

SIL 765: Network & Systems Security

Assignment-3

Abhisek Panda (2020CSZ2445)

April 15, 2020

Problem Statement :

Prototype a transport layer protocol (TLS) to send a message from a client to a server using a trusted third party.

System Requirement:

Below python3 modules must be installed using pip3 utility:

- socket
- pickle
- time
- os
- random
- cryptography
- socketserver
- Sys
- datetime

Cipher Suites :

Below cipher suites are utilized in the prototype:

- Asymmetric key algorithm: ECDSA or RSA,
- Symmetric key algorithm: AES or CHACHA20, and
- Hashing algorithm: SHA256 or SHA384
- Digital Signature: ECDSA with SHA384 or ECDSA with SHA256 or RSA with SHA256 or RSA with SHA384.
- Key Exchange Algorithm: Authenticated Diffie-Hellman (key-size: 1024 bit) or Authenticated Elliptical Curve Diffie-Hellman.
- MAC Algorithm: CMAC or HMAC

Implementation:

The proposed prototype consists of three entities: Trusted Third Party (TTP), Server, and Client. In the following subsection, we would discuss each of the entities and inter-communication between them.

Trusted Third Party (TTP):

The trusted third party is responsible for issuing the digital certificates for the client and server. To issue a digital certificate, either the server or the client must provide their public key of the entity and the subject information object. The subject information object contains the COUNTRY_NAME, STATE_OR_PROVINCE_NAME, LOCALITY_NAME, ORGANIZATION_NAME, and COMMON_NAME to generate the subject information for the certificate. The TTP responds with the certificate, digital signature specification ID, length, and public key of TTP in response.

Server:

The COMMON_NAME can uniquely identify the server, SIL765.iitd.ac.in. The server accepts the request from the client. Using handshake protocol establishes master_key, mac_secret, and cipher_nonce with the client, further utilized in record protocol. The server is responsible for generating the OTP for the transaction request made by the client.

Client:

The COMMON_NAME can uniquely identify the client, csz202445.iitd.ac.in. The client sends the transaction request to the server. Using handshake protocol establishes master_key, mac_secret, and cipher_nonce with the client, further utilized in record protocol. The client is responsible for making a transaction request to the server.

Intercommunication:

The intercommunication between the TTP, Server, and Client (Fig-1) can be summarized below:

1. The server initializes the RSA and ECDSA public key, Diffie Hileman Parameters, subject information object, and other necessary parameters.
2. The server requests the digital certificate from the TTP for the public keys.
3. After receiving the digital certificate, the server waits for the client to connect.
4. The client requests the digital certificate from the TTP for the RSA and ECDSA public keys.
5. After receiving the digital certificate, the client connects to the server.
6. The client and the server establishes the master_key (for symmetric encryption), mac_secret (for MAC algorithm), and a nonce (for CTR mode and CHAHCHA20 algorithm) using the handshake protocol.
7. The client sends the client_hello message, which contains parameters such as version, random, cipher_suite, compression_method, and session_id to the server.
8. The server processes the client_hello message and decides upon the suitable session parameters, version, random, cipher_suite, digital_signature, compression_method, and session_id.
9. The server sends its certificate to the client.
10. Subsequently, the server sends the server key exchange message, which contains the key exchange parameters agreed upon, followed by the parameters' digital signature using the appropriate specification.
11. The server sends the certificate request message to notify the client to provide its certificate, followed by the server-done message.

12. On receiving the above messages, the client verifies the parameters' digital signature using the server's public key provided in the certificate, followed by generating the appropriate exchange parameters.
13. The client sends the certificate to the server.
14. Subsequently, the client sends the client key exchange message, which contains the key exchange parameters agreed upon, followed by the parameters' digital signature using the appropriate specification.
15. The client sends the certificate_verify message, which contains the signature of the handshake_messages, i.e., all Handshake Protocol messages sent or received starting at client_hello but not including this message.
16. The server and client use the HMAC utility to generate the master_secret, mac_secret, and a nonce.
 - master_key = HMAC(pre_master_key, "master_key" || client.random || server.random)
 - mac_secret = HMAC(pre_master_key, "mac_secret" || client.random || server.random)
 - cipher_nonce = HMAC(pre_master_key, "nonce" || client.random || server.random)
17. For sending a message over TLS, the below steps are executed in order:
 - Generate the MAC of the message using the mac_algorithm agreed by the client and server using the mac_secret key.
 - Append MAC to the message.
 - Encrypt the message_mac with the cipher_algo agreed by the client and server using the master_key. Based upon cipher_algo, the AES algorithm is used in CTR mode initialized with cipher_nonce, and the CHACHA20 algorithm is initialized with a cipher_nonce.
 - Create a request object with attributes message, size, and length. The message attribute contains the ciphertext, the length contains the length of the original message, and the size contains the size of message in bytes.
18. After receiving a message over TLS, the below steps are executed in order:
 - Decrypt the message_mac with the cipher_algo agreed by the client and server using the master_key. Based upon cipher_algo, the AES algorithm is used in CTR mode using the cipher_nonce and the CHACHA20 algorithm is initialized with a cipher_nonce.
 - Based on the length attribute of the request object, split the decrypted message to extract the message and MAC.
 - Generate the MAC of the message using the mac_algorithm agreed by the client and server using the mac_secret key. Verify it against the received MAC.
 - After verification sends the message to the higher layers of TCP/IP protocol

