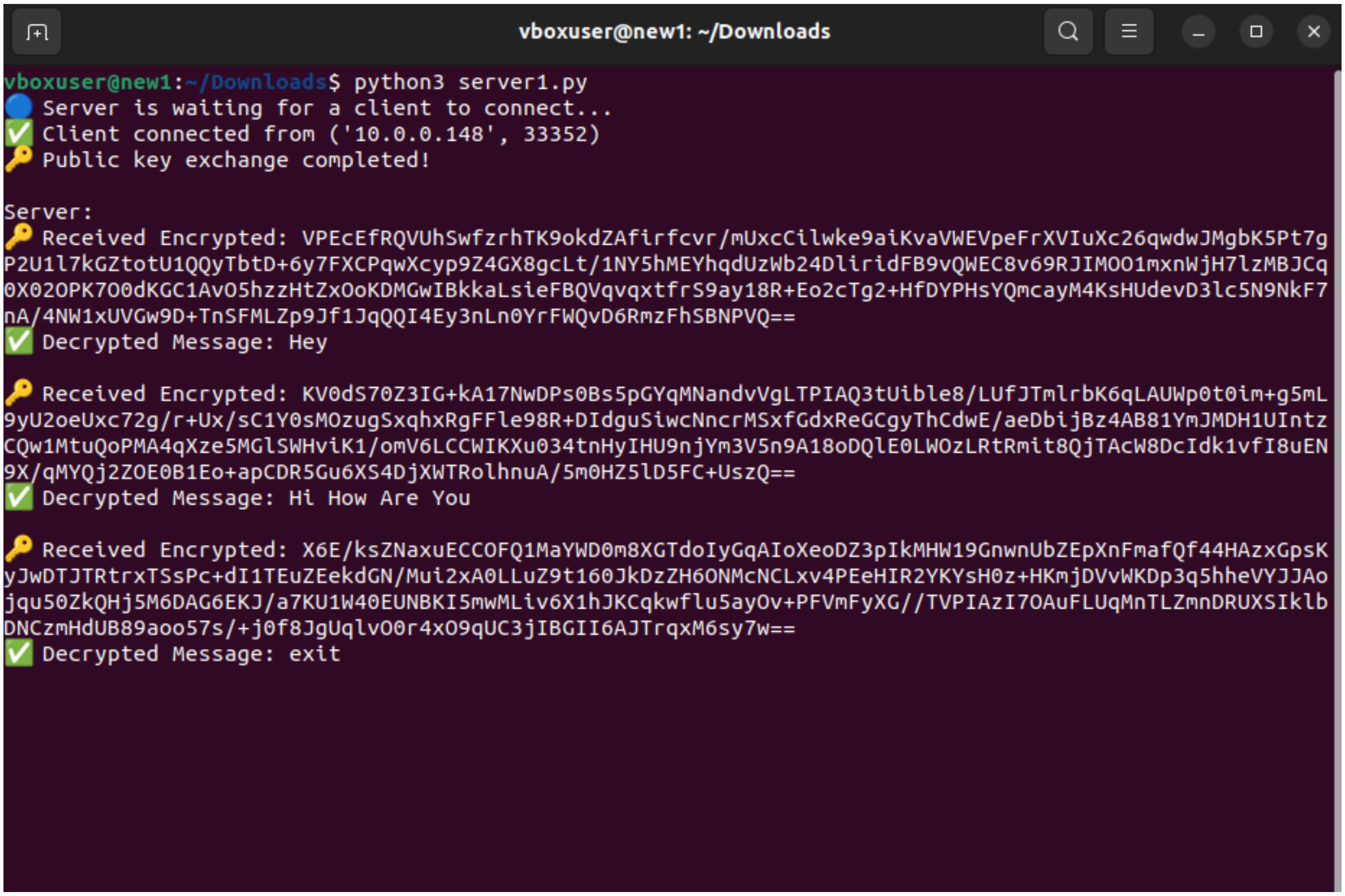
The design for this project is a client-server architecture where the messages sent between the client and server are encrypted and authenticated for each and every message and here the server starts running before the client and client connects to server and exchange the RSA encryption keys and then send the message with encryption and whenever a client/server sends exit message the communication will be closed and encryption keys are erased.



