Steps to build and execute the code:

Go to the source code dir and run the following commands in successive order:

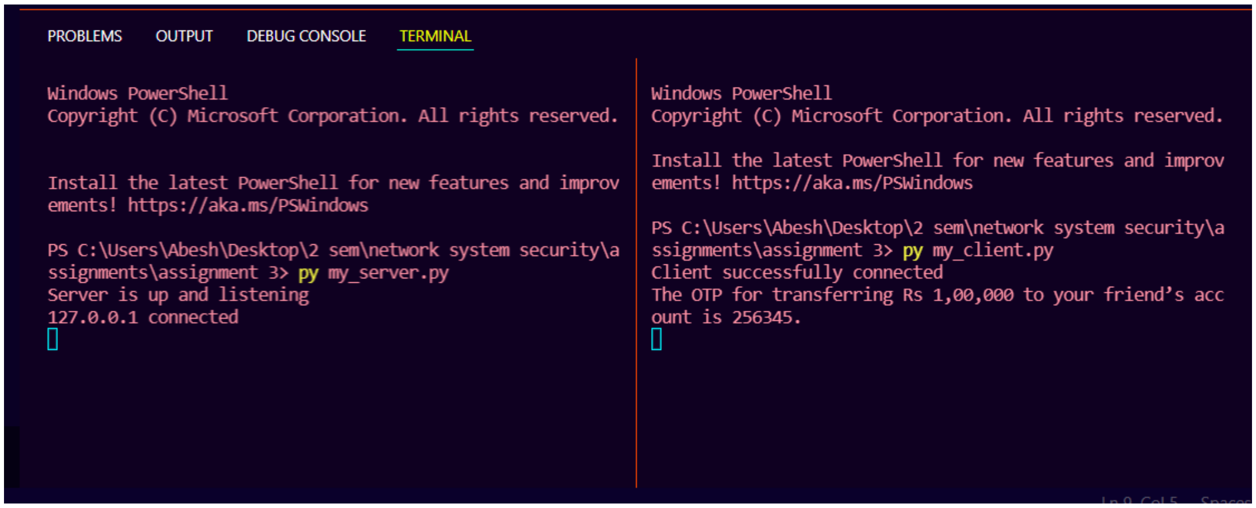
py my_ttp.py

py my_server.py

py my_client.py

Steps taken:

1) **Trivial and Basic Approach:**
   First I tried the simplest and the most basic approach to directly send the message from server to the client just using socket programming. The results obtained were :



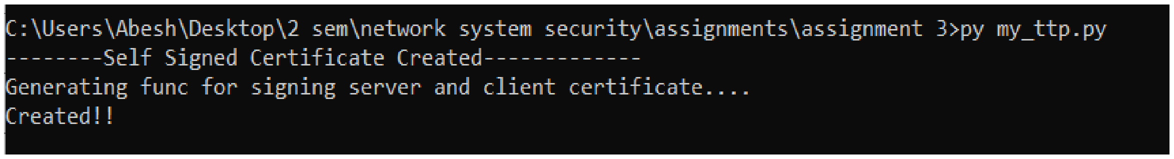
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.


Install the latest PowerShell for new features and improv
ements! https://aka.ms/PSWindows

PS C:\Users\Abesh\Desktop\2 sem\network system security\a
ssignments\assignment 3> py my_server.py
Server is up and listening
127.0.0.1 connected
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improv
ements! https://aka.ms/PSWindows