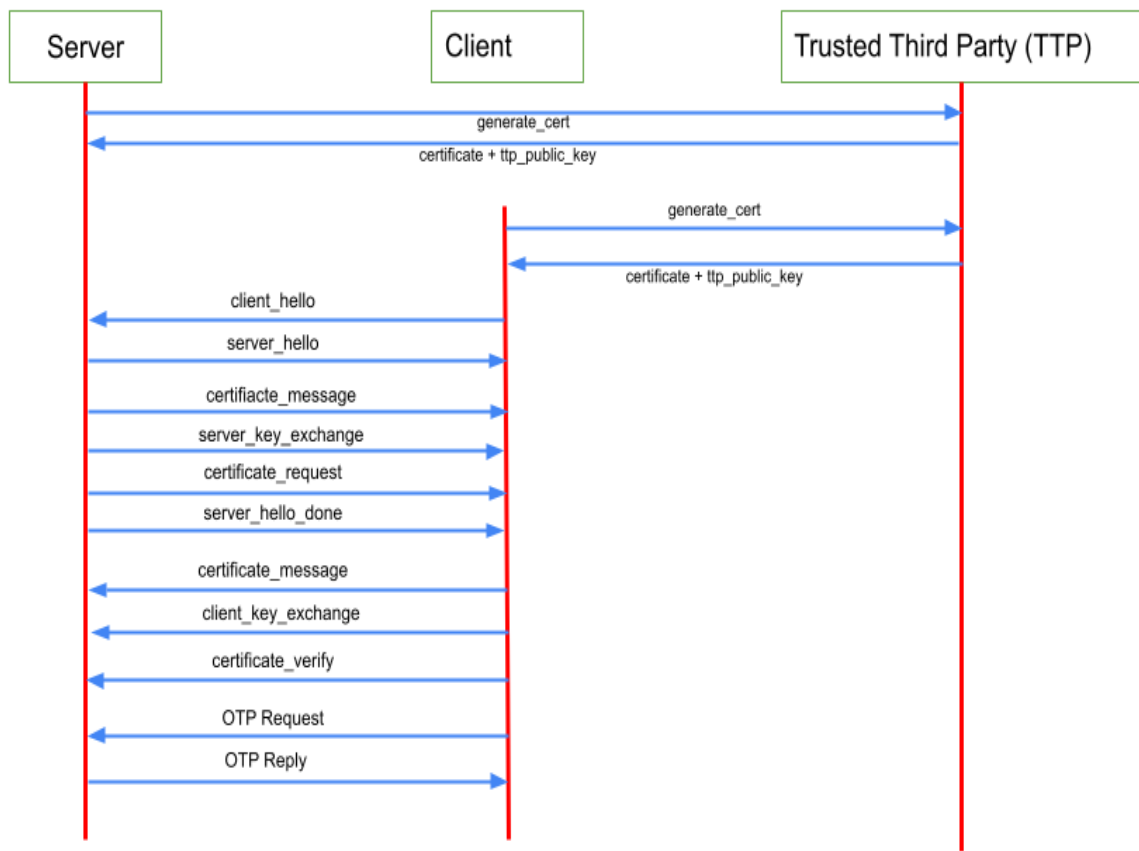


Fig-1: TLS communication between the entities

Security Analysis:

The TLS prototype provides security against Man in the Middle Attack, Replay Attack, and Downgrade Service attack in the following ways:

- In the handshake protocol, the server and client both use the random number or nonce in client_hello and server_hello message. Hence the replay attack is not possible.
- In the handshake protocol, the server and client both share the certificate message to each other. While sharing the server_key_exchange message and client_key_exchange_message, the signature is added to verify the authenticity of the message. Therefore, Man in the Middle attack is prevented.
- In the handshake protocol, the certificate_verify message contains the signature of the handshake_messages, i.e., all Handshake Protocol messages sent or received starting at client_hello but not including this message. At the server, the cross-validation of the signature generated is performed. Therefore, the downgrade service attack is prevented.

Computational Cost Analysis:

The total number of symmetric encryptions was used = 1 (record_protocol / request)

The total number of asymmetric encryptions was used = 4 (cert creation) + 2 (key_exchange) +
1(cert_verify)
= 7

The total number of encryption operations used = 8

The total number of symmetric decryptions was used = 1 (record_protocol / request)

The total number of asymmetric decryptions was used = 2 (cert verification) + 2 (key_exchange)
1(cert_verify)
= 7

The total number of decryption operations used = 8

Total Encryption and Decryption performed = 16

Communication Cost Analysis:

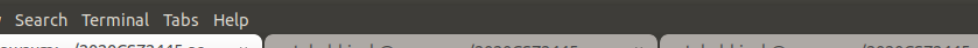
<u>Parameters</u>	<u>Size (in Bytes)</u>
generate_cert message	248
certificate message	248
client_hello	240
server_hello	128
certificate_message	1026
server_key_exchange	240
certificate_request	240
server_hello_done	49
certificate_message	1030
client_key_exchange	240
certificate_verify	240
OTP_Request	126
OTP_Reply	139

Total Communication Cost = 5442 Bytes

For executing the TLS prototype, open three terminals and execute the below commands in the mentioned order on each terminal:

```
$ python3 my_client.py
```

1. Starting the TTP:



```
catchabhisek@awsun: ~/2020CSZ2445-assignment-3
File Edit View Search Terminal Tabs Help
catchabhisek@awsun: ~/2020CSZ2445-as... x catchabhisek@awsun: ~/2020CSZ2445-as... x catchabhisek@awsun: ~/2020CSZ2445-ass... x
catchabhisek@awsun:~/2020CSZ2445-assignment-3$ python3 my_ttp.py
The Trusted Third Party Server is running. Press Ctrl + C to close the server
The issuer have the following Identity have COMMON_NAME: iitd.ac.in
The Trusted Third Party server uses the RSA+SHA384 specification for Digital Signature
-----
```

Fig-2: TTP Server Started

2. generate cert message and certification message between server and TTP:

[illegible]

Fig-3: generate cert and certificate at Server

```

catchabhisek@awsun: ~/2020CSZ2445-assignment-3
File Edit View Search Terminal Tabs Help
catchabhisek@awsun: ~/2020CSZ2445-assignment-3$ python3 my_client.py
The Trusted Third Party Server is running. Press Ctrl + C to close the server
The issuer have the following Identity have COMMON_NAME: iitd.ac.in
The Trusted Third Party server uses the RSA+SHA384 specification for Digital Signature

-----
Host 127.0.0.1 requesting to generate certificate
Generated Certificate for the Host 127.0.0.1 is: b'-----BEGIN CERTIFICATE-----\nMIICp
TCCAY2gAwIBAgIUesxijELcWlc0mD00TVp0lqwLu7IwDQYJKoZIhvcNAQEM\nBQAwVjELMAkGA1UEBhMCSU4xZDZ
NBGNVBAgMBK9yaXNzYTESMBAGA1UEBwwJQmVY\naGFTcHlVYMQ0wCwYDVQQKDARJSVREMRMwEQYDVQDDAppaXRkL
mFjLmUuMB4XDTIx\nMDQxNDEyMzAyM1oXDTIxMDUxNTEyMzAyM1owXDELMakGA1UEBhMCSU4xZDZjAMBGNV\nBAGMB
URFTEhJMRiEAYDVQHDALiQVVAIEtIQVoxDTALBgNVBAoMBELJVEQxGjAY\nBGNVBAgMBEVBjJDE2NS5paXRkLmF
jLmUuMHYwEAYHkoZiZj0CAQYFK4EEACIDYgAE\n87bdDRaQJGFzd0PJ2JT4vL3Cq4qgXv7k5qDpuIqZwXrGPMB
k8P21K4XnPZQD8K\nnxuhqFVxQgYJrTbJdGeVEuzgcpD+fA+0YwMtIJJ0n52GnHnIRAKue+r5rbrBB6VMc\nnoXmWwE
TAPBGNVHRMBAF8EBTADAQH/MA0GCSqGSIb3DQEBAUAA4IBAQAk4MiOk1kL\nnJnmS6NqDKnzPJF5FW6Ues/JZntD
bo4oDxiUJ4SNqN8fmL04nUQcIJEp9q0QRWBj\nn4Q7F7pL4pTLRm5q4aJNEQ/Ytb78hI5Y4h0Xa52FwKGNUoY9LB
x4TB+uD+qdzf1jY\nnAvXJarsbPula0uiW0mwxn9B18xQ3UBuGoIIJZGgF02twIBpgd9/TquILbfpDg/5\nnnNdR3
qsHcw2LcW7vHEE4q/6YWE+7/hekTittjKbgs6tZ9P3ERXglWjPz/e3wFT8Q\nn5Nvd86dpXn87b0wLVjMBfG8f4gG
meGP5NUcsuUse7NGUq42H7/QQyqIwLmfqtlm4\nnr1MEvrsGz/tL\n-----END CERTIFICATE-----\n'
-----

