Assignment 7: Secure chat using openssl and MITM attacks

Team members:

1) Ashish B Emmanuel: CS23MTECH11004.
2) Suryansh Gautam : CS23MTECH11020.
3) Anil Kumar Sharma : CS23MTECH13001.

Task 1: Generate keys and certificates

a) Generating root CA certificate:

1) The command to generate an elliptic curve private key of 512 bits using using brainpoolP512r1 curve: *$ openssl ecparam -genkey -name brainpoolP512r1 -out private-key-root.pem*
   Command to derive a public key out of private key: *$ openssl ec -in private-key-root.pem -pubout public-key-root.pem*

```
ubuntu@cs6903-14:~$ openssl ecparam -genkey -name brainpoolP512r1 -out private-key-root.pem
ubuntu@cs6903-14:~$ openssl ec -in private-key-root.pem -pubout -out public-key-root.pem
read EC key
writing EC key
ubuntu@cs6903-14:~$ cat public-key-root.pem
-----BEGIN PUBLIC KEY-----
MIGbMBQGByqGSM49AgEGCSskAwMCCAEBDQOBggAEWUISgDc4rQL0v8o2w7STXqZg
v6JEq8XI5zuQwl7kIwk+txgKsX5bvLrkAAOXV1+7DlAwsK6dZr8ddocwhK6nkFF2
C6SSD6wkjEdeAdmhX3BJYaiSAFqSs8LiWALI6G7XRL8YSLOSTHJnHakzGoF7NsxR
aTeZSdrizJ3QPynTC8E=
-----END PUBLIC KEY-----
ubuntu@cs6903-14:~$ cat private-key-root.pem
-----BEGIN EC PARAMETERS-----
BgkrJAMDAggBAQ0=
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MIHaAgEBBECFqZG8nJ0r2KDzGUNWE8c4xYfTZDIk9jil4GB0kmGxQrou0rtjF5L5
y9ZRiuwUFEAUqZ5/E63eOcOFHOU0w8RMoAsGCSskAwMCCAEBDaGBhQOBggAEWUIS
gDc4rQL0v8o2w7STXqZgv6JEq8XI5zuQwl7kIwk+txgKsX5bvLrkAAOXV1+7DlAw
sK6dZr8ddocwhK6nkFF2C6SSD6wkjEdeAdmhX3BJYaiSAFqSs8LiWALI6G7XRL8Y
SLOSTHJnHakzGoF7NsxRaTeZSdrizJ3QPynTC8E=
-----END EC PRIVATE KEY-----
ubuntu@cs6903-14:~$
```

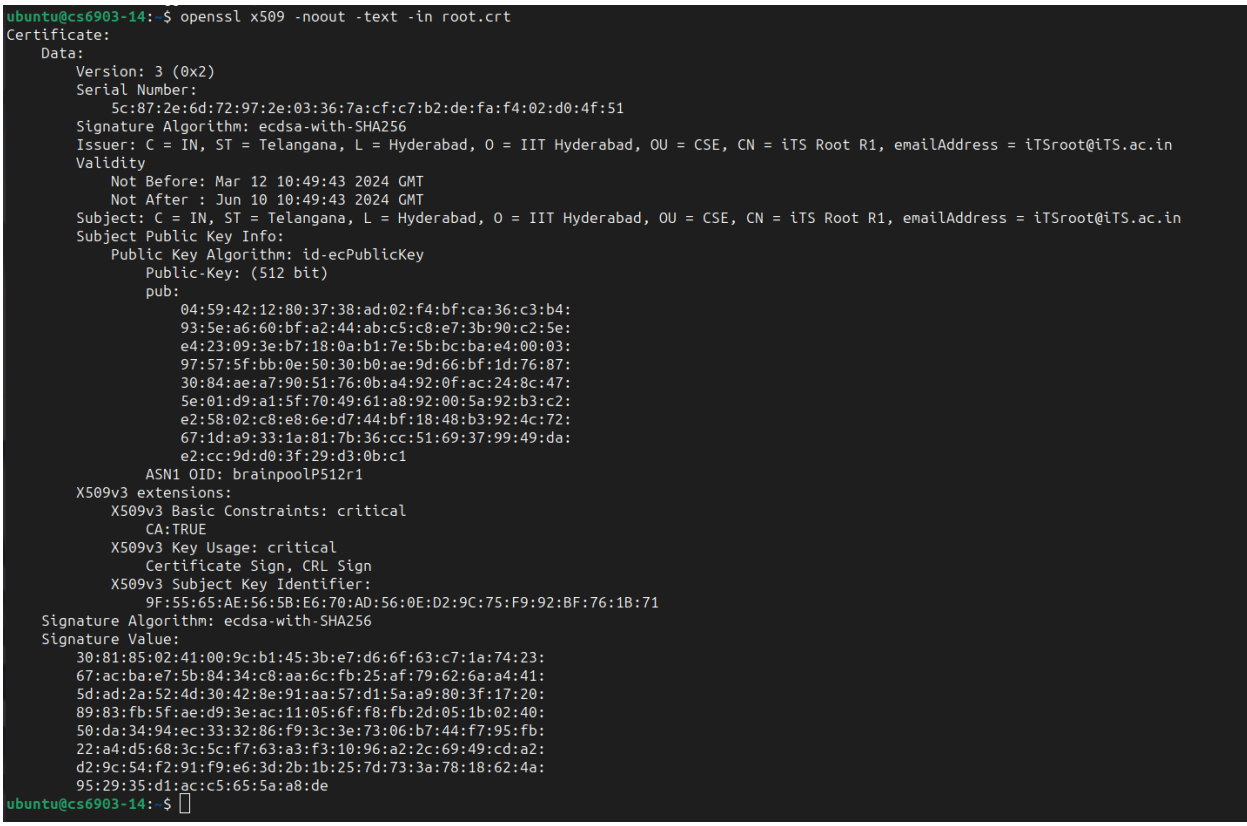
2) Create a configuration file with root CA's information to help in generating the root CA certificate. This also contains information about basic constraints, key usage and certificate version.

```
ubuntu@cs6903-14:~$ vim rootCA.cnf
ubuntu@cs6903-14:~$ cat rootCA.cnf
[req]
default_bits       = 512
distinguished_name = req_distinguished_name
x509_extensions = v3_ca

[req_distinguished_name]
countryName = IN
stateOrProvinceName = Telangana
localityName = Hyderabad
organizationName = IIT Hyderabad
organizationalUnitName = CSE
commonName = iTS Root R1
emailAddress = iTSroot@iTS.ac.in

[v3_ca]
basicConstraints = critical,CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
ubuntu@cs6903-14:~$
```