PS C:\Users\Abesh\Desktop\2 sem\network system security\a
ssignments\assignment 3> py my_client.py
Client successfully connected
The OTP for transferring Rs 1,00,000 to your friend's acc
ount is 256345.
```

   It can be clearly seen that in this approach the text is being sent in plain text and has very serious security issues like data confidentiality and anyone can see the message. Thus, data encryption is required.

2) **Need for Encryption:**
   Since TLS follows, MAC then encrypt for maximum security, I will this approach for it. But the symmetric key is established in the handshake phase. So, first authentic key exchange is required. So, a private/public key is needed to be generated in the server. I will use RSA-2048 bit for this. I used crypto library from open ssl library to achieve both public and private key and store it in a file.

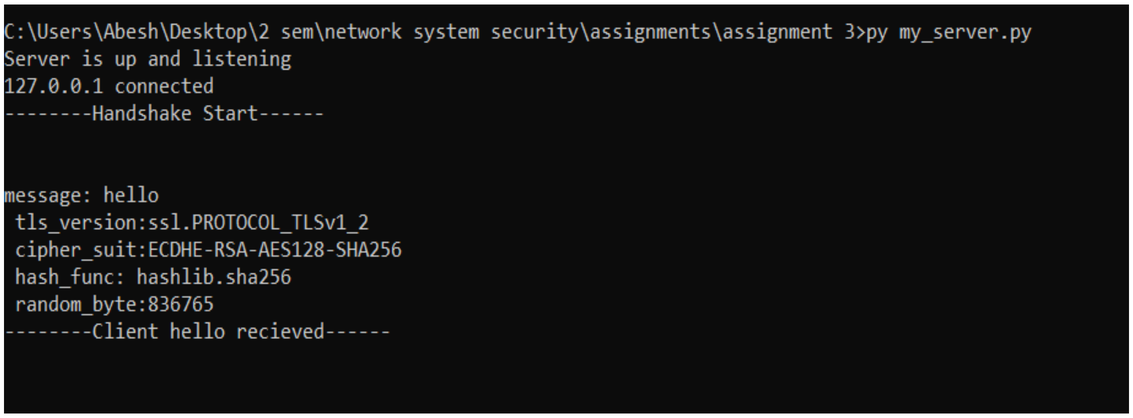
3) **CERTIFICATE GENERATION FORM TTP:**
   a. TTP generates a self-signed certificate which is used to create a certificate chain to verify other requested certificates.
   b. Both the client and the server use their public key and their associated identity and send the certificate generation request to the TTP. The TTP then signs the request with its own private key to generate an authenticated certificate which can be used and verified with anyone having TTP public key. Again, crypto library from openssl is used for this purpose. **RSA-2048 with sha-256 is used for self signed certificate by TTP whereas RSA-2048 with sha-384 is used for certificate generation of server and client.**



```
C:\Users\Abesh\Desktop\2 sem\network system security\assignments\assignment 3>py my_ttp.py
--------Self Signed Certificate Created-------------
Generating func for signing server and client certificate....
Created!!
```

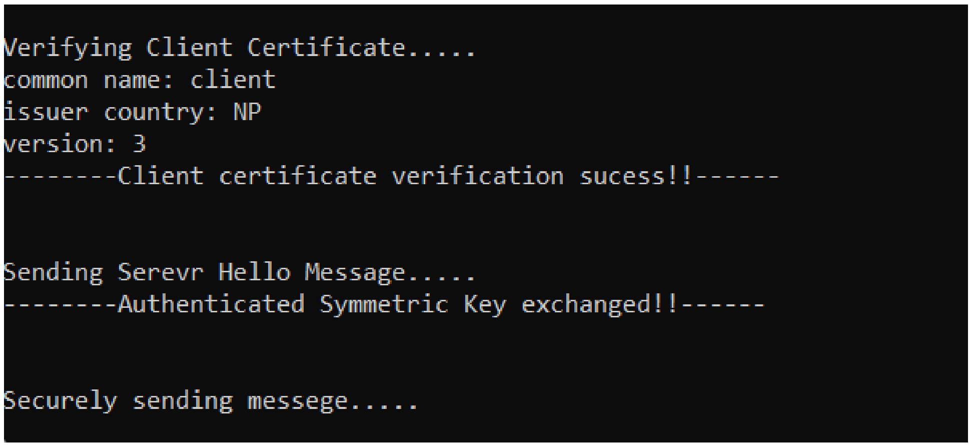
4) HandShake Protocol
   a. Protocol Negotiation: In this step of implementation, I send a client hello message along with the TLS version, hashing algorithm to be used, cipher suite and finally the version. The server on receiving the hello message, requests client for its certificate and also sends its certificate for verification.

   Server receiving hello and protocol negotiation from client.



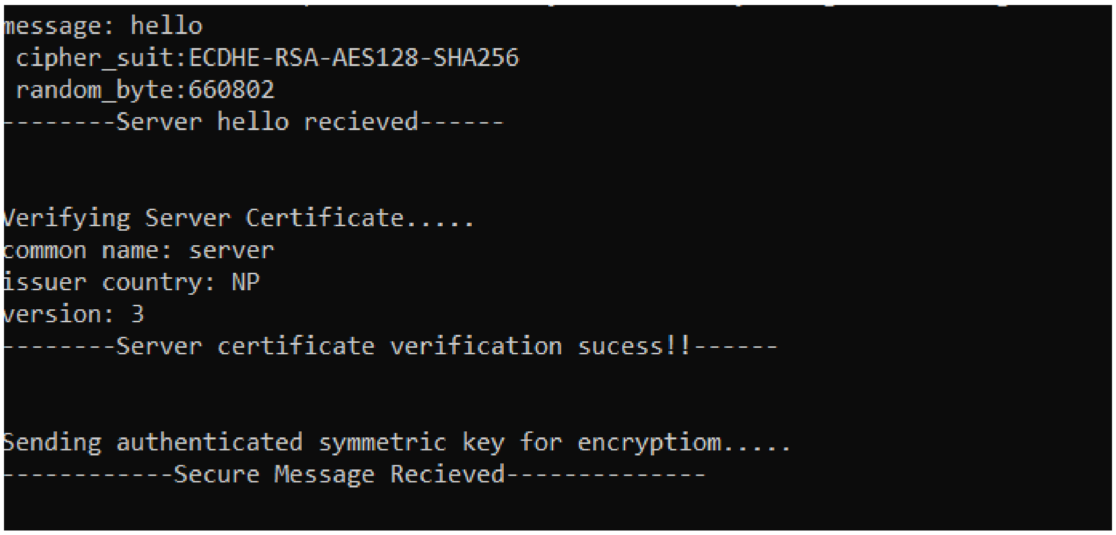
```
C:\Users\Abesh\Desktop\2 sem\network system security\assignments\assignment 3>py my_server.py
Server is up and listening
127.0.0.1 connected
--------Handshake Start------