```

Fig-4: generate_cert and certificate at TTP

3. generate_cert message and certification message between client and TTP:

```

catchabhisek@awsun: ~/2020CSZ2445-assignment-3
File Edit View Search Terminal Tabs Help
catchabhisek@awsun: ~/2020CSZ2445-assignment-3$ python3 my_client.py
The Client is requesting for OTP for transaction from server.
The subject have the following Identity, COMMON_NAME: csz2445.iitd.ac.in

-----
Requesting Trusted Third Party to generate a Certificate running on port 4444

-----
Request for certificate for ECDSA based public key is made:
{'subject_info': {'COUNTRY_NAME': 'IN', 'STATE_OR_PROVINCE_NAME': 'DELHI', 'LOCALITY_NAME': 'HAUZ KHAZ', 'ORGANIZATION_NAME': 'IITD', 'COMMON_NAME': 'csz
2445.iitd.ac.in'}, 'public_key': b'-----BEGIN PUBLIC KEY-----\nMHYwEAYHkoZiZj0CAQYFK4EEACIDYgAE\nJ34Mu41Fxn10Z3qneZYPNktl3C5L83\ncpUNjtvKyrLZoawp/KQHhwPK
B4JDKPzGoVU0cJZuUguA40Vky0020r2x5JWS\n62p4TJb+FDf1l2EYXJLczw0R2cpq34\n-----END PUBLIC KEY-----\n', 'type': 'generate_cert'}

-----
Certificate for ECDSA based public key received from Trusted Third Party:
b'-----BEGIN CERTIFICATE-----\nMIICpJCCAY6gAwIBAgIUf1y+9UGPLQVA8ozZkFkk8w3leyQwDQYJKoZIhvcNAQEM\nBQAwVjELMAkGA1UEBhMCSU4xZDZjAMBGNVBAgMBK9yaXNzYTESMBAGA1UEBwwJQmVY\naGFTcHlVYMQ0wCwYDVQQKDARJSVREMRMwEQYDVQDDAppaXRkLmFjLmUuMB4XDTIx\nMDQxNDEyMzAyM1oXDTIxMDUxNTEyMzAyM1owXDELMakGA1UEBhMCSU4xZDZjAMBGNV\nBAGMB
URFTEhJMRiEAYDVQHDALiQVVAIEtIQVoxDTALBgNVBAoMBELJVEQxGjAY\nBGNVBAgMBEVBjJDE2NS5paXRkLmF
jLmUuMHYwEAYHkoZiZj0CAQYFK4EEACIDYgAE\n87bdDRaQJGFzd0PJ2JT4vL3Cq4qgXv7k5qDpuIqZwXrGPMB
k8P21K4XnPZQD8K\nnxuhqFVxQgYJrTbJdGeVEuzgcpD+fA+0YwMtIJJ0n52GnHnIRAKue+r5rbrBB6VMc\nnoXmWwE
TAPBGNVHRMBAF8EBTADAQH/MA0GCSqGSIb3DQEBAUAA4IBAQAk4MiOk1kL\nnJnmS6NqDKnzPJF5FW6Ues/JZntD
bo4oDxiUJ4SNqN8fmL04nUQcIJEp9q0QRWBj\nn4Q7F7pL4pTLRm5q4aJNEQ/Ytb78hI5Y4h0Xa52FwKGNUoY9LB
x4TB+uD+qdzf1jY\nnAvXJarsbPula0uiW0mwxn9B18xQ3UBuGoIIJZGgF02twIBpgd9/TquILbfpDg/5\nnnNdR3
qsHcw2LcW7vHEE4q/6YWE+7/hekTittjKbgs6tZ9P3ERXglWjPz/e3wFT8Q\nn5Nvd86dpXn87b0wLVjMBfG8f4gG
meGP5NUcsuUse7NGUq42H7/QQyqIwLmfqtlm4\nnr1MEvrsGz/tL\n-----END CERTIFICATE-----\n'

-----
Request for certificate for RSA based public key is made:
{'subject_info': {'COUNTRY_NAME': 'IN', 'STATE_OR_PROVINCE_NAME': 'DELHI', 'LOCALITY_NAME': 'HAUZ KHAZ', 'ORGANIZATION_NAME': 'IITD', 'COMMON_NAME': 'csz
2445.iitd.ac.in'}, 'public_key': b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1zsVLSxR2U2Vyr/ehQ00\nndftrQikqhsBuKI71HeHBAFT
NxKtzuIcQgU7Pup7w0bZiuV4veQEu21h/ZxcvKS9\nnL3X7ypyrx3SGxNNGR8T4a4PhYItt40zChcIspJ7ycInA0SA7MDzGzG670/sg/H\nnWY+QDRKskKFubJBtP9unzHqMYMMAMuDLfS+KTQDj0Hvi
VtPip/jIISXF/cpgGPP\nnOKz7tda5PoKLNRMVRYfC+qQExL9X1xiH1L2JfKqLZydxNdazG18CddBuyeh\nn7j+11dt26TSCXGm72zWLU8Ywtd3tnSvBoapa9cc1zS0U/0v7N/p8sL9LW8AA0Mj\n
9wIDAQAB\n-----END PUBLIC KEY-----\n', 'type': 'generate_cert'}

-----
Certificate for RSA based public key received from Trusted Third Party:
b'-----BEGIN CERTIFICATE-----\nMIIDVCCAiygAwIBAgIUfBAffR330kSmaJhDvkuBt3q5IwWdQYJKoZIhvcNAQEM\nBQAwVjELMAkGA1UEBhMCSU4xZDZjAMBGNVBAgMBK9yaXNzYTESMBAGA1UEBwwJQmVY\naGFTcHlVYMQ0wCwYDVQQKDARJSVREMRMwEQYDVQDDAppaXRkLmFjLmUuMB4XDTIx\nMDQxNDEyMzAyM1oXDTIxMDUxNTEyMzAyM1owXDELMakGA1UEBhMCSU4xZDZjAMBGNV\nBAGMB
URFTEhJMRiEAYDVQHDALiQVVAIEtIQVoxDTALBgNVBAoMBELJVEQxGjAY\nBGNVBAgMBEVBjJDE2NS5paXRkLmF
jLmUuMHYwEAYHkoZiZj0CAQYFK4EEACIDYgAE\n87bdDRaQJGFzd0PJ2JT4vL3Cq4qgXv7k5qDpuIqZwXrGPMB
k8P21K4XnPZQD8K\nnxuhqFVxQgYJrTbJdGeVEuzgcpD+fA+0YwMtIJJ0n52GnHnIRAKue+r5rbrBB6VMc\nnoXmWwE
TAPBGNVHRMBAF8EBTADAQH/MA0GCSqGSIb3DQEBAUAA4IBAQAk4MiOk1kL\nnJnmS6NqDKnzPJF5FW6Ues/JZntD
bo4oDxiUJ4SNqN8fmL04nUQcIJEp9q0QRWBj\nn4Q7F7pL4pTLRm5q4aJNEQ/Ytb78hI5Y4h0Xa52FwKGNUoY9LB
x4TB+uD+qdzf1jY\nnAvXJarsbPula0uiW0mwxn9B18xQ3UBuGoIIJZGgF02twIBpgd9/TquILbfpDg/5\nnnNdR3
qsHcw2LcW7vHEE4q/6YWE+7/hekTittjKbgs6tZ9P3ERXglWjPz/e3wFT8Q\nn5Nvd86dpXn87b0wLVjMBfG8f4gG
meGP5NUcsuUse7NGUq42H7/QQyqIwLmfqtlm4\nnr1MEvrsGz/tL\n-----END CERTIFICATE-----\n'
-----

