Assignment 6: Secure chat using openssl and MITM attacks

Student 1: Vishesh Kothari (CS22MTECH12004)
Student 2: K Saravanan (CS22MTECH12007)

Task 1: Generate keys and certificates

- Key Generation for RootCA, InterCA (Intermediate CA), Alice & Bob:

1. To generate the key for RootCA, the following command was used:

```
openssl ecparam -name brainpoolP512r1 -genkey -noout -out
root-private-key.pem
```

2. To generate the key for IntermediateCA, the following command was used:

```
openssl genpkey -out Inter-private-key.pem -algorithm RSA -pkeyopt
rsa_keygen_bits:2048
```

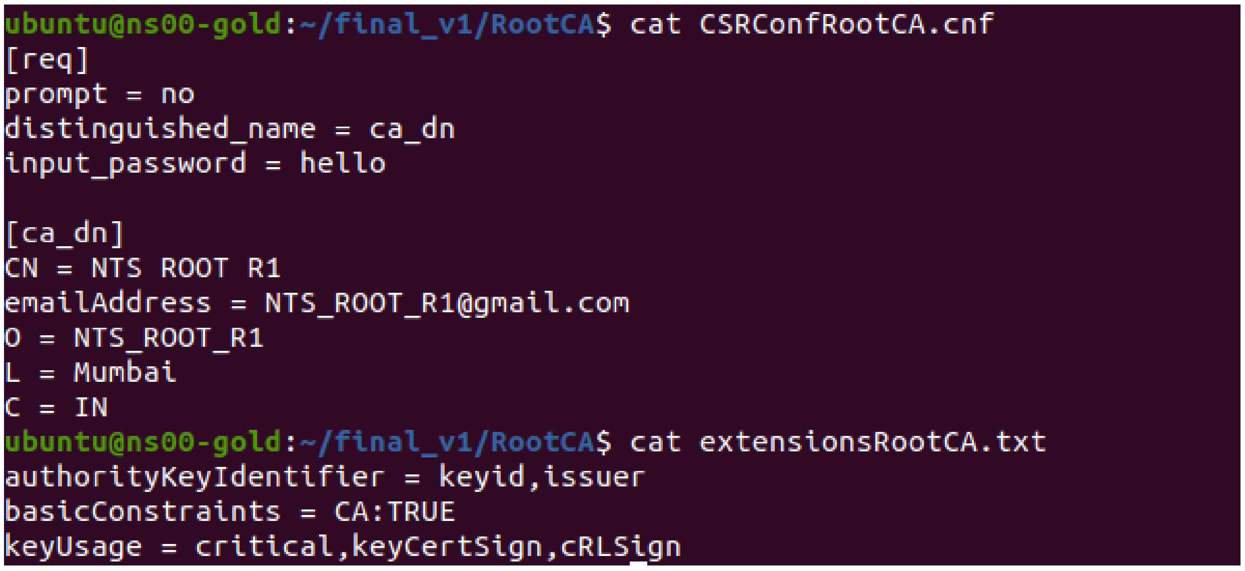
3. To generate the key for Alice, the following command was used:

```
openssl genpkey -out Alice.pem -algorithm RSA -pkeyopt rsa_keygen_bits:2048
```

4. To generate the key for Bob, the following command was used:

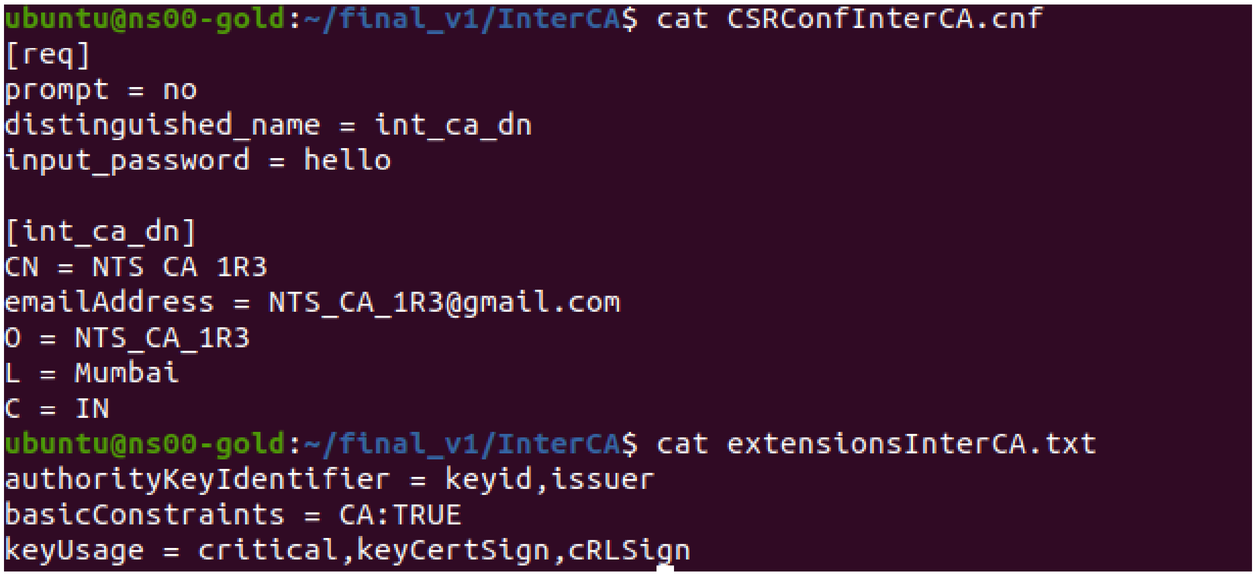
```
openssl ecparam -name prime256v1 -genkey -noout -out Bob.pem
```

Please find the Screenshots for the **CSR & Certificate extension Conf** files which were used to generate the Certificates and CSR's for the RootCA, IntermediateCA, Alice and Bob.
RootCA:



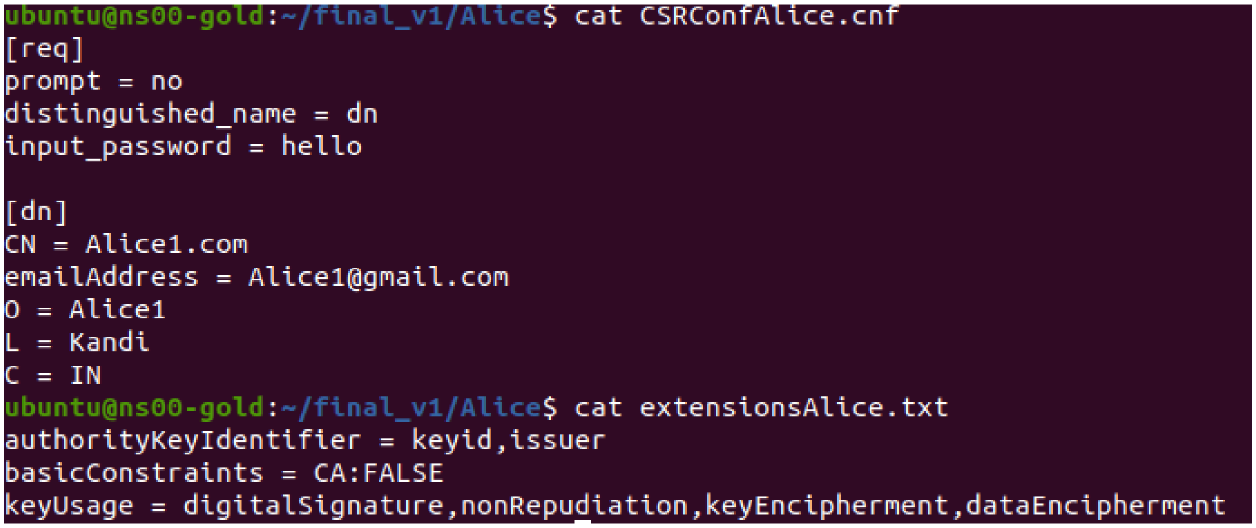
```
ubuntu@ns00-gold:~/final_v1/RootCA$ cat CSRConfRootCA.cnf
[req]
prompt = no
distinguished_name = ca_dn
input_password = hello

[ca_dn]
CN = NTS ROOT R1
emailAddress = NTS_ROOT_R1@gmail.com
O = NTS_ROOT_R1
L = Mumbai
C = IN
ubuntu@ns00-gold:~/final_v1/RootCA$ cat extensionsRootCA.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
```

Intermediate CA:



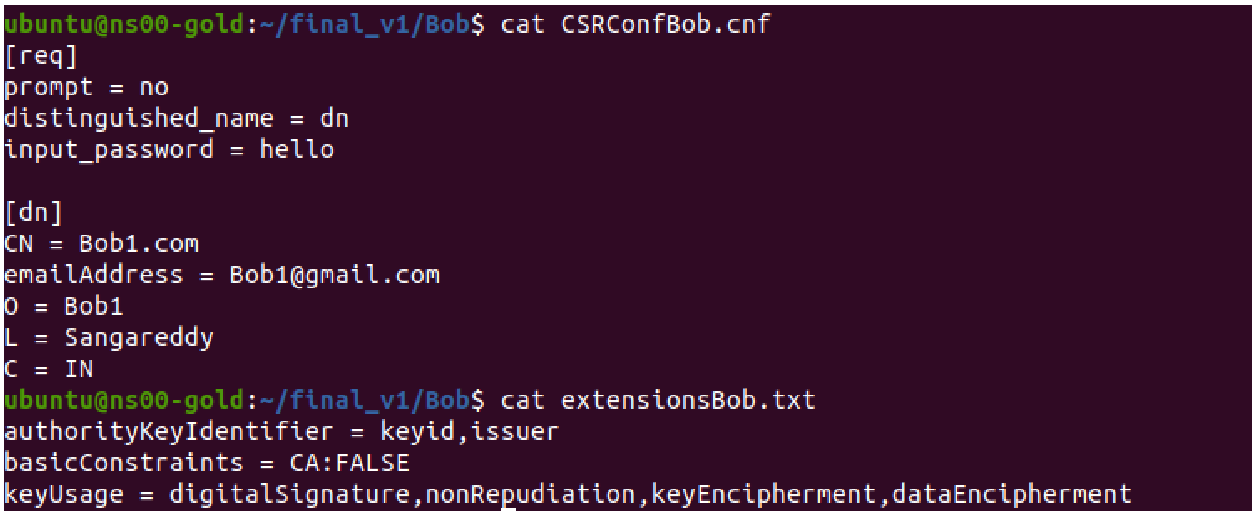
```
ubuntu@ns00-gold:~/final_v1/InterCA$ cat CSRConfInterCA.cnf
[req]
prompt = no
distinguished_name = int_ca_dn
input_password = hello

[int_ca_dn]
CN = NTS CA 1R3
emailAddress = NTS_CA_1R3@gmail.com
O = NTS_CA_1R3
L = Mumbai
C = IN
ubuntu@ns00-gold:~/final_v1/InterCA$ cat extensionsInterCA.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
```

For Alice:



```
ubuntu@ns00-gold:~/final_v1/Alice$ cat CSRConfAlice.cnf
[req]
prompt = no
distinguished_name = dn
input_password = hello

[dn]
CN = Alice1.com
emailAddress = Alice1@gmail.com
O = Alice1
L = Kandi
C = IN
ubuntu@ns00-gold:~/final_v1/Alice$ cat extensionsAlice.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:FALSE
keyUsage = digitalSignature,nonRepudiation,keyEncipherment,dataEncipherment
```

For Bob:



```
ubuntu@ns00-gold:~/final_v1/Bob$ cat CSRConfBob.cnf
[req]
prompt = no
distinguished_name = dn
input_password = hello

[dn]
CN = Bob1.com
emailAddress = Bob1@gmail.com
O = Bob1
L = Sangareddy
C = IN
ubuntu@ns00-gold:~/final_v1/Bob$ cat extensionsBob.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:FALSE
keyUsage = digitalSignature,nonRepudiation,keyEncipherment,dataEncipherment
```

- CSR Generation for RootCA, InterCA, Alice & Bob:

RootCA:

```
openssl req -new -config CSRConfRootCA.cnf -key root-private-key.pem -out
RootCA.csr
```

IntermediateCA:

```
openssl req -new -config CSRConfInterCA.cnf -key Inter-private-key.pem -out
InterCA.csr
```

Alice:

```
openssl req -new -config CSRConfAliceCA.cnf -key Alice.pem -out Alice.csr
```

Bob:

```
openssl req -new -config CSRConfBobCA.cnf -key Bob.pem -out Bob.csr
```

- Certificate Generation for RootCA, InterCA, Alice & Bob:

RootCA(<u>Creating a Self Signed X509v3 certificate</u>):

```
openssl x509 -req -days 365 -in RootCA.csr -signkey root-private-key.pem
-out rootCA.crt -extfile extensionsRootCA.txt
```

IntermediateCA:

```
openssl x509 -req -in InterCA.csr -CA ../RootCA/rootCA.crt -extfile
extensionsInterCA.txt -CAkey ../RootCA/root-private-key.pem -CAcreateserial
-out InterCA.crt -days 365 -sha256
```

Alice:

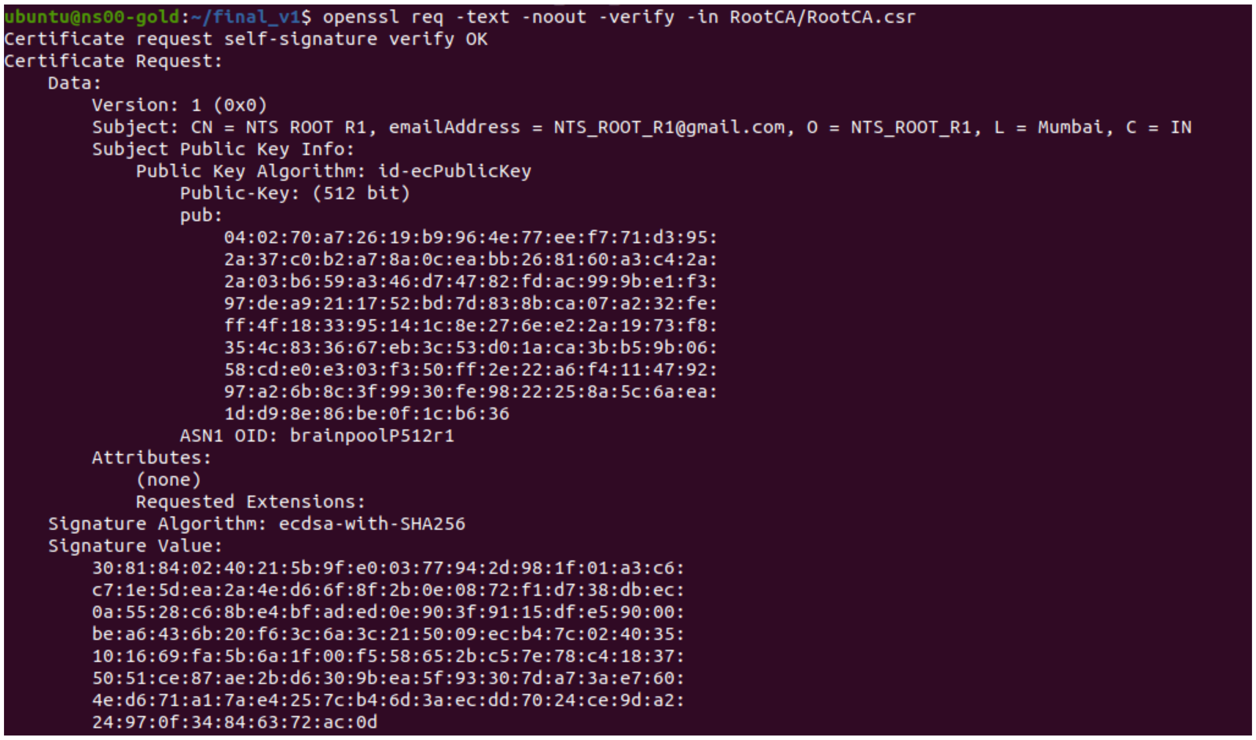
```
openssl x509 -req -in Alice.csr -CA ../Inter/InterCA.crt -extfile
extensionsAlice.txt -CAkey ../InterCA/Inter-private-key.pem -CAcreateserial
-out Alice.crt -days 365 -sha256
```

Bob:

```
openssl x509 -req -in Bob.csr -CA ../InterCA/InterCA.crt -extfile Bob.txt
-CAkey ../InterCA/Inter-private-key.pem -CAcreateserial -out Bob.crt -days
365 -sha256
```

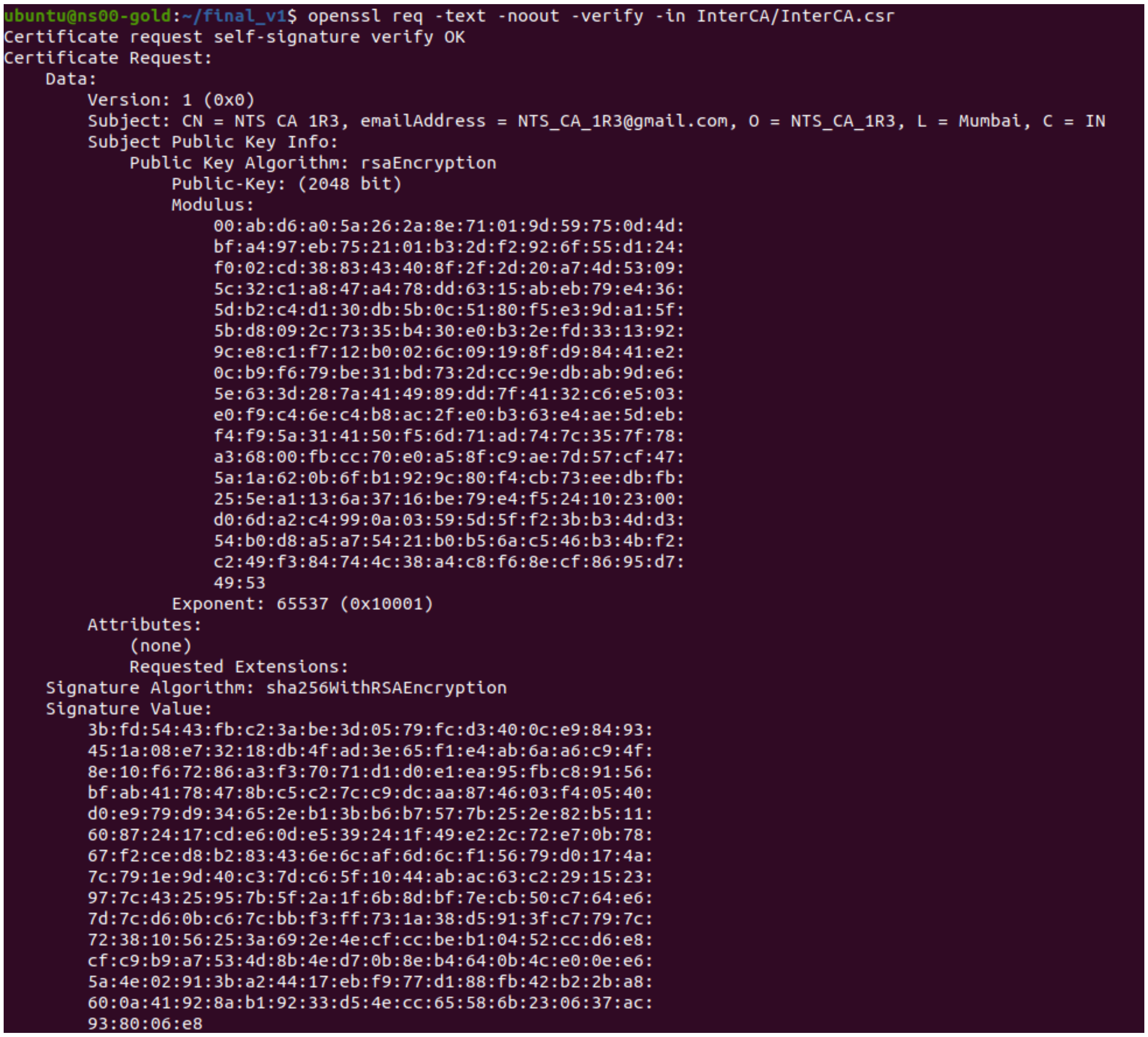
Certificate Authentication and Integrity Check for RootCA, InterCA, Alice & Bob:
Using the following commands for verifying the **CSR for RootCA, InterCA, Alice and Bob**:

```
openssl req -text -noout -verify -in RootCA.csr
```



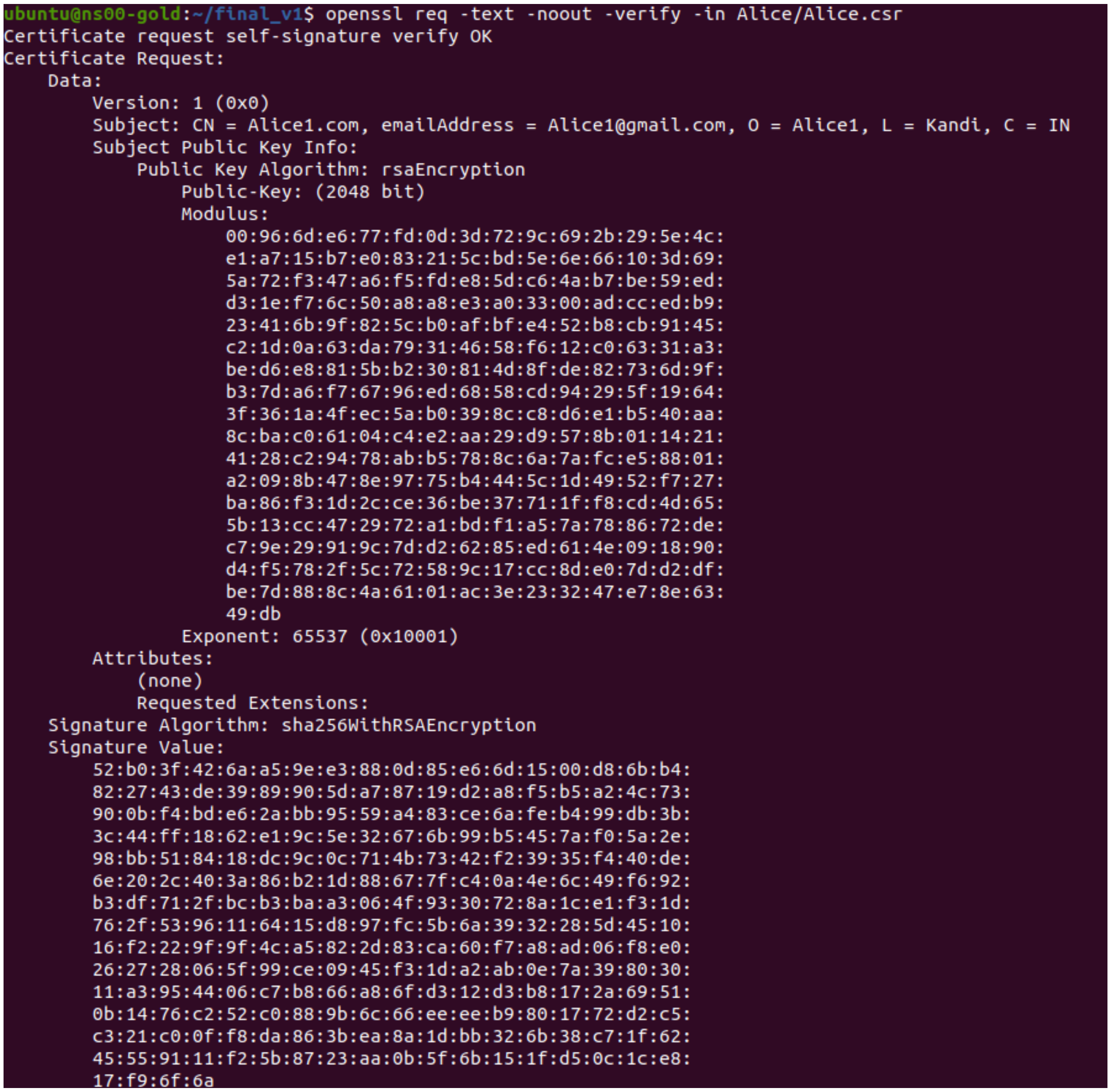
```
ubuntu@ns00-gold:~/final_v1$ openssl req -text -noout -verify -in RootCA/RootCA.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = NTS ROOT R1, emailAddress = NTS_ROOT_R1@gmail.com, O = NTS_ROOT_R1, L = Mumbai, C = IN
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:02:70:a7:26:19:b9:96:4e:77:ee:f7:71:d3:95:
                    2a:37:c0:b2:a7:8a:0c:ea:bb:26:81:60:a3:c4:2a:
                    2a:03:b6:59:a3:46:d7:47:82:fd:ac:99:9b:e1:f3:
                    97:de:a9:21:17:52:bd:7d:83:8b:ca:07:a2:32:fe:
                    ff:4f:18:33:95:14:1c:8e:27:6e:e2:2a:19:73:f8:
                    35:4c:83:36:67:eb:3c:53:d0:1a:ca:3b:b5:9b:06:
                    58:cd:e0:e3:03:f3:50:ff:2e:22:a6:f4:11:47:92:
                    97:a2:6b:8c:3f:99:30:fe:98:22:25:8a:5c:6a:ea:
                    1d:d9:8e:86:be:0f:1c:b6:36
                ASN1 OID: brainpoolP512r1
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:81:84:02:40:21:5b:9f:e0:03:77:94:2d:98:1f:01:a3:c6:
        c7:1e:5d:ea:2a:4e:d6:6f:8f:2b:0e:08:72:f1:d7:38:db:ec:
        0a:55:28:c6:8b:e4:bf:ad:ed:0e:90:3f:91:15:df:e5:90:00:
        be:a6:43:6b:20:f6:3c:6a:3c:21:50:09:ec:b4:7c:02:40:35:
        10:16:69:fa:5b:6a:1f:00:f5:58:65:2b:c5:7e:78:c4:18:37:
        50:51:ce:87:ae:2b:d6:30:9b:ea:5f:93:30:7d:a7:3a:e7:60:
        4e:d6:71:a1:7a:e4:25:7c:b4:6d:3a:ec:dd:70:24:ce:9d:a2:
        24:97:0f:34:84:63:72:ac:0d
```

```
openssl req -text -noout -verify -in InterCA.csr
```