message: hello
 tls_version:ssl.PROTOCOL_TLSv1_2
 cipher_suit:ECDHE-RSA-AES128-SHA256
 hash_func: hashlib.sha256
 random_byte:836765
--------Client hello recieved------
```

Server verifying client certificate.



```
Verifying Client Certificate.....
common name: client
issuer country: NP
version: 3
--------Client certificate verification sucess!!------


Sending Serevr Hello Message.....
--------Authenticated Symmetric Key exchanged!!------


Securely sending messege.....
```

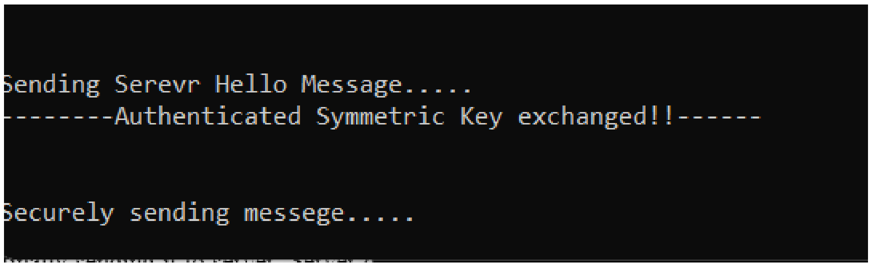
Client verifying server certificate, extracting the public key of server from the certificate and then generating an symmetric key encrypted by server's public key and finally sending it to server. Server decrypting the encrypted message with its own private key and then extracting the symmetric for future encryption and authentication of the messages**. For this purpose I have used, RSA-2048-Enc based encryption and decryption.** RSA is



```
message: hello
 cipher_suit:ECDHE-RSA-AES128-SHA256
 random_byte:660802
--------Server hello recieved------


Verifying Server Certificate.....
common name: server
issuer country: NP
version: 3
--------Server certificate verification sucess!!------