```

Fig-5: generate_cert and certificate at Client


```
Host 127.0.0.1 requesting to generate certificate
Generated Certificate for the Host 127.0.0.1 is: b'-----BEGIN CERTIFICATE-----\nMIICpCjCCAY6gAwIBAgIUfY1+9UGPLQVA8ozZkFkk8w3leyOqDQYJKoZIhvcNAQEM\nbNBAQwVJELMAKGA1UEBmMCSUx4dXZANBgNVBAgMBk9yaXNzYTESBAGIA1UEBmQwJmVl\nnaGFTcHdhVjQ0ODARJSVERMRmMEQDYVQQDDAppaXRKLmF1LmLmB4XDITXi\nnMDQxNDExeDNM0NF0XDTIeMDXuNmDM0NF0XDTLMCA1UEBiBmVmlnaGMBURFTEhJMREWEADYVQVQHDALIQVVAIEtIQvxdDALBGnvBAOMBELEJVEqxGZAznbgNBVAMMEMNejeJNDUuaWl0ZCShy5pbjB2MABGaqGSM\nq9AagEGBSUbBAAIa2IA\nnBJSSedLnRcZ7JDgd6p3mWdzSrZdwuzFN3KVdy7VSsq5kaGlqfykB8ncxcjFVNHI2VLKILgOLDLZksjjttkt8duSVkutgeYh/hqxzdzcRGF4y3M8DKhKKat+nP\nKqzMBEdmWdVDVR0TAQH/BAAUAWEB/zAlNbgahtkiC9w0BAQwFAOCQAQEASsoArm0s\nnMXvf7mybPc/gNGC54yBE4US6Pa+8CA+H+tIKg0degqw//KGZ2PD5ZXBK43xIey0/nbZYAu0kv8KDzuZ4nnLo\n+Hz/Vmxz/exwZphfZ15btX3h0PVNM0Ic1j4C5UCj9nGa3jYlhJhl5l1ciGyhA3xL8Snwn76w3ret0lfTP8BVVF7gm8BMCMQNZF0yz2InEno+PuL/Eyvba7xjwSLr2ug/SPTQW32DohCs45RIrk0\nhvXB3XZg73IEZ5RsdfPg\nng5N4l+RKQ0rUeFU9nzFJ+y4T+7gwE9CX07ufuyWxiwNOJHCioqGMgtXsd7Mxf/h/nQ1XW3IXCEe4og=\n-----END CERTIFICATE-----\n'
```

Fig-6: generate_cert and certificate at TTP

4. client_hello and server_hello messages between client and server:

```
-----
Starting Handshake Protocol with the Server running on 4445
Sending the client hello message to server:
The request parameters are: {'type': 'client_hello', 'content': {'version': 1.3, 'random': '704514256314735072489591610712892088858841', 'session_id': '6013168', 'cipher_suite': [{'key_exchg_algo': 'DH', 'cipher_algo': 'AES', 'mac_algo': 'HMAC', 'cipher_type': 'Block'}, {'key_exchg_algo': 'DH', 'cipher_algo': 'AES', 'mac_algo': 'CMAC', 'cipher_type': 'Block'}, {'key_exchg_algo': 'DH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'HMAC', 'cipher_type': 'Stream'}, {'key_exchg_algo': 'DH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'CMAC', 'cipher_type': 'Stream'}, {'key_exchg_algo': 'ECDH', 'cipher_algo': 'AES', 'mac_algo': 'HMAC', 'cipher_type': 'Block'}, {'key_exchg_algo': 'ECDH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'HMAC', 'cipher_type': 'Stream'}, {'key_exchg_algo': 'ECDH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'CMAC', 'cipher_type': 'Stream'}], 'compression_method': [], 'length': 240}}
Received the server hello message from server: {'type': 'server_hello', 'content': {'version': 1.3, 'random': '72592055833086161281110318795648974609829', 'session_id': '6013168', 'compression_method': [], 'cipher_suite': {'key_exchg_algo': 'ECDH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'CMAC', 'cipher_type': 'Stream'}, 'specification_id': 2}, 'length': 368}
The client uses the ECDSA+SHA256 specification for Digital Signature
-----
```