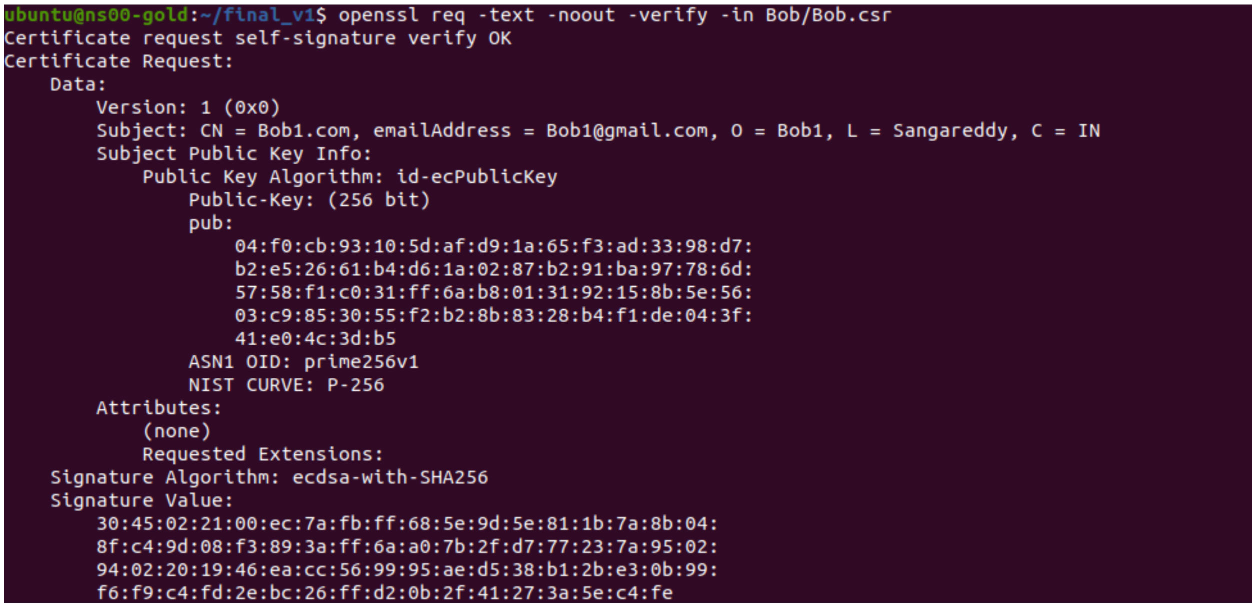
```
ubuntu@ns00-gold:~/final_v1$ openssl req -text -noout -verify -in InterCA/InterCA.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = NTS CA 1R3, emailAddress = NTS_CA_1R3@gmail.com, O = NTS_CA_1R3, L = Mumbai, C = IN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ab:d6:a0:5a:26:2a:8e:71:01:9d:59:75:0d:4d:
                    bf:a4:97:eb:75:21:01:b3:2d:f2:92:6f:55:d1:24:
                    f0:02:cd:38:83:43:40:8f:2f:2d:20:a7:4d:53:09:
                    5c:32:c1:a8:47:a4:78:dd:63:15:ab:eb:79:e4:36:
                    5d:b2:c4:d1:30:db:5b:0c:51:80:f5:e3:9d:a1:5f:
                    5b:d8:09:2c:73:35:b4:30:e0:b3:2e:fd:33:13:92:
                    9c:e8:c1:f7:12:b0:02:6c:09:19:8f:d9:84:41:e2:
                    0c:b9:f6:79:be:31:bd:73:2d:cc:9e:db:ab:9d:e6:
                    5e:63:3d:28:7a:41:49:89:dd:7f:41:32:c6:e5:03:
                    e0:f9:c4:6e:c4:b8:ac:2f:e0:b3:63:e4:ae:5d:eb:
                    f4:f9:5a:31:41:50:f5:6d:71:ad:74:7c:35:7f:78:
                    a3:68:00:fb:cc:70:e0:a5:8f:c9:ae:7d:57:cf:47:
                    5a:1a:62:0b:6f:b1:92:9c:80:f4:cb:73:ee:db:fb:
                    25:5e:a1:13:6a:37:16:be:79:e4:f5:24:10:23:00:
                    d0:6d:a2:c4:99:0a:03:59:5d:5f:f2:3b:b3:4d:d3:
                    54:b0:d8:a5:a7:54:21:b0:b5:6a:c5:46:b3:4b:f2:
                    c2:49:f3:84:74:4c:38:a4:c8:f6:8e:cf:86:95:d7:
                    49:53
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        3b:fd:54:43:fb:c2:3a:be:3d:05:79:fc:d3:40:0c:e9:84:93:
        45:1a:08:e7:32:18:db:4f:ad:3e:65:f1:e4:ab:6a:a6:c9:4f:
        8e:10:f6:72:86:a3:f3:70:71:d1:d0:e1:ea:95:fb:c8:91:56:
        bf:ab:41:78:47:8b:c5:c2:7c:c9:dc:aa:87:46:03:f4:05:40:
        d0:e9:79:d9:34:65:2e:b1:3b:b6:b7:57:7b:25:2e:82:b5:11:
        60:87:24:17:cd:e6:0d:e5:39:24:1f:49:e2:2c:72:e7:0b:78:
        67:f2:ce:d8:b2:83:43:6e:6c:af:6d:6c:f1:56:79:d0:17:4a:
        7c:79:1e:9d:40:c3:7d:c6:5f:10:44:ab:ac:63:c2:29:15:23:
        97:7c:43:25:95:7b:5f:2a:1f:6b:8d:bf:7e:cb:50:c7:64:e6:
        7d:7c:d6:0b:c6:7c:bb:f3:ff:73:1a:38:d5:91:3f:c7:79:7c:
        72:38:10:56:25:3a:69:2e:4e:cf:cc:be:b1:04:52:cc:d6:e8:
        cf:c9:b9:a7:53:4d:8b:4e:d7:0b:8e:b4:64:0b:4c:e0:0e:e6:
        5a:4e:02:91:3b:a2:44:17:eb:f9:77:d1:88:fb:42:b2:2b:a8:
        60:0a:41:92:8a:b1:92:33:d5:4e:cc:65:58:6b:23:06:37:ac:
        93:80:06:e8
```

```
openssl req -text -noout -verify -in Alice.csr
```



```
ubuntu@ns00-gold:~/final_v1$ openssl req -text -noout -verify -in Alice/Alice.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = Alice1.com, emailAddress = Alice1@gmail.com, O = Alice1, L = Kandi, C = IN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:96:6d:e6:77:fd:0d:3d:72:9c:69:2b:29:5e:4c:
                    e1:a7:15:b7:e0:83:21:5c:bd:5e:6e:66:10:3d:69:
                    5a:72:f3:47:a6:f5:fd:e8:5d:c6:4a:b7:be:59:ed:
                    d3:1e:f7:6c:50:a8:a8:e3:a0:33:00:ad:cc:ed:b9:
                    23:41:6b:9f:82:5c:b0:af:bf:e4:52:b8:cb:91:45:
                    c2:1d:0a:63:da:79:31:46:58:f6:12:c0:63:31:a3:
                    be:d6:e8:81:5b:b2:30:81:4d:8f:de:82:73:6d:9f:
                    b3:7d:a6:f7:67:96:ed:68:58:cd:94:29:5f:19:64:
                    3f:36:1a:4f:ec:5a:b0:39:8c:c8:d6:e1:b5:40:aa:
                    8c:ba:c0:61:04:c4:e2:aa:29:d9:57:8b:01:14:21:
                    41:28:c2:94:78:ab:b5:78:8c:6a:7a:fc:e5:88:01:
                    a2:09:8b:47:8e:97:75:b4:44:5c:1d:49:52:f7:27:
                    ba:86:f3:1d:2c:ce:36:be:37:71:1f:f8:cd:4d:65:
                    5b:13:cc:47:29:72:a1:bd:f1:a5:7a:78:86:72:de:
                    c7:9e:29:91:9c:7d:d2:62:85:ed:61:4e:09:18:90:
                    d4:f5:78:2f:5c:72:58:9c:17:cc:8d:e0:7d:d2:df:
                    be:7d:88:8c:4a:61:01:ac:3e:23:32:47:e7:8e:63:
                    49:db
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        52:b0:3f:42:6a:a5:9e:e3:88:0d:85:e6:6d:15:00:d8:6b:b4:
        82:27:43:de:39:89:90:5d:a7:87:19:d2:a8:f5:b5:a2:4c:73:
        90:0b:f4:bd:e6:2a:bb:95:59:a4:83:ce:6a:fe:b4:99:db:3b:
        3c:44:ff:18:62:e1:9c:5e:32:67:6b:99:b5:45:7a:f0:5a:2e:
        98:bb:51:84:18:dc:9c:0c:71:4b:73:42:f2:39:35:f4:40:de:
        6e:20:2c:40:3a:86:b2:1d:88:67:7f:c4:0a:4e:6c:49:f6:92:
        b3:df:71:2f:bc:b3:ba:a3:06:4f:93:30:72:8a:1c:e1:f3:1d:
        76:2f:53:96:11:64:15:d8:97:fc:5b:6a:39:32:28:5d:45:10:
        16:f2:22:9f:9f:4c:a5:82:2d:83:ca:60:f7:a8:ad:06:f8:e0:
        26:27:28:06:5f:99:ce:09:45:f3:1d:a2:ab:0e:7a:39:80:30:
        11:a3:95:44:06:c7:b8:66:a8:6f:d3:12:d3:b8:17:2a:69:51:
        0b:14:76:c2:52:c0:88:9b:6c:66:ee:ee:b9:80:17:72:d2:c5:
        c3:21:c0:0f:f8:da:86:3b:ea:8a:1d:bb:32:6b:38:c7:1f:62:
        45:55:91:11:f2:5b:87:23:aa:0b:5f:6b:15:1f:d5:0c:1c:e8:
        17:f9:6f:6a
```

```
openssl req -text -noout -verify -in Bob.csr
```



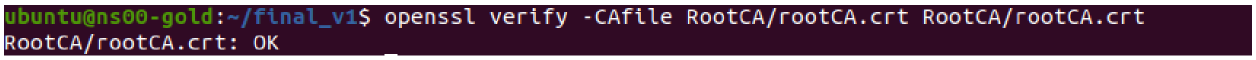
```
ubuntu@ns00-gold:~/final_v1$ openssl req -text -noout -verify -in Bob/Bob.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = Bob1.com, emailAddress = Bob1@gmail.com, O = Bob1, L = Sangareddy, C = IN
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:f0:cb:93:10:5d:af:d9:1a:65:f3:ad:33:98:d7:
                    b2:e5:26:61:b4:d6:1a:02:87:b2:91:ba:97:78:6d:
                    57:58:f1:c0:31:ff:6a:b8:01:31:92:15:8b:5e:56:
                    03:c9:85:30:55:f2:b2:8b:83:28:b4:f1:de:04:3f:
                    41:e0:4c:3d:b5
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:ec:7a:fb:ff:68:5e:9d:5e:81:1b:7a:8b:04:
        8f:c4:9d:08:f3:89:3a:ff:6a:a0:7b:2f:d7:77:23:7a:95:02:
        94:02:20:19:46:ea:cc:56:99:95:ae:d5:38:b1:2b:e3:0b:99:
        f6:f9:c4:fd:2e:bc:26:ff:d2:0b:2f:41:27:3a:5e:c4:fe
```

Using the following commands for verifying the **Certificate for RootCA, InterCA, Alice and Bob**:

RootCA:

```
openssl verify -CAfile RootCA/rootCA.crt RootCA/rootCA.crt
```

Screenshot of the same:

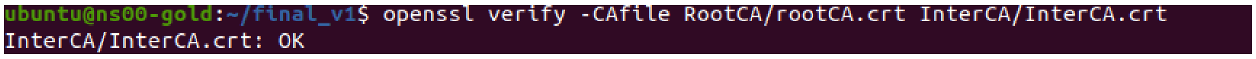


```
ubuntu@ns00-gold:~/final_v1$ openssl verify -CAfile RootCA/rootCA.crt RootCA/rootCA.crt
RootCA/rootCA.crt: OK
```

IntermediateCA:

```
openssl verify -CAfile RootCA/rootCA.crt InterCA/InterCA.crt
```

Screenshot of the same:

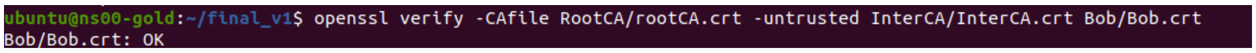


```
ubuntu@ns00-gold:~/final_v1$ openssl verify -CAfile RootCA/rootCA.crt InterCA/InterCA.crt
InterCA/InterCA.crt: OK
```

Bob:

```
openssl verify -CAfile RootCA/rootCA.crt -untrusted InterCA/InterCA.crt
Bob/Bob.crt
```

Screenshot of the same:



```
ubuntu@ns00-gold:~/final_v1$ openssl verify -CAfile RootCA/rootCA.crt -untrusted InterCA/InterCA.crt Bob/Bob.crt
Bob/Bob.crt: OK
```

Alice:

```
openssl verify -CAfile RootCA/rootCA.crt -untrusted InterCA/InterCA.crt
Alice/Alice.crt
```

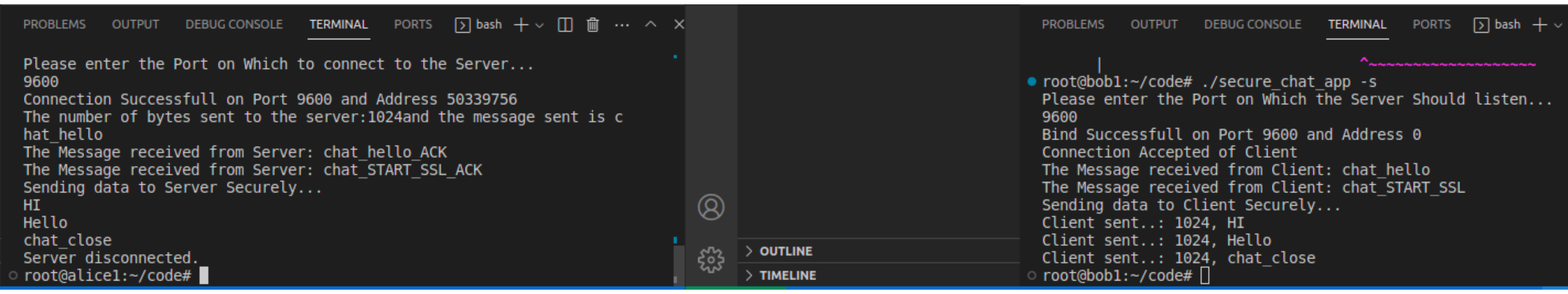
Screenshot of the same:

Task 2: Secure Chat App

The complete commented code is attached.
This section explains the various sections along with Pcap file

Alice IP - 172.31.0.2
Bob IP - 172.31.0.3
Server Port (9600) - Bob
Client Port (41870/38842) - Alice

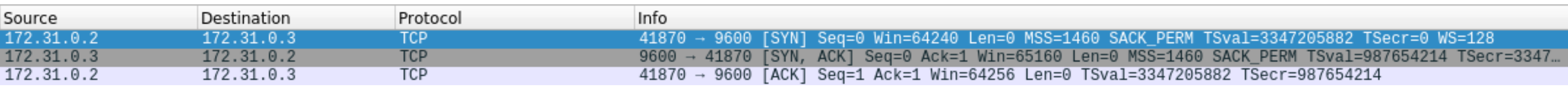
```
void type_client(char *server_name) //Function invoked if client argument passed when running out file


void type_server() //Function invoked if server argument passed when running out file
```



```
Please enter the Port on Which to connect to the Server...
9600
Connection Successfull on Port 9600 and Address 50339756
The number of bytes sent to the server:1024and the message sent is c
hat_hello
The Message received from Server: chat_hello_ACK
The Message received from Server: chat_START_SSL_ACK
Sending data to Server Securely...
HI
Hello
chat_close
Server disconnected.
root@alice1:~/code#
```