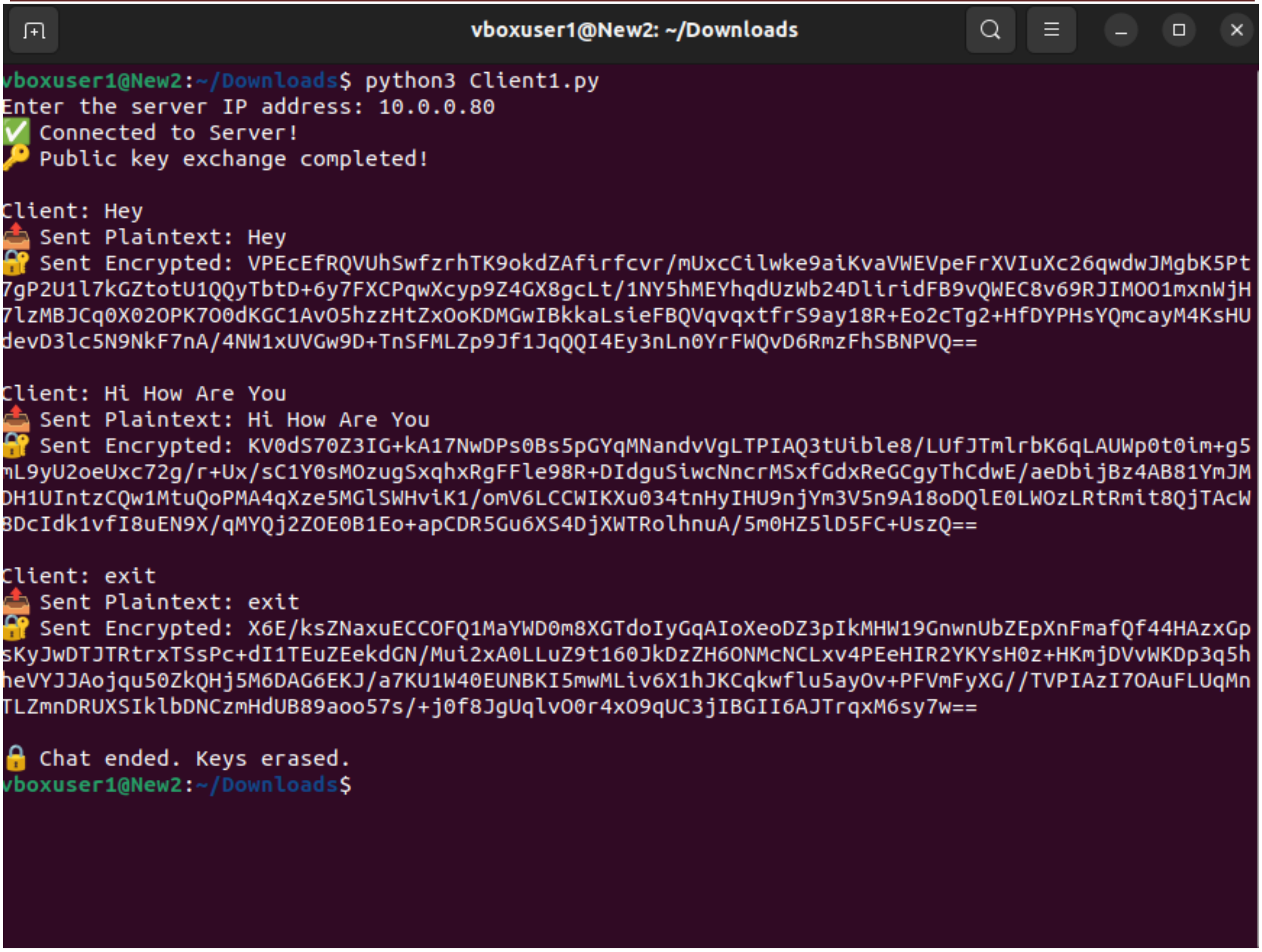
```
vboxuser@new1: ~/Downloads
vboxuser@new1:~/Downloads$ python3 server1.py
🔵 Server is waiting for a client to connect...
✅ Client connected from ('10.0.0.148', 33352)
🔑 Public key exchange completed!

Server:
🔑 Received Encrypted: VPEcEfRQVUhSwfzrhTK9okdZAfirfcvr/mUxcCilwke9aiKvaVWEVpeFrXVIuXc26qwdwJMgbK5Pt7gP2U1l7kGZtotU1QQyTbtD+6y7FXCPqwXcyp9Z4GX8gcLt/1NY5hMEYhqdUzWb24DliridFB9vQWEC8v69RJIMOO1mxnWjH7lzMBJCq0X02OPK7O0dKGC1AvO5hzzHtZxOoKDMGwIBkkaLsieFBQVqvqxtfrS9ay18R+Eo2cTg2+HfDYPHsYQmcayM4KsHUdevD3lc5N9NkF7nA/4NW1xUVGw9D+TnSFMLZp9Jf1JqQQI4Ey3nLn0YrFWQvD6RmzFhSBNPVQ==
✅ Decrypted Message: Hey

🔑 Received Encrypted: KV0dS70Z3IG+kA17NwDPs0Bs5pGYqMNandvVgLTPIAQ3tUible8/LUfJTmlrbK6qLAUWp0t0im+g5mL9yU2oeUxc72g/r+Ux/sC1Y0sMOzugSxqhxRgFFle98R+DIdguSiwcNncrMSxfGdxReGCgyThCdwE/aeDbijBz4AB81YmJMDH1UIntzCQw1MtuQoPMA4qXze5MGlSWHviK1/omV6LCCWIKXu034tnHyIHU9njYm3V5n9A18oDQlE0LWOzLRtRmit8QjTAcW8DcIdk1vfI8uEN9X/qMYQj2ZOE0B1Eo+apCDR5Gu6XS4DjXWTRolhnuA/5m0HZ5lD5FC+UszQ==
✅ Decrypted Message: Hi How Are You

🔑 Received Encrypted: X6E/ksZNaxuECCOFQ1MaYWD0m8XGTdoIyGqAIoXeoDZ3pIkMHW19GnwnUbZEpXnFmafQf44HAzxGpsKyJwDTJTRtrxTSsPc+dI1TEuZEekdGN/Mui2xA0LLuZ9t160JkDzZH6ONMcNCLxv4PEeHIR2YKYsH0z+HKmjDVvWKDp3q5hheVYJJAojqu50ZkQHj5M6DAG6EKJ/a7KU1W40EUNBKI5mwMLiv6X1hJKCqkwflu5ayOv+PFVmFyXG//TVPIAzI7OAuFLUqMnTLZmnDRUXSIklbDNCzmHdUB89aoo57s/+j0f8JgUqlvO0r4xO9qUC3jIBGII6AJTrqxM6sy7w==
✅ Decrypted Message: exit
```