Fig-7: client hello and server hello at client

```

-----
Received the client hello message from client: {'type': 'client_hello', 'content': {'version': 1.3, 'random': '704514256314735072489591610712892088858841', 'session_id': '6013168', 'cipher_suite': [{'key_exchg_algo': 'DH', 'cipher_algo': 'AES', 'mac_algo': 'HMAC', 'cipher_type': 'Block'}, {'key_exchg_algo': 'DH', 'cipher_algo': 'AES', 'mac_algo': 'CMAC', 'cipher_type': 'Block'}, {'key_exchg_algo': 'DH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'HMAC', 'cipher_type': 'Stream'}, {'key_exchg_algo': 'DH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'CMAC', 'cipher_type': 'Stream'}, {'key_exchg_algo': 'ECDH', 'cipher_algo': 'AES', 'mac_algo': 'HMAC', 'cipher_type': 'Block'}, {'key_exchg_algo': 'ECDH', 'cipher_algo': 'AES', 'mac_algo': 'CMAC', 'cipher_type': 'Block'}, {'key_exchg_algo': 'ECDH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'HMAC', 'cipher_type': 'Stream'}, {'key_exchg_algo': 'ECDH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'CMAC', 'cipher_type': 'Stream'}], 'compression_method': [], 'length': 240}}
Sent the server hello message to client: {'type': 'server_hello', 'content': {'version': 1.3, 'random': '725920558330861612811103187956484974609829', 'session_id': '6013168', 'compression_method': [], 'cipher_suite': {'key_exchg_algo': 'ECDH', 'cipher_algo': 'CHACHA20', 'mac_algo': 'CMAC', 'cipher_type': 'Stream'}, 'specification_id': 2}, 'length': 368}
The server uses the EC+SHA384 specification for Digital Signature
-----

```

Fig-8: client_hello and server_hello at server

5. `certificate_message` from server to client:

[illegible]

Fig-9: certificate_message at client

[illegible]

Fig-10: certificate_message at server

6. server_key_exchange from server to client:

```

catchabhihek@awsam: ~/2020CS22445-assignment-3
File Edit View Search Terminal Tabs Help
catchabhihek@awsam: ~/2020CS22445-assignment-3
catchabhihek@awsam: ~/2020CS22445-assignment-3
catchabhihek@awsam: ~/2020CS22445-assignment-3

Received the server key exchange message from server: {'type': 'server_key_exchange', 'content': {'parameters': {'ec_public_key': b'-----BEGIN PUBLIC KE
Y-----\nMHYEWAYhQZjzj0CAQYFK4EEACIDYGAEX/RHFJbFDYNkAbsXt42dpVN4N3Tm3J3I\|nXk|u5STCAC0Jr3VXpSDFIJZqaEfgqTCBBc45Jm3g0l7zwJ6B0TDCa6c1rL8h0o9|nHR+f1B0F5+deL
9WRkCk3A1znw1EUvp09l|-----END PUBLIC KEY-----\n'}, 'signature': b'0f|x021|x00|xc|xed|xb9|x1c1V|x2vWn|x86|x98yA|xdf-F|xc6|x1b|x170|xainj|xb7|x02|x82|x0b-
|x81|xc4|x97Zf.|x44tn|x18-|xc2|xa7|xf1|x9a|xbf5-|x021|x00|xa8|x02|x150|xd55|x08|x93-|x84r|xec|xfhf|x1|x1|xb7s|xda|x93|xbf|x7f|x13la|xb5|xb|x7e|n|xb
d|xcc|x8b|x82|x9c|xbx|j|x19w|xae|x8d|x99|x80|x01|xb2|x8a|xa5'}, 'length': 240}}
Verifying the Signature on message by using server public key
Verified the Signature on message with server public key

```

Fig-11: server_key_exchange at client

```

catcabbhisek@awsun: ~/2020CS22445-assignment-3
File Edit View Search Terminal Tabs Help
catcabbhisek@awsun: ~/2020CS22445-assignment-3
catcabbhisek@awsun: ~/2020CS22445-assignment-3
catcabbhisek@awsun: ~/2020CS22445-assignment-3

-----
Sent the server key exchange message to client: {'type': 'server_key_exchange', 'content': {'parameters': {'ec_public_key': b'-----BEGIN PUBLIC KEY-----
/nMHYwEAYHwQZiZj9CAQYFK4EEACIDYgAEX/RHFJbFDYNkAbsXi42dpVN4N3Tm33JIxnKX/uS5tCAC0Jr3VxPsdFIJZqaEfggTCBc45Jm3g0l7zwJ60BTDCa6cqlrL8hQ09/nHR+f180f5+del9wRwKc
3A1znw1EUbp0l-----END PUBLIC KEY-----\n'}, 'signature': b'0f0x121x00xcb\xed\xbb\x9c1v\x2d2Vwn\x86\x98yA\xdf-F\xcc6\x1b\x170\x1a1n\xbb7\x02\x82\x0b\x81\x
c4\x97Zf.\x4dt4n\x18-\xc2d\xa7\x1f\x9a\xbf5-\x1x021\x00\x8a\x82\x150\x5d5\x8a\x93_(\x84r\xec\xffh\x1a\x1\x1b7s\xda\x93\xbf7\x7f\x131a\x5b5\xbb\x7e7n\xbd\xcc\x
8b\x82\x9c\xbbX|\x19w\xae\x8d\x99\x80\x1d\x1b2\x8a\x5a5}', 'length': 240}

-----

```

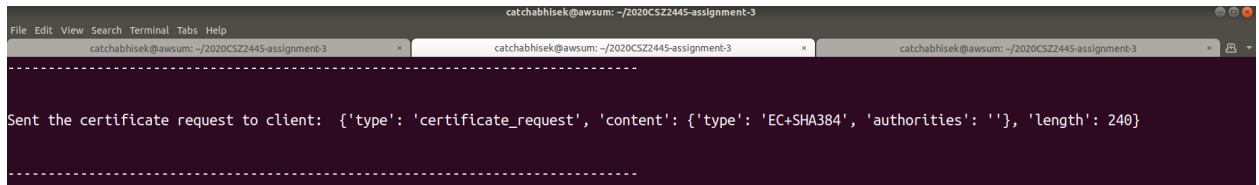
Fig-12: server key exchange at server

7. certificate_request from server to client:



```
catchabhi@awsum: ~/2020CS22445-assignment-3
Received the certificate request from the server: {'type': 'certificate_request', 'content': {'type': 'EC+SHA384', 'authorities': ''}, 'length': 240}
```

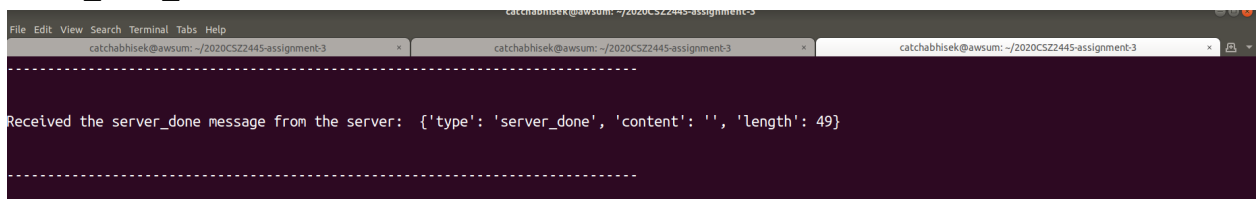
Fig-13: certificate_request at client



```
catchabhi@awsum: ~/2020CS22445-assignment-3
Sent the certificate request to client: {'type': 'certificate_request', 'content': {'type': 'EC+SHA384', 'authorities': ''}, 'length': 240}
```