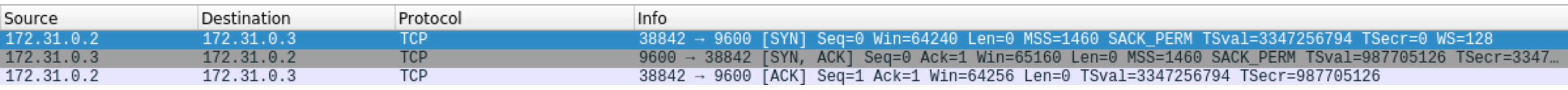
```
root@bob1:~/code# ./secure_chat_app -s
Please enter the Port on Which the Server Should listen...
9600
Bind Successfull on Port 9600 and Address 0
Connection Accepted of Client
The Message received from Client: chat_hello
The Message received from Client: chat_START_SSL
Sending data to Client Securely...
Client sent..: 1024, HI
Client sent..: 1024, Hello
Client sent..: 1024, chat_close
root@bob1:~/code#
```

a) TCP Connection Establishment:
Alice (PCAP):



Source	Destination	Protocol	Info
172.31.0.2	172.31.0.3	TCP	41870 → 9600 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3347205882 TSecr=0 WS=128
172.31.0.3	172.31.0.2	TCP	9600 → 41870 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=987654214 TSecr=3347…
172.31.0.2	172.31.0.3	TCP	41870 → 9600 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3347205882 TSecr=987654214

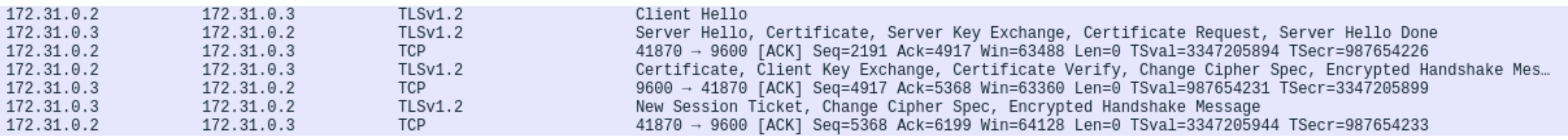
Bob (PCAP):



```
Source          Destination     Protocol            Info
172.31.0.2      172.31.0.3      TCP                 38842 → 9600 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3347256794 TSecr=0 WS=128
172.31.0.3      172.31.0.2      TCP                 9600 → 38842 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=987705126 TSecr=3347…
172.31.0.2      172.31.0.3      TCP                 38842 → 9600 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3347256794 TSecr=987705126
```

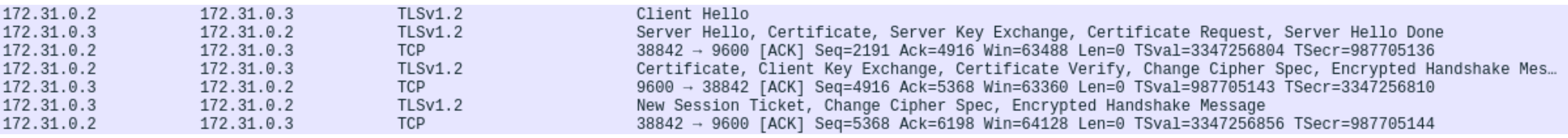
```
    struct sockaddr_in address; //Client Side connection, Server side is
similar (Implemented in code)

    address.sin_port=htons(port);
    address.sin_family=AF_INET;
    address.sin_addr= *((struct in_addr*)host->h_addr_list[0]);
    memset(&(address.sin_zero),'\0',8);
    int client_socket = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);


    int value_connect=connect(client_socket,(struct sockaddr
*)&address,sizeof(address));
    if(value_connect < 0){
        std::cout<<"Connect Failed..Exiting.."<<std::endl;
        exit(1);
    }

    if(value_connect!=-1){
        std::cout<<"Connection Successful on Port "<< port<< " and Address
"<<address.sin_addr.s_addr<<std::endl;
    }
```

b) **Application Layer Handshake:**
Alice (PCAP):



```
172.31.0.2      172.31.0.3      TLSv1.2             Client Hello
172.31.0.3      172.31.0.2      TLSv1.2             Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
172.31.0.2      172.31.0.3      TCP                 41870 → 9600 [ACK] Seq=2191 Ack=4917 Win=63488 Len=0 TSval=3347205894 TSecr=987654226
172.31.0.2      172.31.0.3      TLSv1.2             Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Mes…
172.31.0.3      172.31.0.2      TCP                 9600 → 41870 [ACK] Seq=4917 Ack=5368 Win=63360 Len=0 TSval=987654231 TSecr=3347205899
172.31.0.3      172.31.0.2      TLSv1.2             New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
172.31.0.2      172.31.0.3      TCP                 41870 → 9600 [ACK] Seq=5368 Ack=6199 Win=64128 Len=0 TSval=3347205944 TSecr=987654233
```

Bob (PCAP):

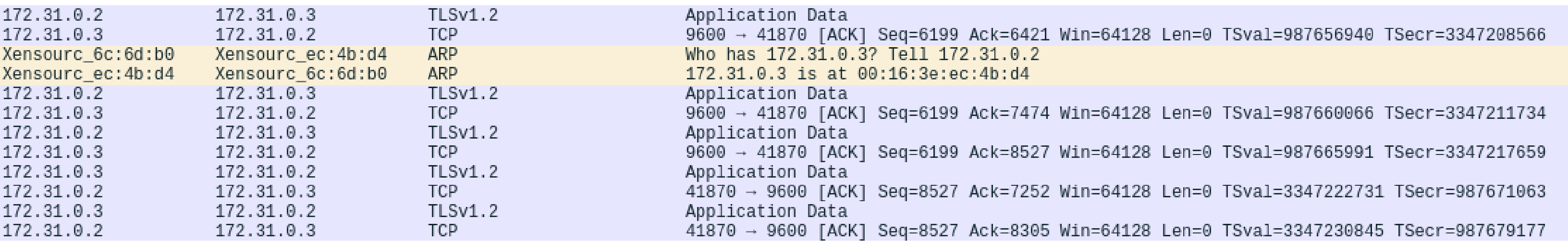


```
172.31.0.2      172.31.0.3      TLSv1.2             Client Hello
172.31.0.3      172.31.0.2      TLSv1.2             Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
172.31.0.2      172.31.0.3      TCP                 38842 → 9600 [ACK] Seq=2191 Ack=4916 Win=63488 Len=0 TSval=3347256804 TSecr=987705136
172.31.0.2      172.31.0.3      TLSv1.2             Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Mes…
172.31.0.3      172.31.0.2      TCP                 9600 → 38842 [ACK] Seq=4916 Ack=5368 Win=63360 Len=0 TSval=987705143 TSecr=3347256810
172.31.0.3      172.31.0.2      TLSv1.2             New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
172.31.0.2      172.31.0.3      TCP                 38842 → 9600 [ACK] Seq=5368 Ack=6198 Win=64128 Len=0 TSval=3347256856 TSecr=987705144
```

```
//Start Chat Hello Client
        int val_sent=send(client_socket , hello_client , 1024, 0);
        std::cout<<"The number of bytes sent to the
server:"<<val_sent<<"and the message sent is "<<hello_client<<std::endl;
        int valread = recv(client_socket ,buffer, 1024,0);
        std::cout<<"The Message received from Server: "<<buffer<<std::endl;

        //Start Chat SSL Hello
        memset(buffer, 0, sizeof(buffer));

        val_sent=send(client_socket , hello_ssl_client , 1024, 0);
        valread = recv(client_socket ,buffer, 1024,0);
        std::cout<<"The Message received from Server: "<<buffer<<std::endl;

        if(strcmp(buffer,"chat_START_SSL_NOT_SUPPORTED") == 0){
            std::cout<<"The Server Received chat_START_SSL_NOT_SUPPORTED.
Hence, proceeding with TCP Transfer..."<<std::endl;
            char buff_cipher[1024]={0};




//Start Chat Hello Server
    valread = recv(new_socket ,buffer, 1024,0);
    if(valread!=-1){
    std::cout<<"The Message received from Client: "<<buffer<<std::endl;
    }
    int valsent = send(new_socket , hello_server , 1024, 0 );

    //Start SSL Chat Hello
    memset(buffer, 0, sizeof(buffer));

    valread = recv(new_socket ,buffer, 1024,0);
    if(valread!=-1){
    std::cout<<"The Message received from Client: "<<buffer<<std::endl;
    }
    /*
    valsent = send(new_socket , hello_ssl_server , 1024, 0 );
    */
    if(strcmp(buffer,"chat_START_SSL_NOT_SUPPORTED") == 0){
        std::cout<<"The Server Received chat_START_SSL_NOT_SUPPORTED.
```

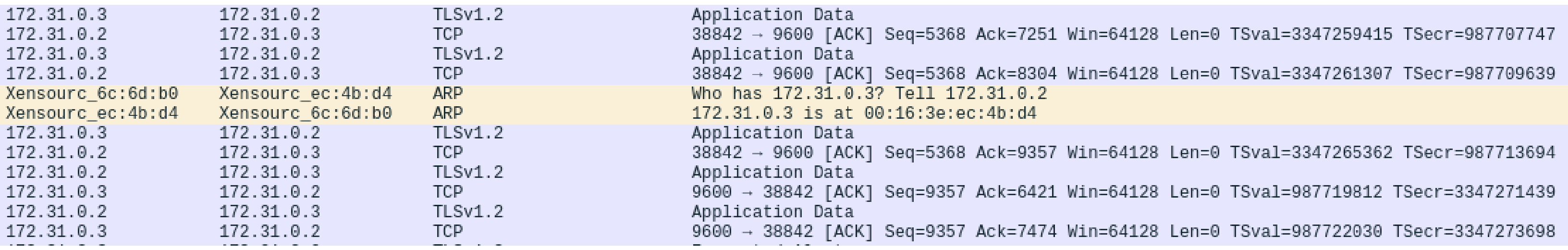
```
Hence, proceeding with TCP Transfer..."<<std::endl;
```

c) **Secure Chat Session:**
Alice (PCAP):



```
172.31.0.2          172.31.0.3          TLSv1.2             Application Data
172.31.0.3          172.31.0.2          TCP                 9600 → 41870 [ACK] Seq=6199 Ack=6421 Win=64128 Len=0 TSval=987656940 TSecr=3347208566
Xensourc_6c:6d:b0   Xensourc_ec:4b:d4   ARP                 Who has 172.31.0.3? Tell 172.31.0.2
Xensourc_ec:4b:d4   Xensourc_6c:6d:b0   ARP                 172.31.0.3 is at 00:16:3e:ec:4b:d4
172.31.0.2          172.31.0.3          TLSv1.2             Application Data
172.31.0.3          172.31.0.2          TCP                 9600 → 41870 [ACK] Seq=6199 Ack=7474 Win=64128 Len=0 TSval=987660066 TSecr=3347211734
172.31.0.2          172.31.0.3          TLSv1.2             Application Data
172.31.0.3          172.31.0.2          TCP                 9600 → 41870 [ACK] Seq=6199 Ack=8527 Win=64128 Len=0 TSval=987665991 TSecr=3347217659
172.31.0.3          172.31.0.2          TLSv1.2             Application Data
172.31.0.2          172.31.0.3          TCP                 41870 → 9600 [ACK] Seq=8527 Ack=7252 Win=64128 Len=0 TSval=3347222731 TSecr=987671063
172.31.0.3          172.31.0.2          TLSv1.2             Application Data
172.31.0.2          172.31.0.3          TCP                 41870 → 9600 [ACK] Seq=8527 Ack=8305 Win=64128 Len=0 TSval=3347230845 TSecr=987679177
```

Bob (PCAP):

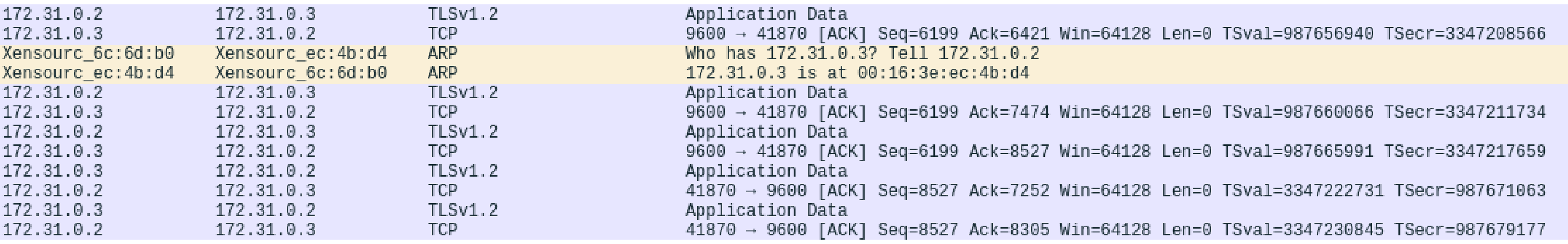


```
172.31.0.3          172.31.0.2          TLSv1.2             Application Data
172.31.0.2          172.31.0.3          TCP                 38842 → 9600 [ACK] Seq=5368 Ack=7251 Win=64128 Len=0 TSval=3347259415 TSecr=987707747
172.31.0.3          172.31.0.2          TLSv1.2             Application Data
172.31.0.2          172.31.0.3          TCP                 38842 → 9600 [ACK] Seq=5368 Ack=8304 Win=64128 Len=0 TSval=3347261307 TSecr=987709639
Xensourc_6c:6d:b0   Xensourc_ec:4b:d4   ARP                 Who has 172.31.0.3? Tell 172.31.0.2
Xensourc_ec:4b:d4   Xensourc_6c:6d:b0   ARP                 172.31.0.3 is at 00:16:3e:ec:4b:d4
172.31.0.3          172.31.0.2          TLSv1.2             Application Data
172.31.0.2          172.31.0.3          TCP                 38842 → 9600 [ACK] Seq=5368 Ack=9357 Win=64128 Len=0 TSval=3347265362 TSecr=987713694
172.31.0.2          172.31.0.3          TLSv1.2             Application Data
172.31.0.3          172.31.0.2          TCP                 9600 → 38842 [ACK] Seq=9357 Ack=6421 Win=64128 Len=0 TSval=987719812 TSecr=3347271439
172.31.0.2          172.31.0.3          TLSv1.2             Application Data
172.31.0.3          172.31.0.2          TCP                 9600 → 38842 [ACK] Seq=9357 Ack=7474 Win=64128 Len=0 TSval=987722030 TSecr=3347273698
```

```
SSL_CTX *ctx= create_cntx_client();

SSL_CTX_set_mode(ctx, SSL_MODE_AUTO_RETRY);
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT,
NULL);

/*Custom Settings*/
SSL_CTX_set_min_proto_version(ctx, TLS1_2_VERSION); //Sets the minimum
protocol version to TLS-1.2
SSL_CTX_set_max_proto_version(ctx, TLS1_3_VERSION); //Sets the maximum
protocol version to TLS-1.2

/*Custom Settings*/

//Sending those Cipher Suites which offer PFS
int cipher_value=SSL_CTX_set_cipher_list(ctx,
"ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-G
CM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:TLS_CHACHA20_POLY1305_SHA256");
if(!cipher_value){
    std::cout<<"No Suitable Cipher Suite found...Exiting.."<<std::endl;
```