3) Command to generate a self-signed root certificate: *$ openssl req -new -key private-key-root.pem -x509 -days 90 -out root.crt -config rootCA.cnf.*
    a. -key private-key-root.pem: Indicates to use the key generated in step 1.
    b. -config rootCA.cnf: Indicates to use configuration file generated in step 2.
    c. -new: Indicates to generate a new CSR file.

```
ubuntu@cs6903-14:~$ openssl req -new -key private-key-root.pem -x509 -days 90 -out root.crt -config rootCA.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
IN []:IN
Telangana []:Telangana
Hyderabad []:Hyderabad
IIT Hyderabad []:IIT Hyderabad
CSE []:CSE
iTS Root R1 []:iTS Root R1
iTSroot@iTS.ac.in []:iTSroot@iTS.ac.in
```

4) To display the root CA certificate in human readable form: *$ openssl x509 -noout -text -in root.crt*



```
ubuntu@cs6903-14:~$ openssl x509 -noout -text -in root.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5c:87:2e:6d:72:97:2e:03:36:7a:cf:c7:b2:de:fa:f4:02:d0:4f:51
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = iTS Root R1, emailAddress = iTSroot@iTS.ac.in
        Validity
            Not Before: Mar 12 10:49:43 2024 GMT
            Not After : Jun 10 10:49:43 2024 GMT
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = iTS Root R1, emailAddress = iTSroot@iTS.ac.in
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:59:42:12:80:37:38:ad:02:f4:bf:ca:36:c3:b4:
                    93:5e:a6:60:bf:a2:44:ab:c5:c8:e7:3b:90:c2:5e:
                    e4:23:09:3e:b7:18:0a:b1:7e:5b:bc:ba:e4:00:03:
                    97:57:5f:bb:0e:50:30:b0:ae:9d:66:bf:1d:76:87:
                    30:84:ae:a7:90:51:76:0b:a4:92:0f:ac:24:8c:47:
                    5e:01:d9:a1:5f:70:49:61:a8:92:00:5a:92:b3:c2:
                    e2:58:02:c8:e8:6e:d7:44:bf:18:48:b3:92:4c:72:
                    67:1d:a9:33:1a:81:7b:36:cc:51:69:37:99:49:da:
                    e2:cc:9d:d0:3f:29:d3:0b:c1
                ASN1 OID: brainpoolP512r1
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Subject Key Identifier:
                9F:55:65:AE:56:5B:E6:70:AD:56:0E:D2:9C:75:F9:92:BF:76:1B:71
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:81:85:02:41:00:9c:b1:45:3b:e7:d6:6f:63:c7:1a:74:23:
        67:ac:ba:e7:5b:84:34:c8:aa:6c:fb:25:af:79:62:6a:a4:41:
        5d:ad:2a:52:4d:30:42:8e:91:aa:57:d1:5a:a9:80:3f:17:20:
        89:83:fb:5f:ae:d9:3e:ac:11:05:6f:f8:fb:2d:05:1b:02:40:
        50:da:34:94:ec:33:32:86:f9:3c:3e:73:06:b7:44:f7:95:fb:
        22:a4:d5:68:3c:5c:f7:63:a3:f3:10:96:a2:2c:69:49:cd:a2:
        d2:9c:54:f2:91:f9:e6:3d:2b:1b:25:7d:73:3a:78:18:62:4a:
        95:29:35:d1:ac:c5:65:5a:a8:de
ubuntu@cs6903-14:~$
```