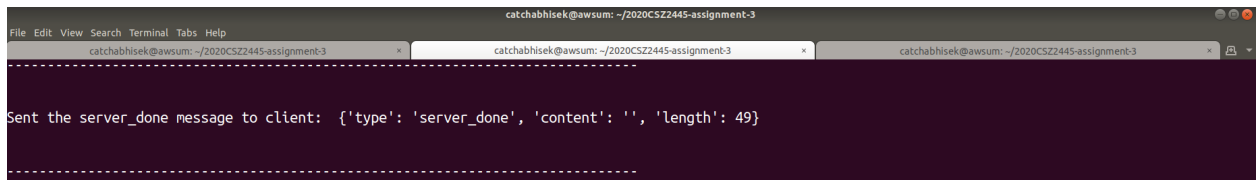
Fig-14: certificate_request at server

8. server_hello_done from server to client:



```
catchabhi@awsum: ~/2020CS22445-assignment-3
Received the server_done message from the server: {'type': 'server_done', 'content': '', 'length': 49}
```

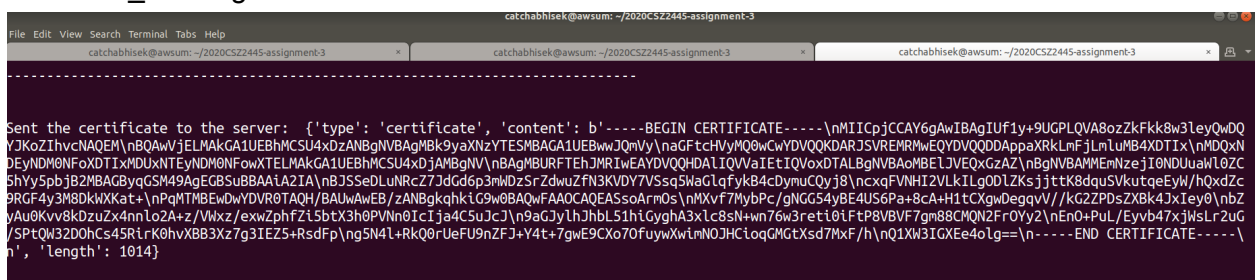
Fig-15: server_hello_done at client



```
catchabhi@awsum: ~/2020CS22445-assignment-3
Sent the server_done message to client: {'type': 'server_done', 'content': '', 'length': 49}
```

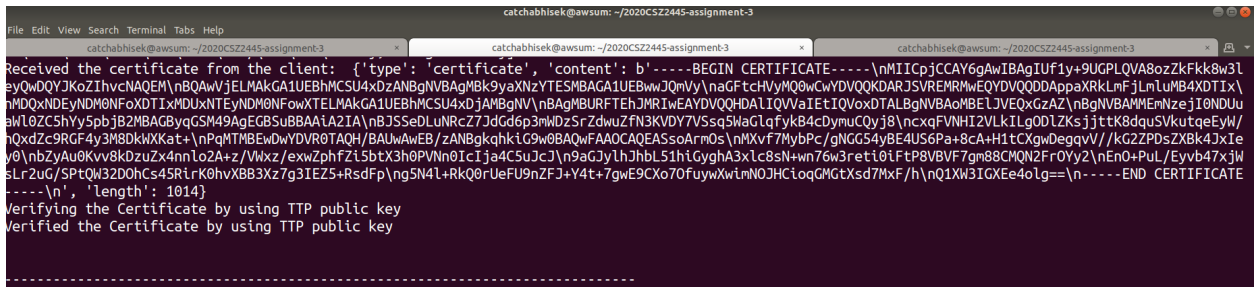
Fig-16: server_hello_done at server

9. certificate_message from client to server:

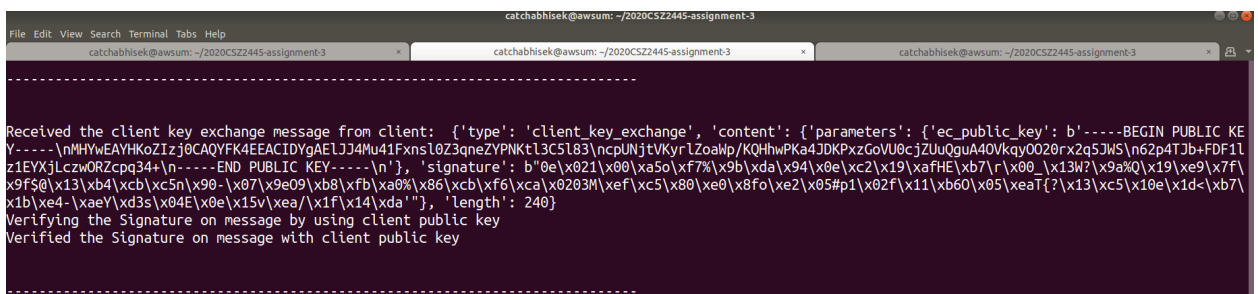
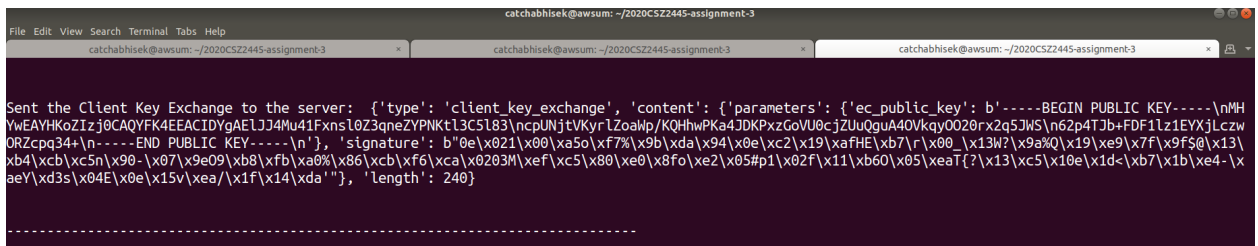


```
catchabhi@awsum: ~/2020CS22445-assignment-3
Sent the certificate to the server: {'type': 'certificate', 'content': 'b'-----BEGIN CERTIFICATE-----\nMIIChjCCAY6gAwIBAgIUf1y+9UGPLQVA8ozZkFkk8w31eyQwDQYJKoZIhvcNAQEM\n\\nBQAwVjELMAkGA1UEBhMCU4xDzANBgNVBAMt9yaXNzYTESMBAG1UEBwwJQmVl\n\\naGFtchVtY0w0cWcwYDVoQKdARjSVREMRmEQYDVQDDAppaXRRkLmFjLnLlUMB4XDIT\n\\nMDQxNDYNDM0NFoXDTIeMDUxNTEyNDM0NFowXTELMakGA1UEBhMCU4xDzANBgNV\n\\nBAGMBURFTEhJMRIwEAYDVQQHDAlIQVNa1EtIQVoxDTALBgNVBAoMBE1JVEQxGzAZ\n\\nBgNVBAMMENzeji0NDUuawT0ZC5hYy5pbjB2MBAGByqGSM49AgEGSsBBAAIA2IA\\nBJSSeDLuNRcZ7JdGd6p3mWDzSrZdwuZFN3KVDY7V5sq5WaGlfykB4cDymuQyJ8\\nncxqFVNHI2VLkILg0DLZKsjttK8dquSVkutqeEYw/hQxdZc9RGF4y3M8DkwxKat+\\n\nPqMTMBEwdwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQwFAAOCAQEAASoArm0s\\n\nMXvf7MybPc/gNGG54yBE4US6Pa+8cA+H1tCXgwDegqvV//KG2ZPDsZX8k4JxIey0\\n\nbZyAu0Kvv8kDzuZx4nnLo2A+z/Vwxz/exwZphFZi5btX3h0PVNn0IcIja4C5uJcJ\\n\n9aGJy/LhJhbl51hGyghA3xlc8sN+wn76w3reti0iFtP8VBVF7gm88CQMN2Fr0Yy2\\n\nEn0+PuL/Eyvb47xjWslr2uG/SPTQW32DohCs45RirK0hvxBB3Xz7g3IEZ5+RsdFp\\nng5N4L+RkQ0rUeFU9nZFJ+Y4t+7gwE9CXo70FuywXwlmNOJHCioqQMgtXsd7Mx\n\\nh', 'length': 1014}
```