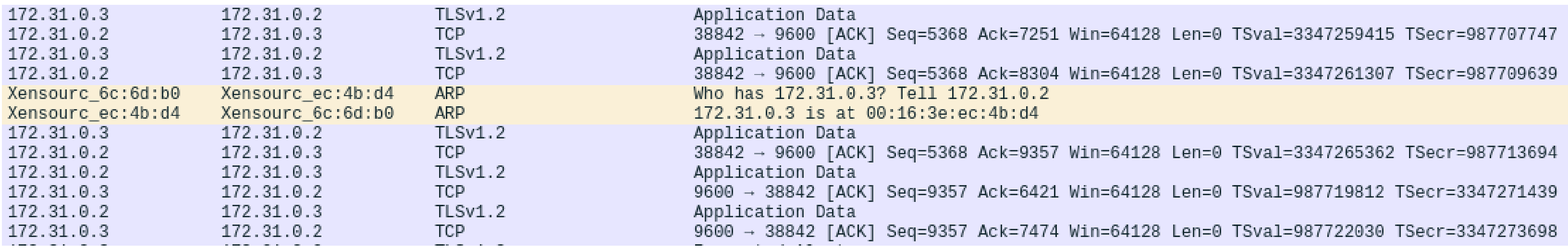
```
        exit(1);
    }
    SSL_CTX_set_ecdh_auto(ctx, 1);

    SSL_Certificates_cfgr_client(ctx); // This loads the Certificates, Private
    Key of the Client and also verifies the CA's Certificates (Similar
    implementation of server)
```

d) Encrypted Chat:
Alice (PCAP):



```
172.31.0.2          172.31.0.3          TLSv1.2                   Application Data
172.31.0.3          172.31.0.2          TCP                       9600 → 41870 [ACK] Seq=6199 Ack=6421 Win=64128 Len=0 TSval=987656940 TSecr=3347208566
Xensourc_6c:6d:b0   Xensourc_ec:4b:d4   ARP                       Who has 172.31.0.3? Tell 172.31.0.2
Xensourc_ec:4b:d4   Xensourc_6c:6d:b0   ARP                       172.31.0.3 is at 00:16:3e:ec:4b:d4
172.31.0.2          172.31.0.3          TLSv1.2                   Application Data
172.31.0.3          172.31.0.2          TCP                       9600 → 41870 [ACK] Seq=6199 Ack=7474 Win=64128 Len=0 TSval=987660066 TSecr=3347211734
172.31.0.2          172.31.0.3          TLSv1.2                   Application Data
172.31.0.3          172.31.0.2          TCP                       9600 → 41870 [ACK] Seq=6199 Ack=8527 Win=64128 Len=0 TSval=987665991 TSecr=3347217659
172.31.0.3          172.31.0.2          TLSv1.2                   Application Data
172.31.0.2          172.31.0.3          TCP                       41870 → 9600 [ACK] Seq=8527 Ack=7252 Win=64128 Len=0 TSval=3347222731 TSecr=987671063
172.31.0.3          172.31.0.2          TLSv1.2                   Application Data
172.31.0.2          172.31.0.3          TCP                       41870 → 9600 [ACK] Seq=8527 Ack=8305 Win=64128 Len=0 TSval=3347230845 TSecr=987679177
```

Bob (PCAP):



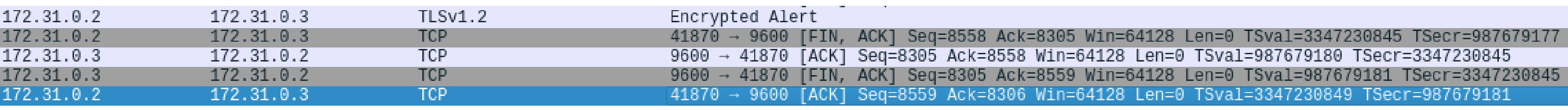
```
172.31.0.3          172.31.0.2          TLSv1.2                   Application Data
172.31.0.2          172.31.0.3          TCP                       38842 → 9600 [ACK] Seq=5368 Ack=7251 Win=64128 Len=0 TSval=3347259415 TSecr=987707747
172.31.0.3          172.31.0.2          TLSv1.2                   Application Data
172.31.0.2          172.31.0.3          TCP                       38842 → 9600 [ACK] Seq=5368 Ack=8304 Win=64128 Len=0 TSval=3347261307 TSecr=987709639
Xensourc_6c:6d:b0   Xensourc_ec:4b:d4   ARP                       Who has 172.31.0.3? Tell 172.31.0.2
Xensourc_ec:4b:d4   Xensourc_6c:6d:b0   ARP                       172.31.0.3 is at 00:16:3e:ec:4b:d4
172.31.0.3          172.31.0.2          TLSv1.2                   Application Data
172.31.0.2          172.31.0.3          TCP                       38842 → 9600 [ACK] Seq=5368 Ack=9357 Win=64128 Len=0 TSval=3347265362 TSecr=987713694
172.31.0.2          172.31.0.3          TLSv1.2                   Application Data
172.31.0.3          172.31.0.2          TCP                       9600 → 38842 [ACK] Seq=9357 Ack=6421 Win=64128 Len=0 TSval=987719812 TSecr=3347271439
172.31.0.2          172.31.0.3          TLSv1.2                   Application Data
172.31.0.3          172.31.0.2          TCP                       9600 → 38842 [ACK] Seq=9357 Ack=7474 Win=64128 Len=0 TSval=987722030 TSecr=3347273698
```

```
    /*We used Polling for seamless communication from both sides, i.e Any side
    doesn't have to wait for a message from other side to get reply
    Like in above PCAP - server is sending back to back message from port 9600
    (Last two lines) without getting any message from client
    (Can be observed in Task 3 plaintext communication more readily)*/

    SSL *ssl;
    char buff_cipher[1024]={0};

        struct pollfd fds[1];

        fds[0].fd = STDIN_FILENO;
        fds[0].events = POLLIN;
```

```
fds[1].fd = client_socket;
fds[1].events = POLLIN;

int n_clients = 0;

ssl=SSL_new(ctx);
SSL_set_verify_depth(ssl, 2);
//SSL_set_tlsext_host_name(ssl,"Bob1.com");

if(ssl == NULL){
    std::cout<<"Failed..."<<std::endl;
}
SSL_set_fd(ssl, client_socket);

if (SSL_connect(ssl) <= 0) {
    std::cout<<"There is an error"<<std::endl;
    ERR_print_errors_fp(stderr);
}
else
{   std::cout<<"Sending data to Server Securely..."<<std::endl;
    while(poll(fds, 2, -1)!=-1){
    char buf[1024];
    memset(buf,0,1024);
    if (fds[0].revents & POLLIN) {
        ssize_t len = read(STDIN_FILENO, buf, sizeof(buf));
        if (len > 0) {
                SSL_write(ssl,buf,1024);
        }
    }
    if (fds[1].revents & POLLIN) {
        ssize_t len = SSL_read(ssl,buf,1024);
        if (len > 0) {
            printf("Server sent..: %d, %s", (int)len, buf);
            if(strcmp(buf,"chat_close\n")==0){
                std::cout << "yoyoyo";
                SSL_shutdown(ssl);
                SSL_free(ssl);
                close(value_connect);
                break;
            }
        } else {

            printf("Server disconnected.\n");
```

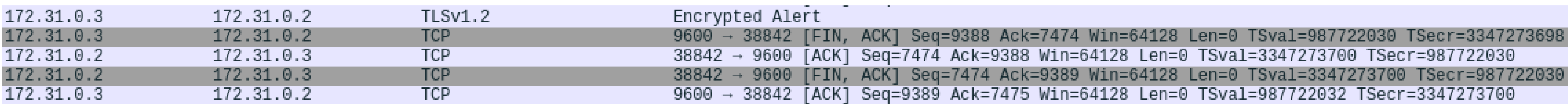
```
                    break;
                }
            }
        }
    }
close(client_socket);
SSL_CTX_free(ctx);
        }
}
```

e) **Chat Close:**
Alice (PCAP):



```
172.31.0.2   172.31.0.3   TLSv1.2   Encrypted Alert
172.31.0.2   172.31.0.3   TCP       41870 → 9600 [FIN, ACK] Seq=8558 Ack=8305 Win=64128 Len=0 TSval=3347230845 TSecr=987679177
172.31.0.3   172.31.0.2   TCP       9600 → 41870 [ACK] Seq=8305 Ack=8558 Win=64128 Len=0 TSval=987679180 TSecr=3347230845
172.31.0.3   172.31.0.2   TCP       9600 → 41870 [FIN, ACK] Seq=8305 Ack=8559 Win=64128 Len=0 TSval=987679181 TSecr=3347230845
172.31.0.2   172.31.0.3   TCP       41870 → 9600 [ACK] Seq=8559 Ack=8306 Win=64128 Len=0 TSval=3347230849 TSecr=987679181
```

Bob (PCAP):



```
172.31.0.3   172.31.0.2   TLSv1.2   Encrypted Alert
172.31.0.3   172.31.0.2   TCP       9600 → 38842 [FIN, ACK] Seq=9388 Ack=7474 Win=64128 Len=0 TSval=987722030 TSecr=3347273698
172.31.0.2   172.31.0.3   TCP       38842 → 9600 [ACK] Seq=7474 Ack=9388 Win=64128 Len=0 TSval=3347273700 TSecr=987722030
172.31.0.2   172.31.0.3   TCP       38842 → 9600 [FIN, ACK] Seq=7474 Ack=9389 Win=64128 Len=0 TSval=3347273700 TSecr=987722030
172.31.0.3   172.31.0.2   TCP       9600 → 38842 [ACK] Seq=9389 Ack=7475 Win=64128 Len=0 TSval=987722032 TSecr=3347273700
```

```
if (fds[1].revents & POLLIN) {
           ssize_t len = SSL_read(ssl,buf,1024);
           if (len > 0) {
               printf("Server sent..: %d, %s", (int)len, buf);
               if(strcmp(buf,"chat_close\n")==0){
                   SSL_shutdown(ssl);
                   SSL_free(ssl);
                   close(value_connect);
                   break;
               }
           } //Chat close implementation using strcmp
```

Task 3: START_SSL downgrade attack #1 for eavesdropping

Attack:

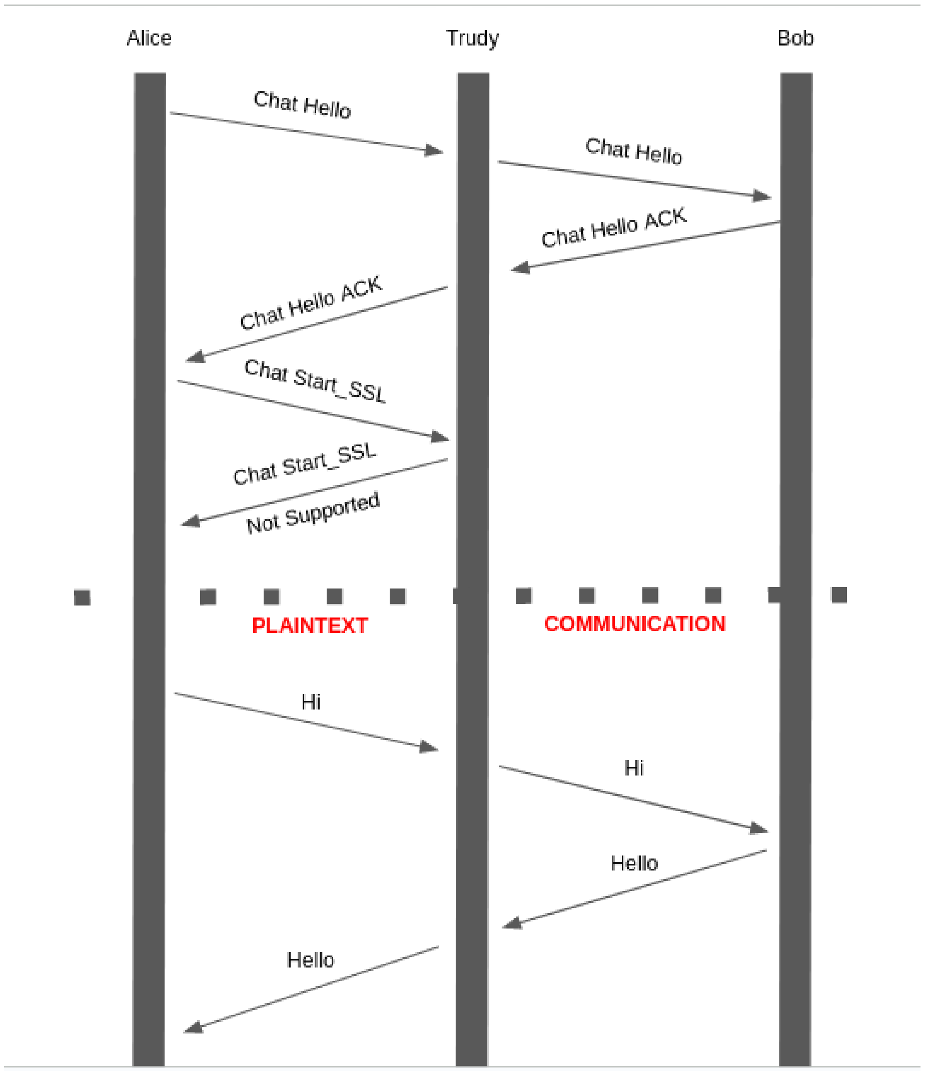
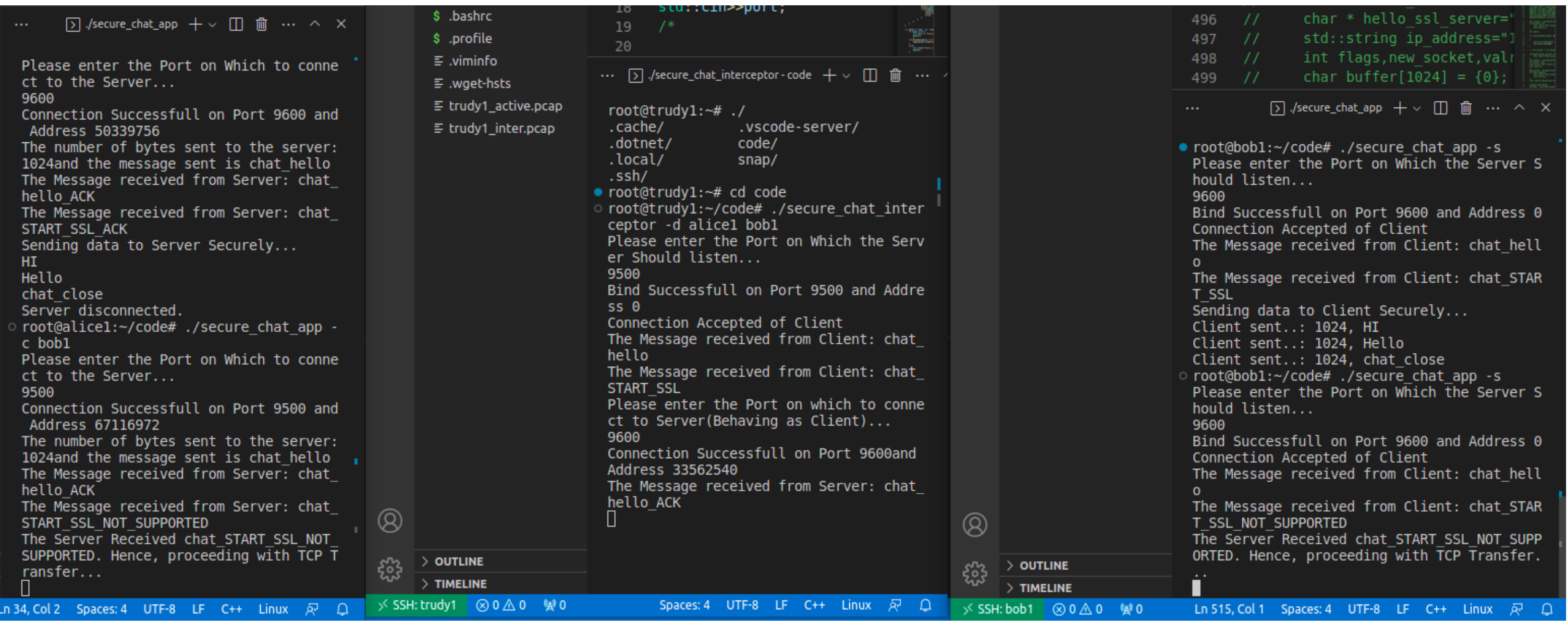


Fig. This shows that the connection is not encrypted and the data is being transferred in plain text over TCP sockets.


```
./secure_chat_app

Please enter the Port on Which to conne
ct to the Server...
9600
Connection Successfull on Port 9600 and
 Address 50339756
The number of bytes sent to the server:
1024and the message sent is chat_hello
The Message received from Server: chat_
hello_ACK
The Message received from Server: chat_
START_SSL_ACK
Sending data to Server Securely...
HI
Hello
chat_close
Server disconnected.
root@alice1:~/code# ./secure_chat_app -
c bob1
Please enter the Port on Which to conne
ct to the Server...
9500
Connection Successfull on Port 9500 and
 Address 67116972
The number of bytes sent to the server:
1024and the message sent is chat_hello
The Message received from Server: chat_
hello_ACK
The Message received from Server: chat_
START_SSL_NOT_SUPPORTED
The Server Received chat_START_SSL_NOT_
SUPPORTED. Hence, proceeding with TCP T
ransfer...
```

```
$ .bashrc
$ .profile
≡ .viminfo
≡ .wget-hsts
≡ trudy1_active.pcap
≡ trudy1_inter.pcap
```

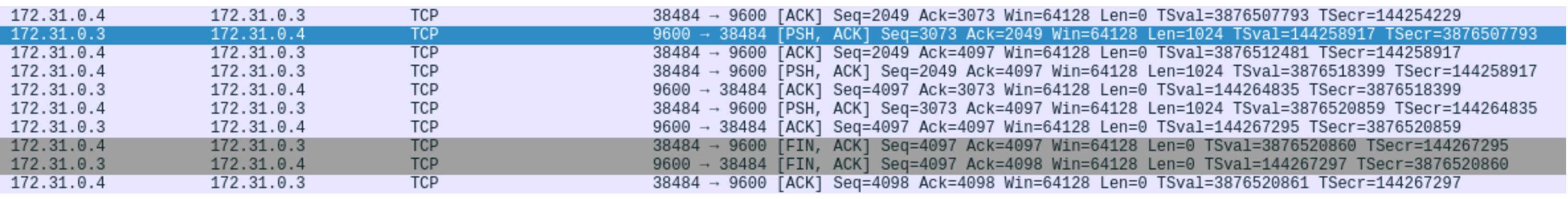
```
std::cin>>port;
/*
```

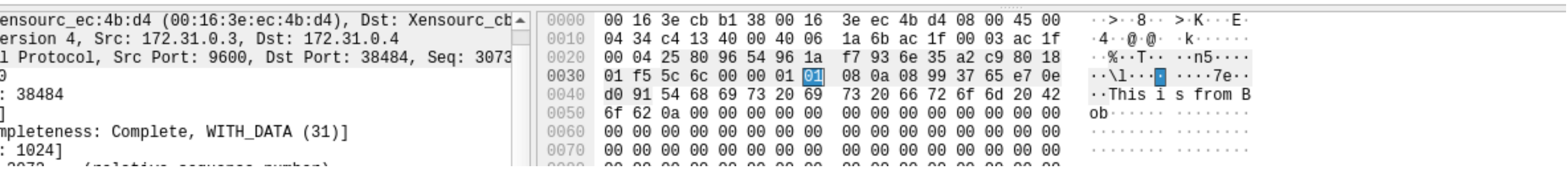
```
./secure_chat_interceptor - code

root@trudy1:~# ./
.cache/          .vscode-server/
.dotnet/         code/
.local/          snap/
.ssh/
root@trudy1:~# cd code
root@trudy1:~/code# ./secure_chat_inter
ceptor -d alice1 bob1
Please enter the Port on Which the Serv
er Should listen...
9500
Bind Successfull on Port 9500 and Addre
ss 0
Connection Accepted of Client
The Message received from Client: chat_
hello
The Message received from Client: chat_
START_SSL
Please enter the Port on which to conne
ct to Server(Behaving as Client)...
9600
Connection Successfull on Port 9600and
Address 33562540
The Message received from Server: chat_
hello_ACK
```

```
//      char * hello_ssl_server="
//      std::string ip_address="
//      int flags,new_socket,val
//      char buffer[1024] = {0};
```

```
./secure_chat_app

root@bob1:~/code# ./secure_chat_app -s
Please enter the Port on Which the Server S
hould listen...
9600
Bind Successfull on Port 9600 and Address 0
Connection Accepted of Client
The Message received from Client: chat_hell
o
The Message received from Client: chat_STAR
T_SSL
Sending data to Client Securely...
Client sent..: 1024, HI
Client sent..: 1024, Hello
Client sent..: 1024, chat_close
root@bob1:~/code# ./secure_chat_app -s
Please enter the Port on Which the Server S
hould listen...
9600
Bind Successfull on Port 9600 and Address 0
Connection Accepted of Client
The Message received from Client: chat_hell
o
The Message received from Client: chat_STAR
T_SSL_NOT_SUPPORTED
The Server Received chat_START_SSL_NOT_SUPP
ORTED. Hence, proceeding with TCP Transfer.
..
```

Fig. Executing from VsCode


```
172.31.0.4    172.31.0.3    TCP    38484 → 9600 [ACK] Seq=2049 Ack=3073 Win=64128 Len=0 TSval=3876507793 TSecr=144254229
172.31.0.3    172.31.0.4    TCP    9600 → 38484 [PSH, ACK] Seq=3073 Ack=2049 Win=64128 Len=1024 TSval=144258917 TSecr=3876507793
172.31.0.4    172.31.0.3    TCP    38484 → 9600 [ACK] Seq=2049 Ack=4097 Win=64128 Len=0 TSval=3876512481 TSecr=144258917
172.31.0.4    172.31.0.3    TCP    38484 → 9600 [PSH, ACK] Seq=2049 Ack=4097 Win=64128 Len=1024 TSval=3876518399 TSecr=144258917
172.31.0.3    172.31.0.4    TCP    9600 → 38484 [ACK] Seq=4097 Ack=3073 Win=64128 Len=0 TSval=144264835 TSecr=3876518399
172.31.0.4    172.31.0.3    TCP    38484 → 9600 [PSH, ACK] Seq=3073 Ack=4097 Win=64128 Len=1024 TSval=3876520859 TSecr=144264835
172.31.0.3    172.31.0.4    TCP    9600 → 38484 [ACK] Seq=4097 Ack=4097 Win=64128 Len=0 TSval=144267295 TSecr=3876520859
172.31.0.4    172.31.0.3    TCP    38484 → 9600 [FIN, ACK] Seq=4097 Ack=4097 Win=64128 Len=0 TSval=3876520860 TSecr=144267295
172.31.0.3    172.31.0.4    TCP    9600 → 38484 [FIN, ACK] Seq=4097 Ack=4098 Win=64128 Len=0 TSval=144267297 TSecr=3876520860
172.31.0.4    172.31.0.3    TCP    38484 → 9600 [ACK] Seq=4098 Ack=4098 Win=64128 Len=0 TSval=3876520861 TSecr=144267297
```


```
ensourc_ec:4b:d4 (00:16:3e:ec:4b:d4), Dst: Xensourc_cb
ersion 4, Src: 172.31.0.3, Dst: 172.31.0.4
l Protocol, Src Port: 9600, Dst Port: 38484, Seq: 3073
0
: 38484
]
mpleteness: Complete, WITH_DATA (31)]
: 1024]
```

```
0000  00 16 3e cb b1 38 00 16  3e ec 4b d4 08 00 45 00   ··>··8·· >·K···E·
0010  04 34 c4 13 40 00 40 06  1a 6b ac 1f 00 03 ac 1f   ·4··@·@· ·k······
0020  00 04 25 80 96 54 96 1a  f7 93 6e 35 a2 c9 80 18   ··%··T·· ··n5····
0030  01 f5 5c 6c 00 00 01 01  08 0a 08 99 37 65 e7 0e   ··\l···· ····7e··
0040  d0 91 54 68 69 73 20 69  73 20 66 72 6f 6d 20 42   ··This i s from B
0050  6f 62 0a 00 00 00 00 00  00 00 00 00 00 00 00 00   ob······ ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

Fig. Messages are going in plain text due to Trudy's intrusion

Code snippet of simultaneous polling on both sides for seamless chat:

```
    /*

 ############################################################################
##
        Now proceeding with intruding the traffic of Alice and Bob via TCP
transfer...