Here we can see that server starts running and waiting for client to connect and once client connects it exchanges the encryption keys and starts the communication. After exchanging some messages the client send the exit message and communication will be closed and the encryption keys are erased.



Here client start running and connected to server and exchanges the encryption keys and starts the communications after exchanging some messages the client sends exit message and connection is closed. Once the connection is closed the keys are erased.

Code Explanation

SERVER CODE

- A 2048-bit R.S.A private-key is produced by rsa. generate_private_ key(). The public key for encryption is extracted using public_ key().

```
# Generate RSA Key Pair
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
public_key = private_key.public_key()
```

- The public key is **convrted to a byte format** (P.E.M) before sending.

```
# Serialize Public Key
public_key_bytes = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
```

- The servr **sends its pubic key** to the client. The server **receivs the client's public key** and diserializes it.

```
# Exchange public keys
conn.sendall(public_key_bytes)
peer_public_key_bytes = conn.recv(2048)
peer_public_key = serialization.load_pem_public_key(peer_public_key_bytes)
print("🔑 Public key exchange completed!")
```

- receivs a message that has been encrypted and signed. uses the private key on the server to decrypt the message. uses the client's public key to validate the signature. An error message appears if verification is unsuccessful.

```
def receive_messages():
    while True:
        try:
            encrypted_message = conn.recv(4096)
            signature = conn.recv(256)

            if not encrypted_message:
                break

            # Decrypt the message
            decrypted_message = private_key.decrypt(
                encrypted_message,
                padding.OAEP(
                    mgf=padding.MGF1(algorithm=hashes.SHA256()),
                    algorithm=hashes.SHA256(),
                    label=None
                )
            )

            # Verify signature
            try:
                peer_public_key.verify(
                    signature,
                    decrypted_message,
                    padding.PSS(
                        mgf=padding.MGF1(hashes.SHA256()),
                        salt_length=padding.PSS.MAX_LENGTH
                    ),
                    hashes.SHA256()
                )
                print(f"\n🔑 Received Encrypted:
{base64.b64encode(encrypted_message).decode()}")
                print(f"✅ Decrypted Message: {decrypted_message.decode()}")
            except:
                print("❌ Signature verification failed!")

            if decrypted_message.decode().strip().lower() == "exit":
                break

        except:
            break
```

- The client's publc key is used to encrypt the message. The private key of the servr is used to establish a digitl signature. Together, the signd document and encrypted mesage are transmited..

```
# Sending messages
while True:
    message = input("\nServer: ").encode()

    encrypted_message = peer_public_key.encrypt(
        message,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

    # Sign the message
    signature = private_key.sign(
        message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )

    conn.sendall(encrypted_message)
    conn.sendall(signature)

    print(f"📤 Sent Plaintext: {message.decode()}")
    print(f"🔐 Sent Encrypted: {base64.b64encode(encrypted_message).decode()}")

    if message.decode().strip().lower() == "exit":
        break
```

- The connection is **closed**, and the **keys are deleted** for security.

```
# Close connection and erase keys
conn.close()
del private_key, public_key, peer_public_key
print("\n🔒 Chat ended. Keys erased.")
```

CLIENT CODE

The **client code** is almost identical to the **server**, except:
• Rather than waiting for a connection, it establishes a connection with the server. After connecting, it exchanges public keys. The encryption, signing, decryption, and verification procedures are all the same.

```
# Connect to Server
server_ip = input("Enter the server IP address: ").strip()
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_ip, 12345))
print("✅ Connected to Server!")
```

- The client **receives the server's public key** and **sends its own**.

```
# Exchange public keys
peer_public_key_bytes = client_socket.recv(2048)
peer_public_key = serialization.load_pem_public_key(peer_public_key_bytes)
client_socket.sendall(public_key_bytes)
print("🔑 Public key exchange completed!")
```

- **Closes the connection** and **removes keys** for security.

```
# Close connection and erase keys
client_socket.close()
del private_key, public_key, peer_public_key
print("\n🔒 Chat ended. Keys erased.")
```