b) <u>Generating intermediate CA certificate.</u>

1) Command to generate an RSA private key of 4096 bits: *$ openssl genrsa -out private-key-inter.pem 4096.*
   Derive public key from the private key: *$ openssl rsa -in private-key-inter.pem -pubout -out public-key-inter.pem*

```
ubuntu@cs6903-14:~$ openssl genrsa -out private-key-inter.pem 4096
ubuntu@cs6903-14:~$ openssl rsa -in private-key-inter.pem -pubout -out public-key-inter.pem
writing RSA key
ubuntu@cs6903-14:~$ cat private-key-inter.pem
-----BEGIN PRIVATE KEY-----
MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQCU5rY8WHpvZ8Xv
eJ/RfglG8BWHXhiE9F4ep7x38MIThYY53v7A0fGGwzY6LU8qSHDLWkAgUNA469Ho
JXI9HzfHqa5Hxx30Ki0blQgGbfvZrWrU8SfP44HXel/GFc5052M/T4lE9Ntbrpuh
xujs3kx6noJ+Ot7pDQcUYaITkw4XNmdQunGB5uZSR3rGqJrbI/vg/LqwzZYwJxEx
zB7/qIgChkEzPAPtXQCQ2UQQl2q5UD+rEIpYLdWb/pEiFTVCegn4UbMjU/H/Y9tu
Up2bt7vSTrDtEz+wACW7Iez4dE2Qb2JbHxXq0XSc7iR3xQcZKIbLc+vlMpFCNb0C
VcAdeMW7eukpN+ZZwDeSWk/s9LJOC6Q6jYh6bymd1lhiIatM1+oLZc5sZOJla7lh
dQpdHzFR15r8L7gz8I6dDQTZ4xY4EK9A8ZFCes+uuVbCuXTLmMtNV+wBMTJ+9kEt
9tzbaI6qwoFi+6lJ31N1QXpld/wXvuZiilIucxa4CfyNm0z+UHCgGjDKUALdjo50
DSDPzTGypyodzD2Pw3n4STabFjqRWakNBwY1FaZVhX76kzPb0vmxfCNWiVcgxYeu
9GKdzTxuTsqjBUEAwrnFVwuQv9I6qAkN1hWa30BfymstNTF/NhfLVUyb3g1Vj0ou
kbr2DGGGpGorZj6nYOCYN5UvB2SQowIDAQABAoICAEhPdE+A/lBs2mRttVtEtFel
TycyjIDAtZZ4id/LZUyaqvEeuKPGMN9rnMGK8GeNz20J0pt+82e5KfJtgPlq2ClX
30GPUp4Y/1fmO4P31HdFLtzjkhx0lUEh9jHZtC30kq+TvgtlR8PR21WopcQmPojL
wn5vs4IE6Bu/QDj2UMD4FBtRt77IdbCecuLXzYA/jNoZkcY7yMRwrHp88M7iNs7X
CQHLEvMqF5TAVjwidFszPAAko7Kn65T6ftvI5dj3MDFN58tvwFVw1obQiyashwNe
YLB5CRQ9NhWXJ6AOxv4uAVtUEM480r9FLSOW5NgUZMWX8YaYVI+NTlE8eHQ3ivo/
weKqGRcq+GDYjki0psd+2AVH55fnZTAl+ZI4eiX7XMvvcOaC0UfyDkUYqSUR+nEi
JBiZ54rh/fid65KZWaLP+Kr8DC7E0VKcFVLDRzW1n8uMXGwL6GYfYx6yEGVfVeRp
vQ/x2FdZi5m7zOSsF+zHTMe87pu/fDZvsefzfpBBs+iJV149veD+9Xa/ODFGwWGS
36X6FYFXetY8jAv/yHba62DEOLHz3L3gU0WpqTUzb+BLbsHwFDW6vzPWlxVpoEY1
EgckPyg3RE7CsoR/FlP6+guBVDj28/m0cHKBWreBIuJVpHLcypG9u5nA4KyPC1mJ
hLsWFuXnd4Asvr4dDhZ9AoIBAQDJR0HeQV2/UR3Cs/2ZzC9tLS3WBXaKlVQkyRGh
WUYQKqYeVJwd0i1n3cuKGDRsw+WWgGRxih+XG6cf8CyMcfnO0YjEstlPIp6fxkZQ
Ug5rdLz3cHYYr0lFJO/tMft7RpisKzap/19uSdwzJ63q2SG/nkCKvGtqD2LD2BQk
iVA8goymHQCoFUUqW+GlMF8OcDvkFbHgtIghGJQLR5+i7WrXbP0zxySn5D0Wk9ME
HdCiEkVRgSd6E3/JDsJgGXsrB3tTUT5lAPN41yf2laAJO++kCxTc1MeNcBJrXkYD
tw/kPHVjCXarBR9iRqm0HxIrPP5/7jIe/XMFKXT7jAf33ANHAoIBAQC9YhDzrXXQ
w0pNr/IEnX4erAo8adH0R5Zwvh1Y9s3hMgTKR0F/ah/I9fB1C3VSzgzEl0hpqyvi
fkbyPT2qY8fHLlpcY5xIouV3KAhqvqDXPkl2kQpnQor517z+HLJD9RUsyVol8cSO
mzBjphK60tC+0+0rlbVHqiDd2KRjs24MbphKqmAijU+z3J7HlhcwIAi2GHkEaADb
kKdyfgJjrlQFOhsQuVZ1wYjUhCf7VHzFCd4PxHd8UXo7gbUC+XWS2Z6Kr029XJU4
5JTJQ0B9KkdirmfBWzjklPimC847RQKjwfp9gcn2dtfbqGzbDDIQzUUg04gZHuYI
4QcSZQchER3FAoIBAQCu/rAK30P1eT3psPmVBMxbHIOUZw2IMYqFrPcSE/7xQVNK
XX+aAckOW+yBtMenpK3wqtIpUNBkXVtxTiw9Mz7gjJZfhaufvNayzP0HCkCVNlqw
uutCTOvkHNgonmMp69gDIFMXe/rpdXfoVeBEpeFMqvcsZ+LvxE63PR9OIMBMy6gj
KGYQvOV0jYviRervL68yf5Y2al9MfwT/xZbKXaadg5mkschx2+MsxPsawh1JO59e
3G6JGXjpQlbGnIDUiV20s5cO08Pa1R1eqvoH7T9NnYDW+nxAwSaSiNVQl5FOey4I
HAs7Z2nA1PTosHbodtBCQ8L55Jti6odVyIqI+ZUzAoIBAQCWCWhmtWjegKmQTLFP
iLqPcDsXy7pDbqe4IFY90iIfpDH5U2SHnjsaSgY7pF1JweaJpVUAKlU1NlfcuDnv
kOeIXKfIBD6fqTZkJXotn2Lrw/QNlDEKx8Q01f9rzq5IepVRCyB5aUYQuNAoUj0K
```

```
z0wI2SVoIW48Ye1wF3BGspC5zgc2ganQsiynYJMFI29Ssl1u+Uccp8cVB7pDj3iL
cPUFAoIBAQCtd5ZBNa/R8PwT5CXbGNvieOYsx9+yjDuOKP8msY2d4NmySWCeV2sO
oERTtbeIv3wDCCZRU5KjMjO3s/ZPrIXgCK8FtpxG3PgVRWjvzvqER+3rUfgm5Crv
74FcNURM+j/ycgUXMtmskSDgRULZ+BGau+7EhE0rd2bRXpI/PsZsZhCcNy+rwgw/
nUuq3CiaoX6JbIkwSc8FlLgay8GLkxuGsbYg4lU2pd0bDHCidWNCd/3KwNCD/KVJ
hmgAQ3qaUzAfM0q0c0Haw0ufXO8Ii4QpaxokR5//Y3RgQD0Y6NqM+/IIqHw1+t0B
B3wrSCPHalGzrEY4uKmXOLum5QlPaL+e
-----END PRIVATE KEY-----
ubuntu@cs6903-14:~$ cat public-key-inter.pem
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAlOa2PFh6b2fF73if0X4J
RvAVh14YhPReHqe8d/DCE4WGOd7+wNHxhsM2Oi1PKkhwy1pAIFDQOOvR6CVyPR83
x6muR8cd9CotG5UIBm372a1q1PEnz+OB13pfxhXOdOdjP0+JRPTbW66bocbo7N5M
ep6Cfjre6Q0HFGGiE5MOFzZnULpxgebmUkd6xqia2yP74Py6sM2WMCcRMcwe/6iI
AoZBMzwD7V0AkNlEEJdquVA/qxCKWC3Vm/6RIhU1QnoJ+FGzI1Px/2PbblKdm7e7
0k6w7RM/sAAluyHs+HRNkG9iWx8V6tF0nO4kd8UHGSiGy3Pr5TKRQjW9AlXAHXjF
u3rpKTfmWcA3klpP7PSyTgukOo2Iem8pndZYYiGrTNfqC2XObGTiZWu5YXUKXR8x
Udea/C+4M/COnQ0E2eMWOBCvQPGRQnrPrrlWwrl0y5jLTVfsATEyfvZBLfbc22iO
qsKBYvupSd9TdUF6ZXf8F77mYopSLnMWuAn8jZtM/lBwoBowylAC3Y6OdA0gz80x
sqcqHcw9j8N5+Ek2mxY6kVmpDQcGNRWmVYV++pMz29L5sXwjVolXIMWHrvRinc08
bk7KowVBAMK5xVcLkL/SOqgJDdYVmt9AX8prLTUxfzYXy1VMm94NVY9KLpG69gxh
hqRqK2Y+p2DgmDeVLwdkkKMCAwEAAQ==
-----END PUBLIC KEY-----
ubuntu@cs6903-14:~$
```