 ############################################################################
##
        */
       // char buff_cipher[1024]={0};

    struct pollfd fds[1];

    fds[0].fd = client_socket;  //Client Socket FD.
    fds[0].events = POLLIN;
    fds[1].fd = new_socket;         //Server Socket FD.
    fds[1].events = POLLIN;

    //int n_clients = 0;

        while(poll(fds, 2, -1)!=-1){
        char buf[1024];
        memset(buf,0,1024);
        if (fds[0].revents & POLLIN) {
            ssize_t len = read(client_socket, buf, sizeof(buf));
            if (len > 0) {
                    printf("Server sent..: %d, %s", (int)len, buf);
                    write(new_socket,buf,1024);
                    if(strcmp(buf,"chat_close\n")==0){
                    close(client_socket);
                    close(new_socket);
                    break;
                }
            }
        }
        if (fds[1].revents & POLLIN) {
            ssize_t len = read(new_socket,buf,1024);
            if (len > 0) {
                printf("Client sent..: %d, %s", (int)len, buf);
                write(client_socket,buf,1024);
                if(strcmp(buf,"chat_close\n")==0){
```

```
                close(new_socket);
                close(client_socket);
                break;
            }
        }
    }
}
```

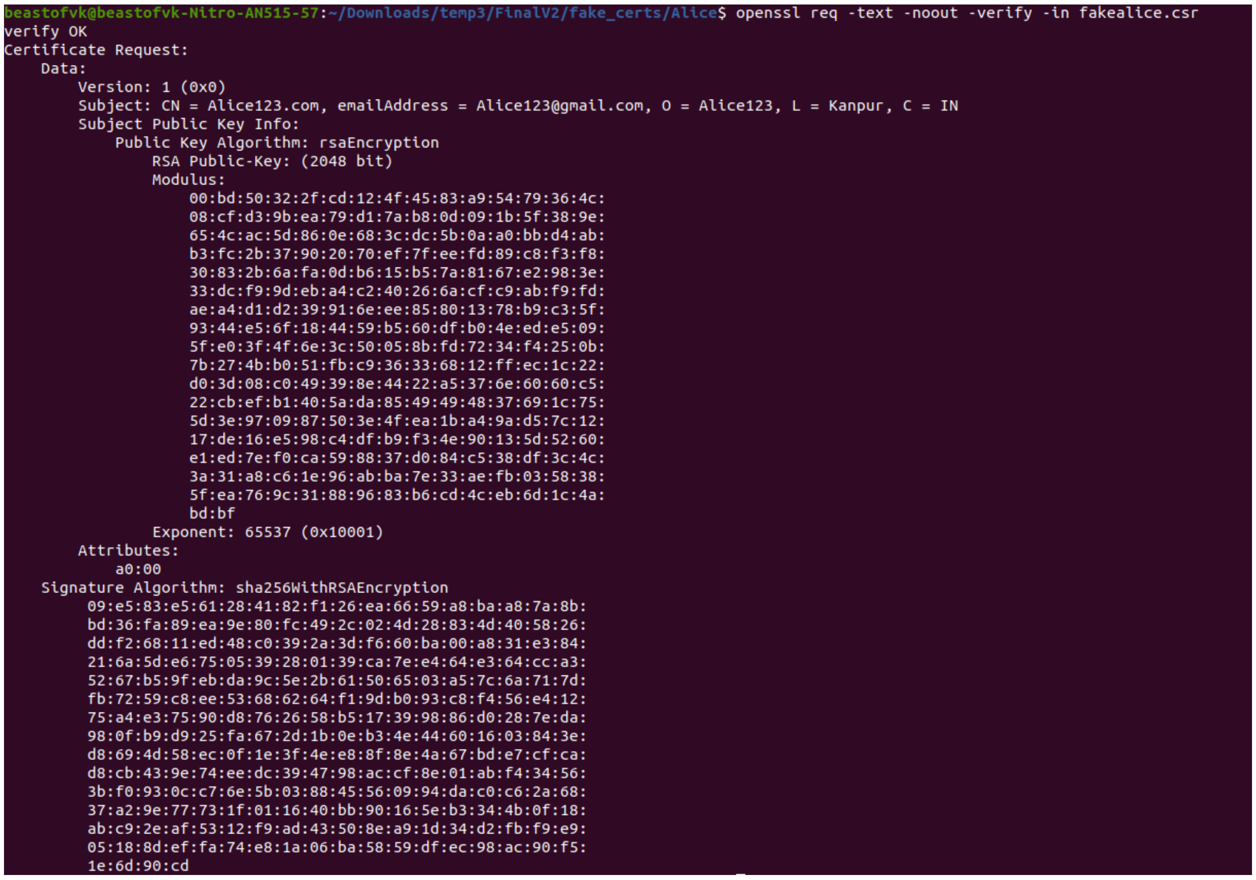
Task 4: START_SSL downgrade attack #2 for tampering

- CSR verification of the fake CSR's of Alice and Bob:

FakeAlice:

```
openssl req -text -noout -verify -in fakealice.csr
```

Screenshot of the same:

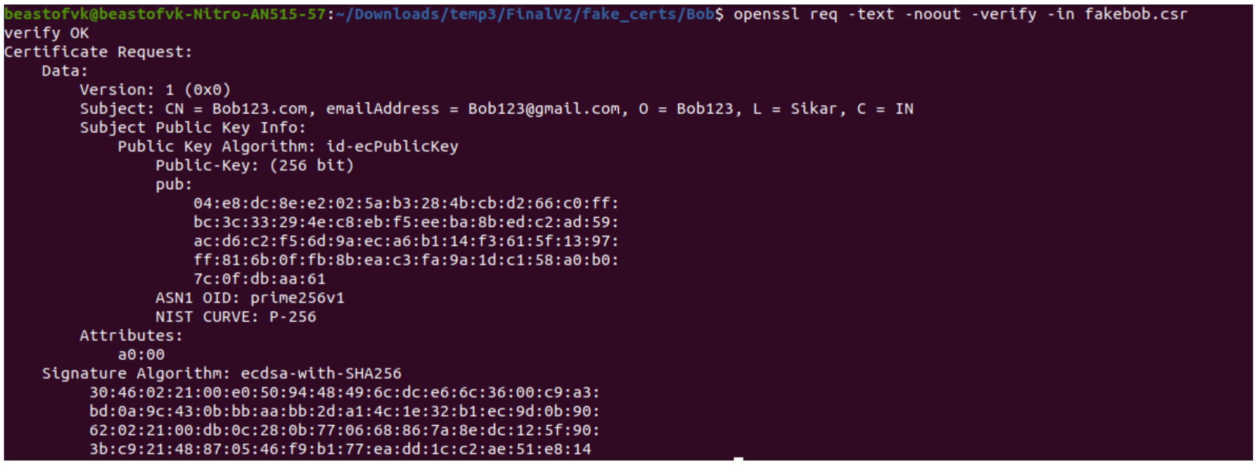


```
beastofvk@beastofvk-Nitro-AN515-57:~/Downloads/temp3/FinalV2/fake_certs/Alice$ openssl req -text -noout -verify -in fakealice.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = Alice123.com, emailAddress = Alice123@gmail.com, O = Alice123, L = Kanpur, C = IN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:bd:50:32:2f:cd:12:4f:45:83:a9:54:79:36:4c:
                    08:cf:d3:9b:ea:79:d1:7a:b8:0d:09:1b:5f:38:9e:
                    65:4c:ac:5d:86:0e:68:3c:dc:5b:0a:a0:bb:d4:ab:
                    b3:fc:2b:37:90:20:70:ef:7f:ee:fd:89:c8:f3:f8:
                    30:83:2b:6a:fa:0d:b6:15:b5:7a:81:67:e2:98:3e:
                    33:dc:f9:9d:eb:a4:c2:40:26:6a:cf:c9:ab:f9:fd:
                    ae:a4:d1:d2:39:91:6e:ee:85:80:13:78:b9:c3:5f:
                    93:44:e5:6f:18:44:59:b5:60:df:b0:4e:ed:e5:09:
                    5f:e0:3f:4f:6e:3c:50:05:8b:fd:72:34:f4:25:0b:
                    7b:27:4b:b0:51:fb:c9:36:33:68:12:ff:ec:1c:22:
                    d0:3d:08:c0:49:39:8e:44:22:a5:37:6e:60:60:c5:
                    22:cb:ef:b1:40:5a:da:85:49:49:48:37:69:1c:75:
                    5d:3e:97:09:87:50:3e:4f:ea:1b:a4:9a:d5:7c:12:
                    17:de:16:e5:98:c4:df:b9:f3:4e:90:13:5d:52:60:
                    e1:ed:7e:f0:ca:59:88:37:d0:84:c5:38:df:3c:4c:
                    3a:31:a8:c6:1e:96:ab:ba:7e:33:ae:fb:03:58:38:
                    5f:ea:76:9c:31:88:96:83:b6:cd:4c:eb:6d:1c:4a:
                    bd:bf
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         09:e5:83:e5:61:28:41:82:f1:26:ea:66:59:a8:ba:a8:7a:8b:
         bd:36:fa:89:ea:9e:80:fc:49:2c:02:4d:28:83:4d:40:58:26:
         dd:f2:68:11:ed:48:c0:39:2a:3d:f6:60:ba:00:a8:31:e3:84:
         21:6a:5d:e6:75:05:39:28:01:39:ca:7e:e4:64:e3:64:cc:a3:
         52:67:b5:9f:eb:da:9c:5e:2b:61:50:65:03:a5:7c:6a:71:7d:
         fb:72:59:c8:ee:53:68:62:64:f1:9d:b0:93:c8:f4:56:e4:12:
         75:a4:e3:75:90:d8:76:26:58:b5:17:39:98:86:d0:28:7e:da:
         98:0f:b9:d9:25:fa:67:2d:1b:0e:b3:4e:44:60:16:03:84:3e:
         d8:69:4d:58:ec:0f:1e:3f:4e:e8:8f:8e:4a:67:bd:e7:cf:ca:
         d8:cb:43:9e:74:ee:dc:39:47:98:ac:cf:8e:01:ab:f4:34:56:
         3b:f0:93:0c:c7:6e:5b:03:88:45:56:09:94:da:c0:c6:2a:68:
         37:a2:9e:77:73:1f:01:16:40:bb:90:16:5e:b3:34:4b:0f:18:
         ab:c9:2e:af:53:12:f9:ad:43:50:8e:a9:1d:34:d2:fb:f9:e9:
         05:18:8d:ef:fa:74:e8:1a:06:ba:58:59:df:ec:98:ac:90:f5:
         1e:6d:90:cd
```

FakeBob:

```
openssl req -text -noout -verify -in fakebob.csr
```

Screenhot of the same:



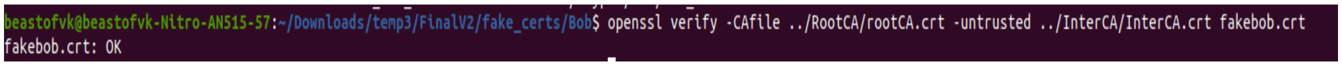
```
beastofvk@beastofvk-Nitro-AN515-57:~/Downloads/temp3/FinalV2/fake_certs/Bob$ openssl req -text -noout -verify -in fakebob.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = Bob123.com, emailAddress = Bob123@gmail.com, O = Bob123, L = Sikar, C = IN
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:e8:dc:8e:e2:02:5a:b3:28:4b:cb:d2:66:c0:ff:
                    bc:3c:33:29:4e:c8:eb:f5:ee:ba:8b:ed:c2:ad:59:
                    ac:d6:c2:f5:6d:9a:ec:a6:b1:14:f3:61:5f:13:97:
                    ff:81:6b:0f:fb:8b:ea:c3:fa:9a:1d:c1:58:a0:b0:
                    7c:0f:db:aa:61
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            a0:00
    Signature Algorithm: ecdsa-with-SHA256
         30:46:02:21:00:e0:50:94:48:49:6c:dc:e6:6c:36:00:c9:a3:
         bd:0a:9c:43:0b:bb:aa:bb:2d:a1:4c:1e:32:b1:ec:9d:0b:90:
         62:02:21:00:db:0c:28:0b:77:06:68:86:7a:8e:dc:12:5f:90:
         3b:c9:21:48:87:05:46:f9:b1:77:ea:dd:1c:c2:ae:51:e8:14
```

- Certificate verification of Alice and Bob's fake certificates.

FakeAlice:

```
openssl verify -CAfile ../RootCA/rootCA.crt -untrusted
../InterCA/InterCA.crt fakealice.crt
```

Screenshot of the same:

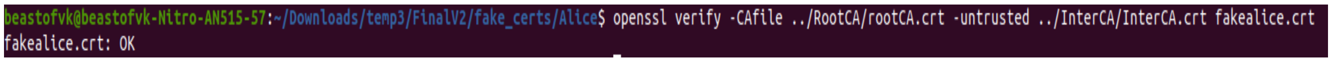


```
beastofvk@beastofvk-Nitro-AN515-57:~/Downloads/temp3/FinalV2/fake_certs/Bob$ openssl verify -CAfile ../RootCA/rootCA.crt -untrusted ../InterCA/InterCA.crt fakebob.crt
fakebob.crt: OK
```

FakeBob:

```
openssl verify -CAfile ../RootCA/rootCA.crt -untrusted
../InterCA/InterCA.crt fakebob.crt
```

Screenshot of the same:



```
beastofvk@beastofvk-Nitro-AN515-57:~/Downloads/temp3/FinalV2/fake_certs/Alice$ openssl verify -CAfile ../RootCA/rootCA.crt -untrusted ../InterCA/InterCA.crt fakealice.crt
fakealice.crt: OK
```

Attack:

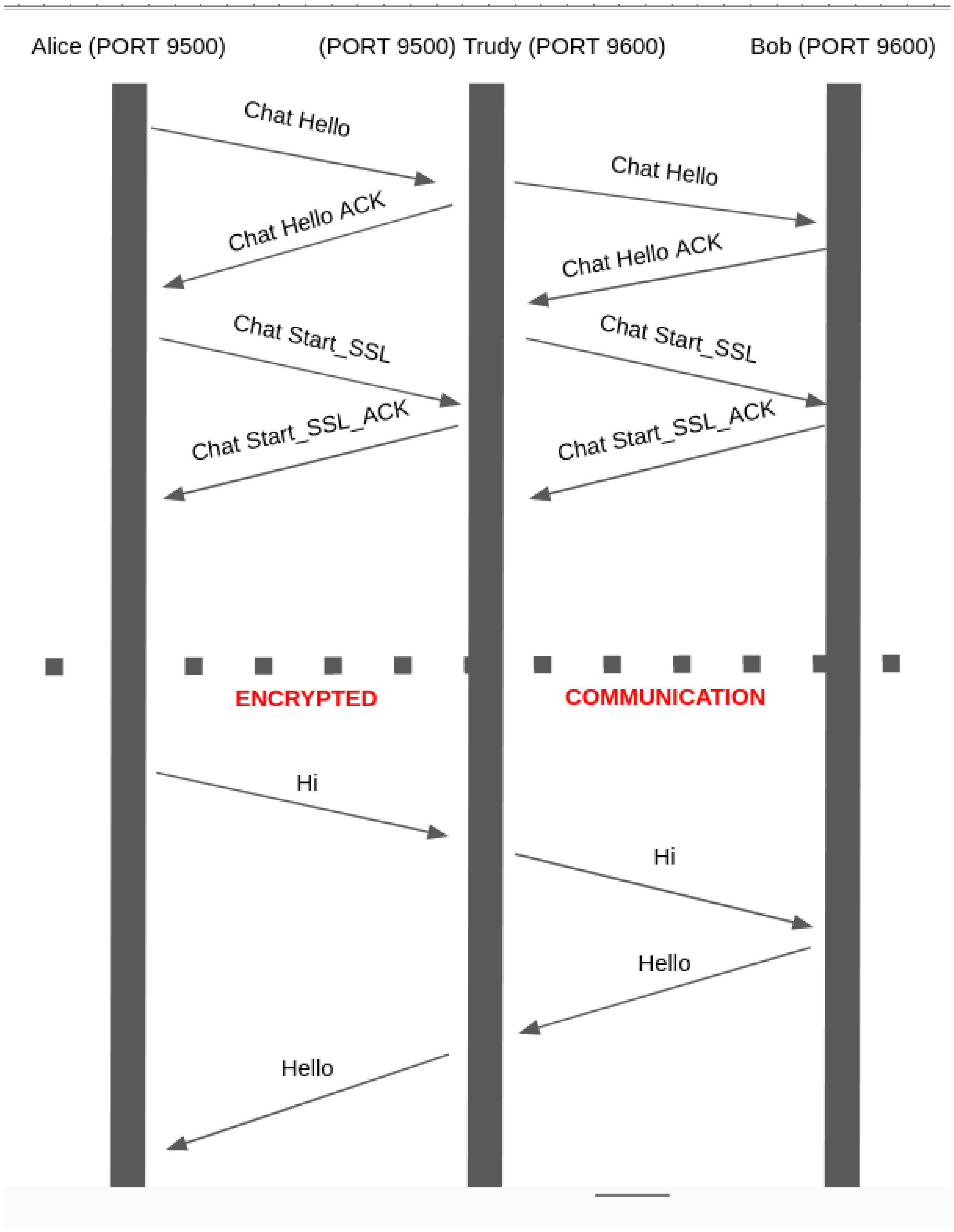
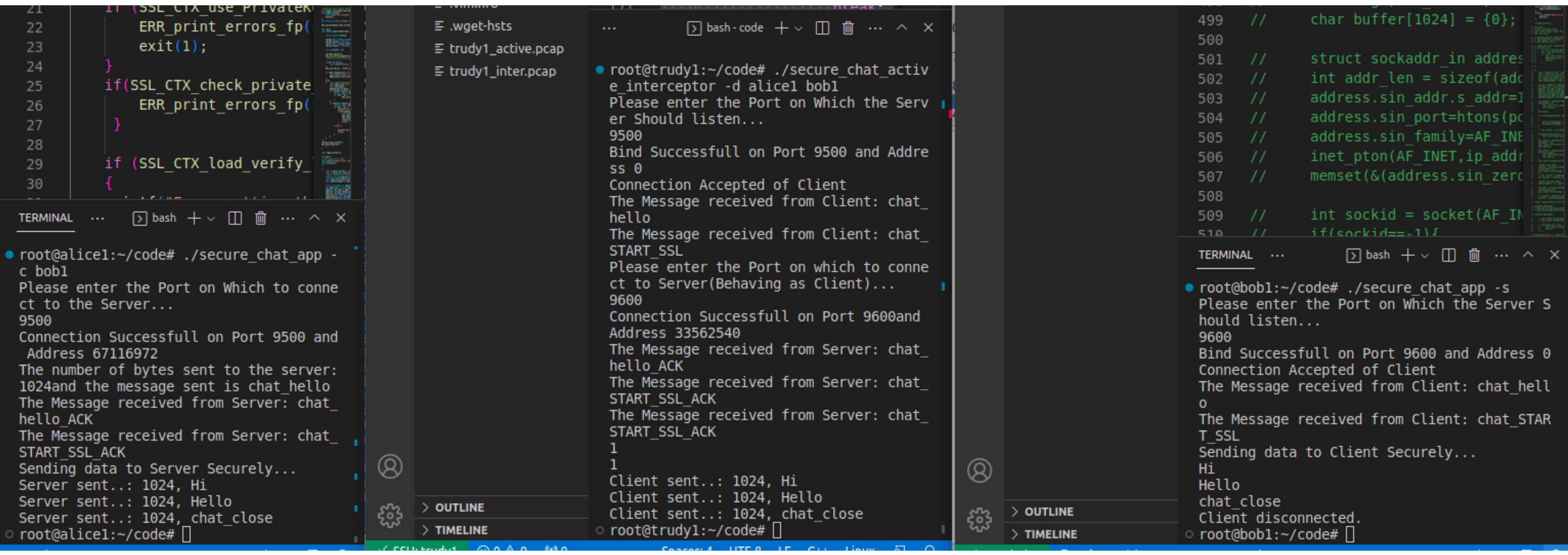


Fig. This shows that the connection is encrypted and the data is being transferred in encrypted text over TLS.



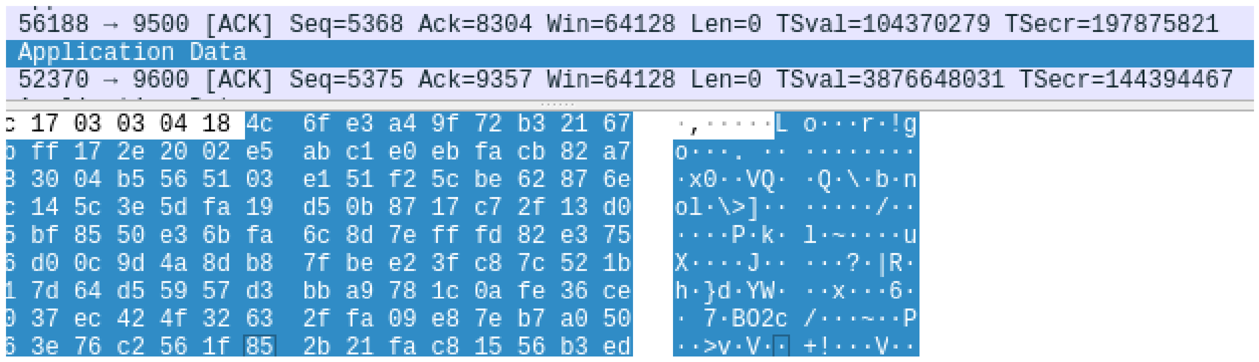
```
        ERR_print_errors_fp(
        exit(1);
    }
    if(SSL_CTX_check_private
        ERR_print_errors_fp(
     }

    if (SSL_CTX_load_verify_
    {
```

```
root@alice1:~/code# ./secure_chat_app -
c bob1
Please enter the Port on Which to conne
ct to the Server...
9500
Connection Successfull on Port 9500 and
 Address 67116972
The number of bytes sent to the server:
1024and the message sent is chat_hello
The Message received from Server: chat_
hello_ACK
The Message received from Server: chat_
START_SSL_ACK
Sending data to Server Securely...
Server sent..: 1024, Hi
Server sent..: 1024, Hello
Server sent..: 1024, chat_close
root@alice1:~/code#
```

```
root@trudy1:~/code# ./secure_chat_activ
e_interceptor -d alice1 bob1
Please enter the Port on Which the Serv
er Should listen...
9500
Bind Successfull on Port 9500 and Addre
ss 0
Connection Accepted of Client
The Message received from Client: chat_
hello
The Message received from Client: chat_
START_SSL
Please enter the Port on which to conne
ct to Server(Behaving as Client)...
9600
Connection Successfull on Port 9600and
Address 33562540
The Message received from Server: chat_
hello_ACK
The Message received from Server: chat_
START_SSL_ACK
The Message received from Server: chat_
START_SSL_ACK
1
1
Client sent..: 1024, Hi
Client sent..: 1024, Hello
Client sent..: 1024, chat_close
root@trudy1:~/code#
```

```
//      char buffer[1024] = {0};

//      struct sockaddr_in addres
//      int addr_len = sizeof(adc
//      address.sin_addr.s_addr=I
//      address.sin_port=htons(pc
//      address.sin_family=AF_INE
//      inet_pton(AF_INET,ip_addr
//      memset(&(address.sin_zerc

//      int sockid = socket(AF_IN
```

```
root@bob1:~/code# ./secure_chat_app -s
Please enter the Port on Which the Server S
hould listen...
9600
Bind Successfull on Port 9600 and Address 0
Connection Accepted of Client
The Message received from Client: chat_hell
o
The Message received from Client: chat_STAR
T_SSL
Sending data to Client Securely...
Hi
Hello
chat_close
Client disconnected.
root@bob1:~/code#
```



```
56188 → 9500 [ACK] Seq=5368 Ack=8304 Win=64128 Len=0 TSval=104370279 TSecr=197875821
Application Data
52370 → 9600 [ACK] Seq=5375 Ack=9357 Win=64128 Len=0 TSval=3876648031 TSecr=144394467
```

```
c 17 03 03 04 18 4c  6f e3 a4 9f 72 b3 21 67    ·,·····L o···r·!g
o ff 17 2e 20 02 e5  ab c1 e0 eb fa cb 82 a7    o···. ·· ········
3 30 04 b5 56 51 03  e1 51 f2 5c be 62 87 6e    ·x0··VQ· ·Q·\·b·n
c 14 5c 3e 5d fa 19  d5 0b 87 17 c7 2f 13 d0    ol·\>]·· ·····/··
5 bf 85 50 e3 6b fa  6c 8d 7e ff fd 82 e3 75    ····P·k· l·~····u
6 d0 0c 9d 4a 8d b8  7f be e2 3f c8 7c 52 1b    X····J·· ···?·|R·
1 7d 64 d5 59 57 d3  bb a9 78 1c 0a fe 36 ce    h·}d·YW· ··x···6·
0 37 ec 42 4f 32 63  2f fa 09 e8 7e b7 a0 50    · 7·BO2c /···~··P
6 3e 76 c2 56 1f 85  2b 21 fa c8 15 56 b3 ed    ··>v·V·· +!···V··
```

As we can see, 56188 (Alice's port) sending to 9500 (Trudy's server side port) and Trudy forwards it to 9600 (Bob's server side port) from 52370 (Trudy's Client side port)

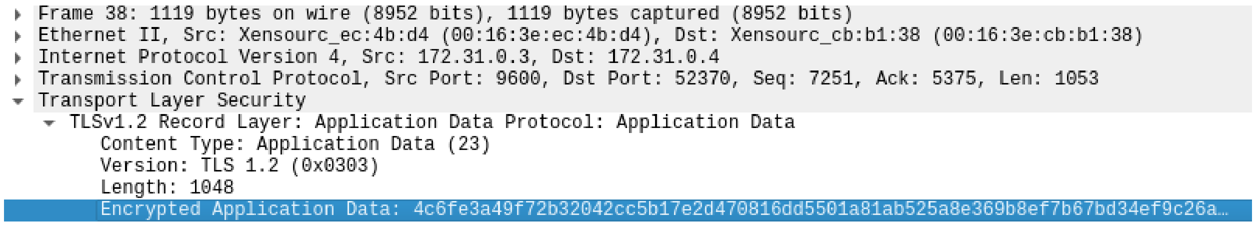
Code below demonstrates operation of separate TLS Pipes that Trudy operates (secure_chat_active_interceptor.cpp) rest of code, setup, etc remains similar to secure chat.

```
struct pollfd fds[1];

fds[0].fd = client_socket;  //Client Socket FD.
fds[0].events = POLLIN;
```

```
    fds[1].fd = new_socket;     //Server Socket FD.
    fds[1].events = POLLIN;

        while(poll(fds, 2, -1)!=-1){
        char buf[1024];
        memset(buf,0,1024);
        if (fds[0].revents & POLLIN) {
            ssize_t len = SSL_read(ssl_client, buf, sizeof(buf));
            if (len > 0) {
                    printf("Client sent..: %d, %s", (int)len, buf);
                    SSL_write(ssl_server,buf,1024);
                    if(strcmp(buf,"chat_close\n")==0){
                    SSL_shutdown(ssl_client);
                    SSL_free(ssl_server);
                    close(client_socket);
                    close(new_socket);
                    break;
                }
            }
        }
        if (fds[1].revents & POLLIN) {
            ssize_t len = SSL_read(ssl_server,buf,1024);
            if (len > 0) {
                printf("Server sent..: %d, %s", (int)len, buf);
                SSL_write(ssl_client,buf,1024);
                if(strcmp(buf,"chat_close\n")==0){
                    SSL_shutdown(ssl_client);
                    SSL_free(ssl_server);
                    close(new_socket);
                    close(client_socket);
                    break;
                }
            }
        }
    }
  }
  SSL_CTX_free(ctx_client);
  SSL_CTX_free(ctx_server);
}
```

Deliverables:

Details of your chat protocol like its headers and typical message flow. For example, HTTP uses GET/POST/OK methods for message flow between client and server:



```
▸ Frame 38: 1119 bytes on wire (8952 bits), 1119 bytes captured (8952 bits)
▸ Ethernet II, Src: Xensourc_ec:4b:d4 (00:16:3e:ec:4b:d4), Dst: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38)
▸ Internet Protocol Version 4, Src: 172.31.0.3, Dst: 172.31.0.4
▸ Transmission Control Protocol, Src Port: 9600, Dst Port: 52370, Seq: 7251, Ack: 5375, Len: 1053
▾ Transport Layer Security
   ▾ TLSv1.2 Record Layer: Application Data Protocol: Application Data
        Content Type: Application Data (23)
        Version: TLS 1.2 (0x0303)
        Length: 1048
        Encrypted Application Data: 4c6fe3a49f72b32042cc5b17e2d470816dd5501a81ab525a8e369b8ef7b67bd34ef9c26a…
```

Details on how various MITM attacks are realized by Trudy - **(Refer to Diagram in Task 3 and Task 4)**

Credit Statement (1-pager): share an accurate and detailed description of each of the group member's contributions to the assignment in terms of coding, report writing, bug fixes, etc :

Vishesh Kothari:

- ☐ Implementation of Secure_Chat_App (Task 2).
- ☐ Certificate Generation.
- ☐ Make File and Readme doc.

K Saravanan:

- ☐ Implementation of Trudy in Task 3.
- ☐ Bug Fixes in Task 4.
- ☐ PCAP trace collection, Trudy attack diagram flow, and testing on containers.

Joint Operation:

- ☐ Task 4
- ☐ Report Design

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: Vishesh Kothari & K Saravanan
Date: 03/04/2023
Signature: VK & KS