdary@Js-MacBook-Pro Downloads % python3 main.py
Choose a component to run:
1. TCP Server
2. TCP Client
3. UDP Server
4. UDP Client
Enter your choice (1-4): 2
Sending (original): Hello, encrypted TCP server!
Sending (encrypted): Khoor, hqfubswhg WFS vhuyhu!
Received from server (encrypted): Vhuyhu uhfhlyhg: Khoor, hqfubswhg WFS vhuyhu!
Received from server (decrypted): Server received: Hello, encrypted TCP server!
(base) jsujanchowdary@Js-MacBook-Pro Downloads %
```

UDP Server

```
Last login: Sun Oct  6 15:49:49 on ttys001
(base) jsujanchowdary@Js-MacBook-Pro ~ % ls
Applications            Movies                  learning.dSYM
Cloud-Lab-Exps          Music                   pyhton_practice.py
Desktop                 Pictures                test.ipynb
Documents               Public                  welcome-to-docker
Downloads               learning
Library                 learning.cpp
(base) jsujanchowdary@Js-MacBook-Pro ~ % cd Downloads
(base) jsujanchowdary@Js-MacBook-Pro Downloads % ls
Docker-Container RA2412033010001 .pdf   main.py
Ollama.app                              screen-capture (1).webm
Resume.pdf                              screen-capture (2).webm
Srmseal.png                             screen-capture.webm
Visual Studio Code.app
(base) jsujanchowdary@Js-MacBook-Pro Downloads % python3 main.py
Choose a component to run:
1. TCP Server
2. TCP Client
3. UDP Server
4. UDP Client
Enter your choice (1-4): 3
UDP Server listening on 127.0.0.1:12346
Received from ('127.0.0.1', 54467)
Encrypted: Khoor, hqfubswhg XGS Vhuyhu!
Decrypted: Hello, encrypted UDP Server!
```

UDP Client



Observations

1. **Encryption and Security:**
    - The use of the Caesar Cipher provides a basic level of encryption, which enhances data privacy during transmission. However, it is not secure against modern cryptographic attacks, highlighting the need for more robust encryption methods in real applications.
2. **Socket Communication:**
    - The implementation successfully demonstrates the differences between TCP and UDP protocols. TCP ensures reliable communication with connection-oriented features, while UDP offers faster, connectionless communication but lacks guaranteed delivery.
3. **Performance:**
    - The TCP server/client handles connections effectively, allowing for one-on-one communication. In contrast, the UDP server/client can handle

multiple messages rapidly without establishing a persistent connection, demonstrating its suitability for time-sensitive applications.

4. **Error Handling:**
    - The implementation could benefit from enhanced error handling. For instance, handling potential exceptions during socket communication would improve robustness.
5. **User Interaction:**
    - The command-line interface is user-friendly, allowing easy selection of components. However, providing additional feedback during execution could enhance user experience.
6. **Extensibility:**
    - The code structure is modular, making it relatively easy to extend functionality, such as integrating more sophisticated encryption algorithms or adding features like multi-client handling in the TCP server.
7. **Limitations:**
    - The Caesar Cipher is simple and can be easily broken. For sensitive applications, stronger encryption algorithms (e.g., AES) should be considered.
    - Both server types only handle a single client at a time; implementing multi-threading or asynchronous I/O could improve scalability.