2) Create a configuration file which will be used to generate a CSR.

```
ubuntu@cs6903-14:~$ vim interCA.cnf
ubuntu@cs6903-14:~$ cat interCA.cnf
[req]
default_bits       = 4096
distinguished_name = req_distinguished_name
x509_extensions = v3_ca

[req_distinguished_name]
countryName = IN
stateOrProvinceName = Telangana
localityName = Hyderabad
organizationName = IIT Hyderabad
organizationalUnitName = CSE
commonName = iTS CA 1R3
emailAddress = iTSinter@iTS.ac.in

[v3_ca]
ubuntu@cs6903-14:~$
```

This configuration file doesn't define basic constraints and key usage as did the root configuration file as it will be defined by the root CA which signs the certificate for intermediate CA.

3) Command to generate a CSR: *$ openssl req -new -key private-key-inter.pem -out inter-csr.csr -config interCA.cnf*
    a. -key private-key-inter.pem: Indicates to use the key generated in step 1.
    b. -config interCA.cnf: Indicates to use configuration file generated in step 2.
    c. -new: Indicates to generate a new CSR file.

```
ubuntu@cs6903-14:~$ openssl req -new -key  private-key-inter.pem  -out inter-csr.csr -config interCA.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
IN []:IN
Telangana []:Telangana
Hyderabad []:Hyderabad
IIT Hyderabad []:IIT Hyderabad
CSE []:CSE
iTS CA 1R3 []:iTS CA 1R3
iTSinter@iTS.ac.in []:iTSinter@iTS.ac.in
```