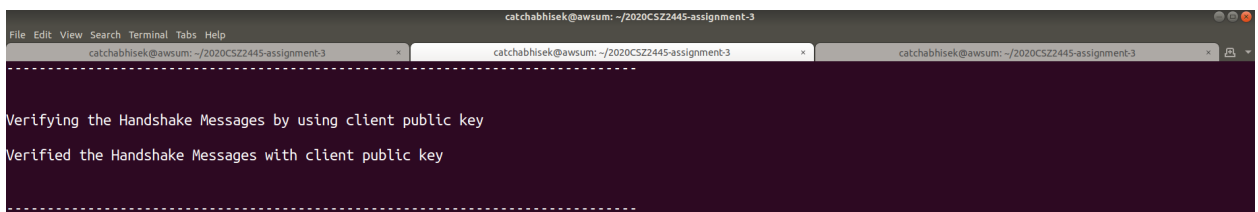
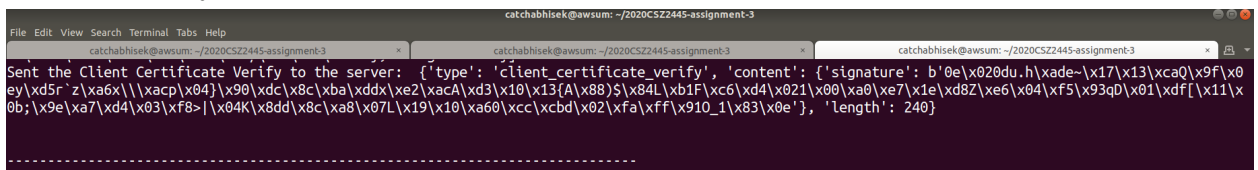
Fig-17: certificate_message at client



10. client_key_exchange from client to server:



11. certificate_verify from client to server:



12. Crypto Parameters at Server and Client:

```
catchabhihek@awsun: ~/2020CS22445-assignment-3
File Edit View Search Terminal Tabs Help
catchabhihek@awsun: ~/2020CS22445-assignment-3
-----
Generating the cryptographic parameters from the pre_master_key
-----
pre_master_key b'Ft\xfb3\x86fV\x0f\xd7a\xd1\xfb\xa0=Ml\xab\xbe!NA_C\x01\xa0\xed\x83\x9dV\xbc\x91\xe6'
master_key b'\x92[\x13\xca\xb8!\x02\x1f\x15\x8a\x01\xd2.,7\x16h\xe2\x8e\x834\xce\x1c\xd13\xdbb\xd2i\x8c\x08-'
mac_secret b'\\\xbb\x97\xe4qy\xcb\xe4K=\x04oJ\x00\x1a\x19\xafuA\xd9\xe7\x0f\x0f\x85\xd49e\xff\xb2\x87-\x0b'
cipher_nonce b'\xf0\x1a\x84i\x07\xa4\xbc\x08Z1\xff6\xe9aL\x0b'
-----
```

Fig-23: cryptography parameter at client

```
catchabhihek@awsun: ~/2020CS22445-assignment-3
File Edit View Search Terminal Tabs Help
catchabhihek@awsun: ~/2020CS22445-assignment-3
-----
Generating the cryptographic parameters from the pre_master_key
-----
pre_master_key b'Ft\xfb3\x86fV\x0f\xd7a\xd1\xfb\xa0=Ml\xab\xbe!NA_C\x01\xa0\xed\x83\x9dV\xbc\x91\xe6'
master_key b'\x92[\x13\xca\xb8!\x02\x1f\x15\x8a\x01\xd2.,7\x16h\xe2\x8e\x834\xce\x1c\xd13\xdbb\xd2i\x8c\x08-'
mac_secret b'\\\xbb\x97\xe4qy\xcb\xe4K=\x04oJ\x00\x1a\x19\xafuA\xd9\xe7\x0f\x0f\x85\xd49e\xff\xb2\x87-\x0b'
cipher_nonce b'\xf0\x1a\x84i\x07\xa4\xbc\x08Z1\xff6\xe9aL\x0b'
-----
```

Fig-24: cryptography parameter at server

13. OTP_Request:

```
catchabhihek@awsun: ~/2020CS22445-assignment-3
File Edit View Search Terminal Tabs Help
catchabhihek@awsun: ~/2020CS22445-assignment-3
-----
Handshake Protocol completed with the Server
-----
Framing the request for OTP from the server
Applying the record protocol for sending the message: b'The OTP for transferring Rs 838505 to your friend\xe2\x80\x99s account'
-----
Generating the MAC of the message using CMAC algorithm
The MAC generated from the CMAC MAC algorithm is: b'\xf1\xec\xd5=\xed(\x9c\xcf\x1d\xba\xed\x01W\xeab\x83'
Concatenating the message and mac for encryption: b'The OTP for transferring Rs 838505 to your friend\xe2\x80\x99s account\xf1\xec\xd5=\xed(\x9c\xcf\x1d\xba\xed\x01W\xeab\x83'
-----
Generating the Encryption of the message using CHACHA20 algorithm
The Encrypted Message generated from the CHACHA20 encryption algorithm is: b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\x06\xb1\xb0\x92\x8b\x03\x97\xad\x08mN\x1a\nmG\xd2#\xfax87P5\xf2K-\xf0f\xef\x07J\x15S\x16*y%\xf1-#\xf6\x806\xa4\x1c\x93M\xa7*\x15]=\xd2\x88\x8e\x85\xc93*\xc4'
The request for OTP sent to server is {'message': b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\x06\xb1\xb0\x92\x8b\x03\x97\xad\x08mN\x1a\nmG\xd2#\xfax87P5\xf2K-\xf0f\xef\x07J\x15S\x16*y%\xf1-#\xf6\x806\xa4\x1c\x93M\xa7*\x15]=\xd2\x88\x8e\x85\xc93*\xc4', 'length': 61, 'size': 110, 'type': 'generate_otp'}
```