Sending authenticated symmetric key for encryptiom.....
------------Secure Message Recieved--------------
```

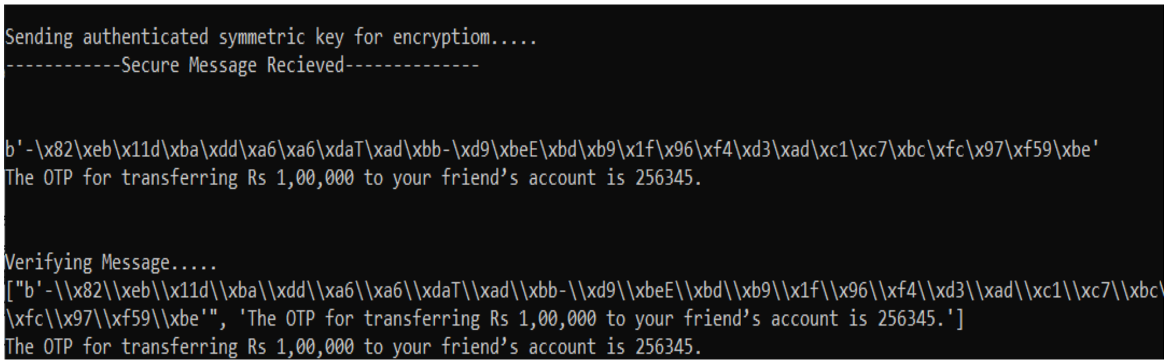
b. Authenticated key exchange: I have used digital signature based key exchange for this purpose. The client requests server's certificate from TTP and verifies the identity and the server also does the same. The public key is then obtained from the associated certificate.

c. Key Transcript Confirmation: Once authenticated key exchange is done, and both server and client has each others public's key, the client generated an asymmetric key for message encryption and encrypts this key using the public key of the server. The server on receiving this encrypted key, decrypts with its own private key to receive the symmetric key for further message encryption.



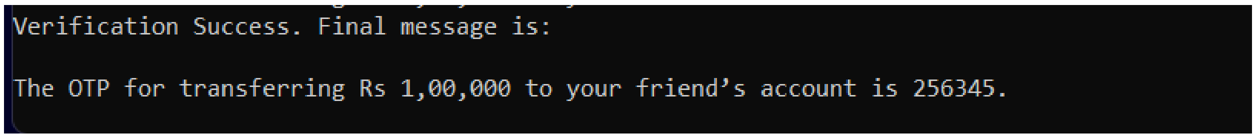
```
Sending Serevr Hello Message.....
--------Authenticated Symmetric Key exchanged!!------


Securely sending messege.....
```

Encrypted message by the symmetric key received at the client side:



```
Sending authenticated symmetric key for encryptiom.....
------------Secure Message Recieved--------------


b'-\x82\xeb\x11d\xba\xdd\xa6\xa6\xdaT\xad\xbb-\xd9\xbeE\xbd\xb9\x1f\x96\xf4\xd3\xad\xc1\xc7\xbc\xfc\x97\xf59\xbe'
The OTP for transferring Rs 1,00,000 to your friend's account is 256345.


Verifying Message.....
["b'-\\x82\\xeb\\x11d\\xba\\xdd\\xa6\\xa6\\xdaT\\xad\\xbb-\\xd9\\xbeE\\xbd\\xb9\\x1f\\x96\\xf4\\xd3\\xad\\xc1\\xc7\\xbc\
\xfc\\x97\\xf59\\xbe'", 'The OTP for transferring Rs 1,00,000 to your friend's account is 256345.']
The OTP for transferring Rs 1,00,000 to your friend's account is 256345.
```

5) Record Protocol:
   a. After both the server and the client has the same symmetric key, the server encrypts the desired message with this key and sends to the client. The client uses the same key to decrypt the message and obtained the plaintext.
   b. After the message is received, the client sends an acknowledgement and then the server can perform a secure communication. **For this purpose I have used HMAC with sha-256 as the hashing function.**


```
Verification Success. Final message is:

The OTP for transferring Rs 1,00,000 to your friend’s account is 256345.
```

Hence, the decrypted and authenticated message received at the client in a secure communication channel.

Security and efficiency of TLS protocol is TLS helps in encrypting the message that is sent over the web. If the data is not encrypted, spoofing, man in the middle attack can occur and all our private information’s like passwords can be compromised. Thus, encryption of data becomes very essential in such cases. Hence, TLS protocol helps in achieving this security and reliability in communication on the web. Also, the efficiency in terms of computational cost is asymmetric cryptography is used for key exchange but this is only one time. After that, symmetric key is used for further encryption of messages which is efficient.