4) CSR in human readable form

```
ubuntu@cs6903-14:~$ openssl req -in inter-csr.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = iTS CA 1R3, emailAddress = iTSinter@iTS.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:94:e6:b6:3c:58:7a:6f:67:c5:ef:78:9f:d1:7e:
                    09:46:f0:15:87:5e:18:84:f4:5e:1e:a7:bc:77:f0:
                    c2:13:85:86:39:de:fe:c0:d1:f1:86:c3:36:3a:2d:
                    4f:2a:48:70:cb:5a:40:20:50:d0:38:eb:d1:e8:25:
                    72:3d:1f:37:c7:a9:ae:47:c7:1d:f4:2a:2d:1b:95:
                    08:06:6d:fb:d9:ad:6a:d4:f1:27:cf:e3:81:d7:7a:
                    5f:c6:15:ce:74:e7:63:3f:4f:89:44:f4:db:5b:ae:
                    9b:a1:c6:e8:ec:de:4c:7a:9e:82:7e:3a:de:e9:0d:
                    07:14:61:a2:13:93:0e:17:36:67:50:ba:71:81:e6:
                    e6:52:47:7a:c6:a8:9a:db:23:fb:e0:fc:ba:b0:cd:
                    96:30:27:11:31:cc:1e:ff:a8:88:02:86:41:33:3c:
                    03:ed:5d:00:90:d9:44:10:97:6a:b9:50:3f:ab:10:
                    8a:58:2d:d5:9b:fe:91:22:15:35:42:7a:09:f8:51:
                    b3:23:53:f1:ff:63:db:6e:52:9d:9b:b7:bb:d2:4e:
                    b0:ed:13:3f:b0:00:25:bb:21:ec:f8:74:4d:90:6f:
                    62:5b:1f:15:ea:d1:74:9c:ee:24:77:c5:07:19:28:
                    86:cb:73:eb:e5:32:91:42:35:bd:02:55:c0:1d:78:
                    c5:bb:7a:e9:29:37:e6:59:c0:37:92:5a:4f:ec:f4:
                    b2:4e:0b:a4:3a:8d:88:7a:6f:29:9d:d6:58:62:21:
                    ab:4c:d7:ea:0b:65:ce:6c:64:e2:65:6b:b9:61:75:
                    0a:5d:1f:31:51:d7:9a:fc:2f:b8:33:f0:8e:9d:0d:
                    04:d9:e3:16:38:10:af:40:f1:91:42:7a:cf:ae:b9:
                    56:c2:b9:74:cb:98:cb:4d:57:ec:01:31:32:7e:f6:
                    41:2d:f6:dc:db:68:8e:aa:c2:81:62:fb:a9:49:df:
                    53:75:41:7a:65:77:fc:17:be:e6:62:8a:52:2e:73:
                    16:b8:09:fc:8d:9b:4c:fe:50:70:a0:1a:30:ca:50:
                    02:dd:8e:8e:74:0d:20:cf:cd:31:b2:a7:2a:1d:cc:
                    3d:8f:c3:79:f8:49:36:9b:16:3a:91:59:a9:0d:07:
                    06:35:15:a6:55:85:7e:fa:93:33:db:d2:f9:b1:7c:
                    23:56:89:57:20:c5:87:ae:f4:62:9d:cd:3c:6e:4e:
                    ca:a3:05:41:00:c2:b9:c5:57:0b:90:bf:d2:3a:a8:
                    09:0d:d6:15:9a:df:40:5f:ca:6b:2d:35:31:7f:36:
                    17:cb:55:4c:9b:de:0d:55:8f:4a:2e:91:ba:f6:0c:
                    61:86:a4:6a:2b:66:3e:a7:60:e0:98:37:95:2f:07:
                    64:90:a3
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
```

```
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        41:47:83:c3:aa:f1:a6:fe:f1:74:cb:c5:96:db:86:2d:83:8c:
        cc:d1:ed:dc:88:a8:7f:29:03:00:94:7c:59:e0:0c:e8:ed:01:
        b2:43:b6:cf:67:f1:91:36:70:17:f4:61:97:b9:56:7c:6b:ab:
        3a:67:66:b6:95:e8:0b:26:46:35:6a:8e:5f:fb:6a:63:51:b2:
        a2:2f:84:15:c1:ba:b6:ba:d0:42:45:a0:d9:85:3b:a8:e5:97:
        5c:56:0b:08:28:76:89:9c:ce:5c:e2:7e:01:6b:d4:c7:ab:8b:
        16:b5:f2:85:03:60:e6:67:95:ad:ec:9b:6c:b9:69:dd:8d:5c:
        48:d8:aa:c9:a3:65:0f:22:7e:97:4a:e7:37:d0:5a:b0:58:12:
        eb:d8:02:50:e4:4c:dd:49:84:9f:84:b4:38:97:e6:a9:dd:ac:
        87:1f:6d:41:62:e0:70:cc:f8:84:1c:ff:49:41:5c:7b:05:af:
        46:26:ec:0b:20:b7:38:c8:2b:bb:95:20:75:17:3a:cb:35:cd:
        e1:02:ea:3c:0a:15:80:30:81:12:85:a1:ff:54:24:11:12:95:
        b0:0a:8f:49:34:ac:89:63:b1:cc:0b:1b:59:2f:95:5e:06:9f:
        b7:fc:fd:ac:96:26:1c:65:0a:b8:22:fc:f1:d8:24:16:db:b1:
        43:97:9f:80:f4:ca:47:25:80:7d:0e:96:4f:46:17:89:0d:72:
        94:b3:1d:d8:1d:fa:95:e5:9e:9a:15:56:b1:77:7c:4a:ae:36:
        f5:fa:e2:38:45:ad:6a:3e:d4:53:47:58:f2:60:16:6f:e3:3b:
        03:1c:4d:5d:c7:ae:42:8d:90:33:43:d5:36:f6:b1:72:30:e0:
        73:83:cf:c6:c0:85:db:01:1d:ce:bc:73:11:93:c9:6a:6c:5d:
        46:d0:d1:9f:a7:72:dc:86:e8:0f:2c:01:98:03:cb:d5:d3:88:
        63:6b:31:81:85:37:f0:63:0a:20:48:6b:3a:25:12:8b:14:01:
        ef:45:47:05:b0:da:21:15:19:8b:91:c2:f5:70:88:22:48:8c:
        95:4f:14:f9:0b:9d:5b:9a:03:83:9e:a5:c9:2f:68:88:2b:fb:
        7c:01:ea:74:56:16:6d:ca:44:bd:1d:0d:57:cc:7c:d7:67:cb:
        1d:65:04:6a:01:66:ac:f8:df:dd:c6:53:5e:02:4b:62:a7:79:
        ed:85:c5:94:d7:ee:4a:3a:93:68:b9:bd:c1:d3:54:4e:e9:a5:
        81:10:e0:04:ae:6f:25:44:ad:7e:14:7f:94:c4:ff:40:90:a1:
        28:a6:a5:ac:16:a7:d6:7d:2c:56:d7:5b:c6:09:28:09:2c:ff:
        33:b3:7a:08:56:83:3a:13
ubuntu@cs6903-14:~$
```