Fig-25: OTP_Request at client

```
catchabhisek@awsun: ~/2020CS22445-assignment-3
-----
The request for OTP sent received from client is {'message': b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\xc6\xb1\xb0\x92\x8b\xc3\x97\xad\xe8mN\x1a\nmG\xd2#\xfax87P5\xf2K<^\x0f\xef\xc7\x07J\x15S\x16* y%\xf1-#\xf6\x806\xa4\x1c\x93M\xa7*\x15]=\xd2\x88\x8e\x85\xc93*\xc4', 'length': 61, 'size': 110, 'type': 'generate_otp'}
-----
Applying the record protocol for receiving the message: b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\xc6\xb1\xb0\x92\x8b\xc3\x97\xad\xe8mN\x1a\nmG\xd2#\xfax87P5\xf2K<^\x0f\xef\xc7\x07J\x15S\x16* y%\xf1-#\xf6\x806\xa4\x1c\x93M\xa7*\x15]=\xd2\x88\x8e\x85\xc93*\xc4'
-----
Generating the Decryption of the message using CHACHA20 algorithm
The Decrypted Message generated from the CHACHA20 encryption algorithm is: b'The OTP for transferring Rs 838505 to your friend\x02\x80\x99s account\xf1\xec\xd5=\xed(\x9c\xcf\x1d\xba\xed\x1W\xeab\x83'
-----
The Message Extracted is: b'The OTP for transferring Rs 838505 to your friend\x02\x80\x99s account'
The MAC received is: b'\xf1\xec\xd5=\xed(\x9c\xcf\x1d\xba\xed\x1W\xeab\x83'
-----
Verifying the MAC of the message using CMAC algorithm
Successfully verified the MAC of the message using CMAC algorithm
The plain text version of request received by server: The OTP for transferring Rs 838505 to your friend's account
-----
```

Fig-26: OTP_Request at server

14. OTP_Reply:

```
-----
The encrypted reply received from the server: {'message': b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\xc6\xb1\xb0\x92\x8b\xc3\x97\xad\xe8mN\x1a\nmG\xd2#\xfax87P5\xf2K<^\x0f\xef\xc7\x07J\x15S\x16* y%\xf1-#\xf6\x806\xa4\xcd\x16\xeb\xba\xfd\x1d\xf9\xcb\xf6\nP}\x94\xc6\xbaAMs\x1a\xa2\xe3\x19\xdc\xf5\xc8\xf8\xeft5', 'length': 74, 'size': 123}
-----
Applying the record protocol for receiving the message: b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\xc6\xb1\xb0\x92\x8b\xc3\x97\xad\xe8mN\x1a\nmG\xd2#\xfax87P5\xf2K<^\x0f\xef\xc7\x07J\x15S\x16* y%\xf1-#\xf6\x806\xa4\xcd\x16\xeb\xba\xfd\x1d\xf9\xcb\xf6\nP}\x94\xc6\xbaAMs\x1a\xa2\xe3\x19\xdc\xf5\xc8\xf8\xeft5'
-----
Generating the Decryption of the message using CHACHA20 algorithm
The Decrypted Message generated from the CHACHA20 encryption algorithm is: b'The OTP for transferring Rs 838505 to your friend\x02\x80\x99s account is : 899839\n\x1f\xf2\x06\xe1-xas\x87\xf5\xb7\x0c\xaa@\xe21'
-----
The Message Extracted is: b'The OTP for transferring Rs 838505 to your friend\x02\x80\x99s account is : 899839\n'
The MAC received is: b'\x1f\xf2\x06\xe1-xas\x87\xf5\xb7\x0c\xaa@\xe21'
-----
Verifying the MAC of the message using CMAC algorithm
Successfully verified the MAC of the message using CMAC algorithm
The plain text version of reply received from the server: The OTP for transferring Rs 838505 to your friend's account is : 899839
-----
```

Fig-27: OTP_Reply at client

```
-----
The plain text version of reply framed by server: The OTP for transferring Rs 838505 to your friend's account is : 899839
-----
Applying the record protocol for sending the message: b'The OTP for transferring Rs 838505 to your friend\x02\x80\x99s account is : 899839\n'
-----
Generating the MAC of the message using CMAC algorithm
The MAC generated from the CMAC MAC algorithm is: b'\x1f\xf2\x06\xe1-xas\x87\xf5\xb7\x0c\xaa@\xe21'
-----
Concatenating the message and mac for encryption: b'The OTP for transferring Rs 838505 to your friend\x02\x80\x99s account is : 899839\n\x1f\xf2\x06\xe1-xas\x87\xf5\xb7\x0c\xaa@\xe21'
-----
Generating the Encryption of the message using CHACHA20 algorithm
The Encrypted Message generated from the CHACHA20 encryption algorithm is: b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\xc6\xb1\xb0\x92\x8b\xc3\x97\xad\xe8mN\x1a\nmG\xd2#\xfax87P5\xf2K<^\x0f\xef\xc7\x07J\x15S\x16* y%\xf1-#\xf6\x806\xa4\xcd\x16\xeb\xba\xfd\x1d\xf9\xcb\xf6\nP}\x94\xc6\xbaAMs\x1a\xa2\xe3\x19\xdc\xf5\xc8\xf8\xeft5'
-----
The encrypted reply sent to client: {'message': b'u\xac\x8ark%\x07\xe8\x07q}\xe3}\x1eC\xc6\xb1\xb0\x92\x8b\xc3\x97\xad\xe8mN\x1a\nmG\xd2#\xfax87P5\xf2K<^\x0f\xef\xc7\x07J\x15S\x16* y%\xf1-#\xf6\x806\xa4\xcd\x16\xeb\xba\xfd\x1d\xf9\xcb\xf6\nP}\x94\xc6\xbaAMs\x1a\xa2\xe3\x19\xdc\xf5\xc8\xf8\xeft5', 'length': 74, 'size': 123}
-----
```

Fig-28: OTP_Reply at server