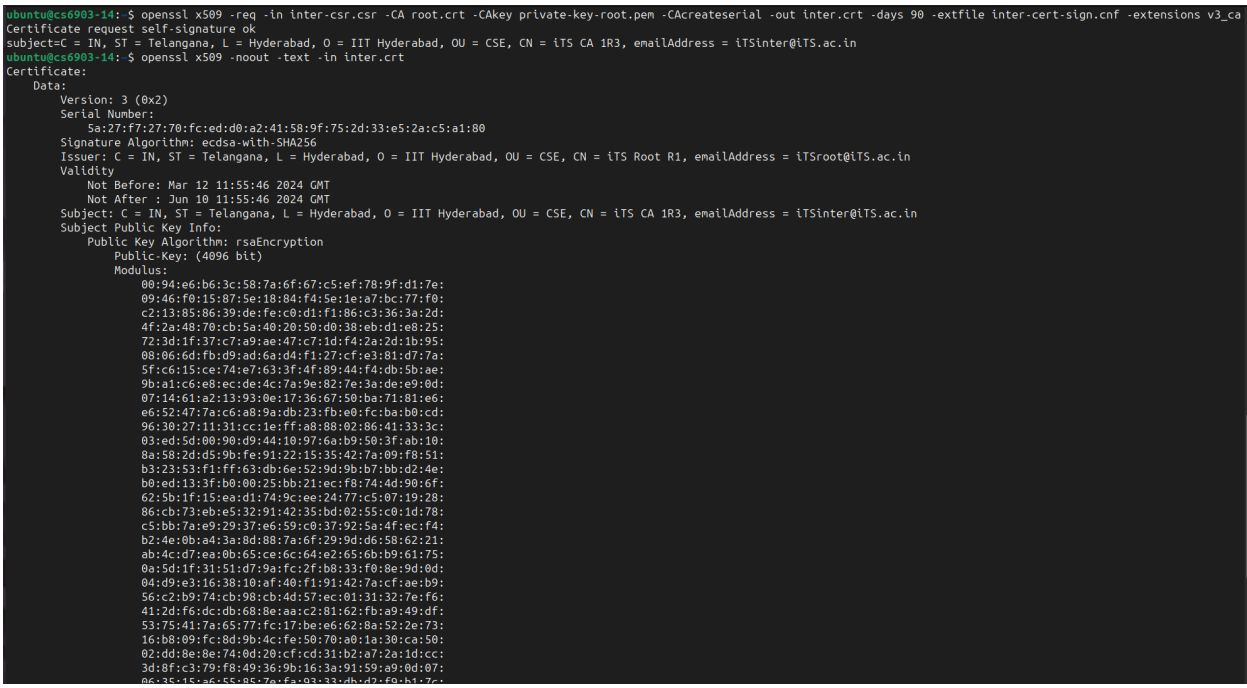
5) Once root CA gets CSR from intermediate CA, it can verify CSR using the command: *$ openssl req -text -verify -in inter-csr.csr*
-verify: tells OpenSSL to verify the signature of CSR. This will ensure that the CSR has been correctly signed by the private key corresponding to the public key in the CSR. This will verify that the CSR is indeed sent by intermediate CA and not any other malicious enitty.

```
ubuntu@cs6903-14:~$ openssl req -text -noout -verify -in inter-csr.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = iTS CA 1R3, emailAddress = iTSinter@iTS.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:94:e6:b6:3c:58:7a:6f:67:c5:ef:78:9f:d1:7e:
                    09:46:f0:15:87:5e:18:84:f4:5e:1e:a7:bc:77:f0:
                    c2:13:85:86:39:de:fe:c0:d1:f1:86:c3:36:3a:2d:
                    4f:2a:48:70:cb:5a:40:20:50:d0:38:eb:d1:e8:25:
                    72:3d:1f:37:c7:a9:ae:47:c7:1d:f4:2a:2d:1b:95:
                    08:06:6d:fb:d9:ad:6a:d4:f1:27:cf:e3:81:d7:7a:
                    5f:c6:15:ce:74:e7:63:3f:4f:89:44:f4:db:5b:ae:
                    9b:a1:c6:e8:ec:de:4c:7a:9e:82:7e:3a:de:e9:0d:
                    07:14:61:a2:13:93:0e:17:36:67:50:ba:71:81:e6:
                    e6:52:47:7a:c6:a8:9a:db:23:fb:e0:fc:ba:b0:cd:
                    96:30:27:11:31:cc:1e:ff:a8:88:02:86:41:33:3c:
                    03:ed:5d:00:90:d9:44:10:97:6a:b9:50:3f:ab:10:
                    8a:58:2d:d5:9b:fe:91:22:15:35:42:7a:09:f8:51:
                    b3:23:53:f1:ff:63:db:6e:52:9d:9b:b7:bb:d2:4e:
                    b0:ed:13:3f:b0:00:25:bb:21:ec:f8:74:4d:90:6f:
```

6) Create a configuration file at root CA with basic constraints and key usage for intermediate certificate.

```
ubuntu@cs6903-14:~$ vim inter-cert-sign.cnf
ubuntu@cs6903-14:~$ cat inter-cert-sign.cnf
[v3_ca]
basicConstraints = critical,CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
ubuntu@cs6903-14:~$
```

7) Command to sign a certificate signing request (CSR): *$ Openssl x509 -req -in inter-csr.csr -CA root.crt -CAkey private-key-root.pem -CAcreateserial -out inter.crt -days 90 -extfile inter-cert-sign.cnf -extensions v3_ca*
    a. *-req:* indicates that the input file is csr
    b. *-in inter-csr.csr:* specifies the file that is CSR
    c. *-CA root.crt:* specifies root CA certificate
    d. *-CAkey private-key-root.pem:* specifies root CA private key.
    e. *-CAcreateserial:* this specifies openssl to generate a serial number for this certificate.
    f. *-extfile inter-cert-sign.cnf:* this option specifies the configuration file which specifies the extensions for the certificate.



```
                    16:b8:09:fc:8d:9b:4c:fe:50:70:a0:1a:30:ca:50:
                    02:dd:8e:8e:74:0d:20:cf:cd:31:b2:a7:2a:1d:cc:
                    3d:8f:c3:79:f8:49:36:9b:16:3a:91:59:a9:0d:07:
                    06:35:15:a6:55:85:7e:fa:93:33:db:d2:f9:b1:7c:
                    23:56:89:57:20:c5:87:ae:f4:62:9d:cd:3c:6e:4e:
                    ca:a3:05:41:00:c2:b9:c5:57:0b:90:bf:d2:3a:a8:
                    09:0d:d6:15:9a:df:40:5f:ca:6b:2d:35:31:7f:36:
                    17:cb:55:4c:9b:de:0d:55:8f:4a:2e:91:ba:f6:0c:
                    61:86:a4:6a:2b:66:3e:a7:60:e0:98:37:95:2f:07:
                    64:90:a3
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Subject Key Identifier:
                1A:E3:F0:33:34:FB:F8:7B:7B:38:B1:6C:19:E8:34:57:55:CB:E7:D2
            X509v3 Authority Key Identifier:
                9F:55:65:AE:56:5B:E6:70:AD:56:0E:D2:9C:75:F9:92:BF:76:1B:71
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:81:85:02:40:7a:8e:87:4c:29:77:17:de:fa:eb:eb:eb:93:
        93:00:94:8b:b3:16:71:b3:c1:52:e1:c3:47:ba:3a:28:1a:db:
        91:1d:20:9e:9d:f4:78:77:b6:7a:d1:fb:6e:d2:d0:8a:04:75:
        dd:6d:b9:42:df:81:dd:18:cb:bd:d4:52:98:ea:f8:02:41:00:
        a2:09:01:e9:f1:24:93:bc:0d:86:fc:fe:b1:27:4a:ec:87:a2:
        2f:29:0d:dc:ae:09:b9:9e:15:2b:8a:6b:39:63:5c:3a:46:a2:
        46:67:8c:cf:8c:0c:30:b5:61:4c:a9:2d:07:ba:ea:79:c5:8a:
        6a:0d:e0:62:c0:e1:c7:85:3c:06
ubuntu@cs6903-14:~$
```

8) Command to verify whether it's a valid certificate and is indeed signed by the root certificate: *$ openssl verify -CAfile root.crt inter.crt*

```
ubuntu@cs6903-14:~$ openssl verify -CAfile root.crt inter.crt
inter.crt: OK
ubuntu@cs6903-14:~$
```

c) Generating certificate for Alice:

1) Command to generate RSA private key for Alice: *$ openssl genrsa -out alice-private-key.pem*
   Derive public key out of private key: *$ openssl rsa -in alice-private-key.pem -pubout -out alice-public-key.pem*

```
root@alice1:~/cert# openssl rsa -in alice-private-key.pem -pubout -out alice-public-key.pem
writing RSA key
root@alice1:~/cert# cat alice-private-key.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQCzQSvlXoUdu29U8ef7O7lopdU+P7aIbmGAtJ0k60WAmdQjPsdQ
HV/tqVatPBEWiOndDiLUlBH1BgFeCLQ68Pbrt6/SgObhX/fQYRZ2g8OfFZt95wnK
P1Q/vPQssCQxwIOT1DHwqrXJyckZGLhhFAvGpmxTIoXA4U3R6FOyRp+oJQIDAQAB
AoGAZUT/aKECsW5wElxsQ8pVECMIGxfhpqg9m3nugaoNZP8DEzpnXgKKbHlLx6j0
und1L/bJhLF2h6fx488lC1uRE10y48F1euiwXN4lN1YMYsodhSA1tkO5oZ0KvwPa
rFasoW1XNybfogk1cuqtNbCKmwnO69nIzxBxjJqticMEEQ0CQQDtFl8zzWM77bQS
J+9NvGyppRL0HhFxZc+x2sD5Tgw/VduffsCH55r54jgg+AeL4jIO0IctstDaKZ+w
FchBoRILAkEAwY3Hir0L9S4E8tAKOGpa9NzORFugLgT4BzJ6Q7kub3bpvyO1A2qM
l1VDjNz3G/rl+hoxBiAyzc+l/0gDR8Q8jwJATz9VHymd6+AueosisCc1YxcsWLDc
P706SUbHjg2KhnkaahuiG0tNQA1fmZWOYDGBbfZB2QKWUPXszJzbTSSbtQJBAK6h
txnMO4+5N6nEXpRMEUjAODKg4MLkpMwJlRqpZ+YVygw0sUslxTd9LHdZnnLWSsz+
+KBoSK483a6YUK1dtq0CQQDeVDswGDh0Qwk697RCarjPufYaWOTG3bcj1bnpJ9g+
zeuMu5mCTe/eiEiPiD5GeCbUWxCxo9jKZJr7qRaalhn/
-----END RSA PRIVATE KEY-----
root@alice1:~/cert# cat alice-public-key.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCzQSvlXoUdu29U8ef7O7lopdU+
P7aIbmGAtJ0k60WAmdQjPsdQHV/tqVatPBEWiOndDiLUlBH1BgFeCLQ68Pbrt6/S
gObhX/fQYRZ2g8OfFZt95wnKP1Q/vPQssCQxwIOT1DHwqrXJyckZGLhhFAvGpmxT
IoXA4U3R6FOyRp+oJQIDAQAB
-----END PUBLIC KEY-----
root@alice1:~/cert#
```

2) Create a configuration file for Alice's CSR. Here we set the size of the public key to be 1024 bits.

```
root@alice1:~/home/ubuntu/secure_chat_app/Task_1/alice# cat alice.cnf
[req]
default_bits=1024
distinguished_name=req_distinguished_name
x509_extensions=v3_ca

[req_distinguished_name]
countryName=IN
stateOrProvinceName=Telangana
localityName=Hyderabad
organizationName=IIT Hyderabad
organizationalUnitName=CSE
commonName=Alice1.com
emailAddress = cs23mtech11004@iith.ac.in

[v3_ca]
```

3) Command to generate CSR for Alice: *$ openssl req -new -key alice-private-key.pem -out alice.csr -config alice.cnf*

```
root@alice1:~/cert# openssl req -in alice.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = "IIT Hyderabad ", OU = CSE, CN = Alice1.com, emailAddress = cs23mtech11004@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (1024 bit)
                Modulus:
                    00:b3:41:2b:e5:5e:85:1d:bb:6f:54:f1:e7:fb:3b:
                    b9:68:a5:d5:3e:3f:b6:88:6e:61:80:b4:9d:24:eb:
                    45:80:99:d4:23:3e:c7:50:1d:5f:ed:a9:56:ad:3c:
                    11:16:88:e9:dd:0e:22:d4:94:11:f5:06:01:5e:08:
                    b4:3a:f0:f6:eb:b7:af:d2:80:e6:e1:5f:f7:d0:61:
                    16:76:83:c3:9f:15:9b:7d:e7:09:ca:3f:54:3f:bc:
                    f4:2c:b0:24:31:c0:83:93:d4:31:f0:aa:b5:c9:c9:
                    c9:19:18:b8:61:14:0b:c6:a6:6c:53:22:85:c0:e1:
                    4d:d1:e8:53:b2:46:9f:a8:25
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         09:73:92:0e:56:54:71:4b:53:6c:64:4c:b7:b6:c6:3b:9b:8b:
         af:e4:45:c2:27:db:7c:ba:08:0d:e4:4d:f5:55:c3:99:02:9e:
         a9:d7:f9:49:15:d2:34:86:1b:86:c1:cc:94:3c:13:3c:54:bf:
         98:a5:c6:82:ea:d2:1e:43:22:ce:33:ab:de:c6:68:ef:cd:9a:
         c6:a7:30:16:33:e0:c2:4a:c7:b5:95:6d:e1:84:a1:f2:ab:a4:
         5c:d4:48:42:be:8d:73:eb:04:4d:9f:1f:05:41:7c:6d:3f:73:
         e3:c6:3a:e3:f0:a4:04:37:e3:f8:f0:68:8e:a1:6e:5a:28:db:
         7f:cc
root@alice1:~/cert#
```

4) Once the intermediate gets the CSR of Alice, he will verify whether it's indeed from Alice using the command: $ openssl req -text -verify -in alice.csr

```
ubuntu@cs6903-0-4:~/secure_chat_app/Task_1/alice$ openssl req -text -verify -in alice.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Alice1.com, emailAddress = cs23mtech11004@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:ca:bc:03:2d:e6:82:b3:7e:69:95:5e:14:76:12:
                    78:9e:01:40:8f:85:f7:ac:95:14:38:94:72:bb:99:
```

5) Configuration file at intermediate CA which defines basic constraints and key usage for Alice.

```
ubuntu@cs6903-14:~/inter$ vim cert.cnf
ubuntu@cs6903-14:~/inter$ cat cert.cnf
[v3_ca]
basicConstraints = critical,CA:FALSE
keyUsage = critical,digitalSignature,keyEncipherment,keyAgreement
ubuntu@cs6903-14:~/inter$
```

6) Command to sign Alice's certificate: Command to sign a certificate signing request (CSR): *$ openssl x509 -req -in alice.csr -CA inter.crt -CAkey private-key-inter.pem -CAcreateserial -out alice.crt -days 90 -extfile cert.cnf -extensions v3_ca*
    a. *-req:* indicates that the input file is csr
    b. *-in alice.csr:* specifies the file that is CSR
    c. *-CA inter.crt:* specifies intermediate CA certificate
    d. *-CAkey private-key-inter.pem:* specifies intermediate CA private key.

e. *-CAcreateserial:* this specifies openssl to generate a serial number for this certificate.
f. *-extfile cert.cnf:* this option specifies the configuration file which specifies the extensions for the certificate.

Command to create a chain of certificate for Alice: *$ cat alice.crt inter.crt root.crt > alice_chain.crt*

```
ubuntu@cs6903-14:~/inter$ openssl x509 -req -in alice.csr -CA inter.crt -CAkey private-key-inter.pem -CAcreateserial -out alice.crt -days 90 -extfile cert.cnf -extensions v3_ca
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = "IIT Hyderabad ", OU = CSE, CN = Alice1.com, emailAddress = cs23mtech11004@iith.ac.in
ubuntu@cs6903-14:~/inter$ cat alice.crt inter.crt root.crt > alice_chain.crt
ubuntu@cs6903-14:~/inter$ lxc file push /home/ubuntu/inter/alice_chain.crt alice1/root/cert/alice_chain.crt
ubuntu@cs6903-14:~/inter$
```

7) Once Alice receives the signed certificate, She can verify that the certificate is indeed signed by the intermediate certificate along with the chain of trust using the command: *$ openssl verify -CAfile root.crt -untrusted inter.crt alice_chain.crt*

```
root@alice1:~/cert# openssl verify -CAfile root.crt -untrusted inter.crt alice_chain.crt
alice_chain.crt: OK
root@alice1:~/cert#
```

8) Alice's certificate in human readable form.

```
root@alice1:~/cert# openssl x509 -noout -text -in alice.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0a:6d:a8:99:17:4f:42:c4:3d:04:e8:f8:ce:4f:08:ba:be:f8:83:74
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = iTS CA 1R3, emailAddress = iTSinter@iTS.ac.in
        Validity
            Not Before: Mar 13 08:08:52 2024 GMT
            Not After : Jun 11 08:08:52 2024 GMT
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = "IIT Hyderabad ", OU = CSE, CN = Alice1.com, emailAddress = cs23mtech11004@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (1024 bit)
                Modulus:
                    00:b3:41:2b:e5:5e:85:1d:bb:6f:54:f1:e7:fb:3b:
                    b9:68:a5:d5:3e:3f:b6:88:6e:61:80:b4:9d:24:eb:
                    45:80:99:d4:23:3e:c7:50:1d:5f:ed:a9:56:ad:3c:
                    11:16:88:e9:dd:0e:22:d4:94:11:f5:06:01:5e:08:
                    b4:3a:f0:f6:eb:b7:af:d2:80:e6:e1:5f:f7:d0:61:
                    16:76:83:c3:9f:15:9b:7d:e7:09:ca:3f:54:3f:bc:
                    f4:2c:b0:24:31:c0:83:93:d4:31:f0:aa:b5:c9:c9:
                    c9:19:18:b8:61:14:0b:c6:a6:6c:53:22:85:c0:e1:
                    4d:d1:e8:53:b2:46:9f:a8:25
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Key Agreement
            X509v3 Subject Key Identifier:
                89:87:08:39:F3:DE:B2:0A:95:86:CD:55:C1:99:CE:09:22:E4:0F:8F
            X509v3 Authority Key Identifier:
                keyid:1A:E3:F0:33:34:FB:F8:7B:7B:38:B1:6C:19:E8:34:57:55:CB:E7:D2

    Signature Algorithm: sha256WithRSAEncryption
         3c:c1:90:67:64:3c:18:69:12:22:5d:e8:74:93:1c:10:fa:99:
         ea:b8:ea:c6:31:d2:00:20:fc:a9:e6:a8:20:1f:42:8e:18:12:
         37:98:39:6d:bd:fe:2b:4a:f0:99:09:ac:64:d6:f7:4b:63:09:
         1e:1f:da:7d:e0:3f:c6:7e:48:ea:7e:64:03:8c:19:b4:de:5e:
         8e:c4:57:ed:bb:60:cf:95:76:c7:9e:d1:2b:63:c1:71:8f:45:
         d6:44:8a:36:ff:11:0d:7b:25:69:d4:d1:5e:70:b8:2b:94:8c:
         69:6b:3e:bb:b9:2b:92:e0:79:14:48:57:5f:c2:58:d3:85:df:
         fe:a1:2d:30:63:b0:59:d3:6b:b0:98:85:74:69:b5:d2:e1:4e:
         2d:5c:14:77:68:fc:70:99:1d:a1:71:8f:96:18:77:b8:73:8e:
```

d) <u>Generating certificate for Bob:</u>

1) To generate a certificate for Bob, we will have to follow the same steps as we did to generate the certificate for Alice except for the key generation phase. The public key generated for Bob is a 256 bit ECC key.

2) Command to generate the ECC private key: *$ openssl ecparam -genkey -name prime256v1 -out bob-private-key.pem*
Derive public key out of private key: *$ openssl ec -in bob-private-key.pem -pubout -out bob-public-key.pem.*

```
root@bob1:~/cert# openssl ecparam -genkey -name prime256v1 -out bob-private-key.pem
root@bob1:~/cert# openssl ec -in bob-private-key.pem -pubout -out bob-public-key.pem
read EC key
writing EC key
root@bob1:~/cert# ls
bob-private-key.pem  bob-public-key.pem  bob.cnf
root@bob1:~/cert#
```

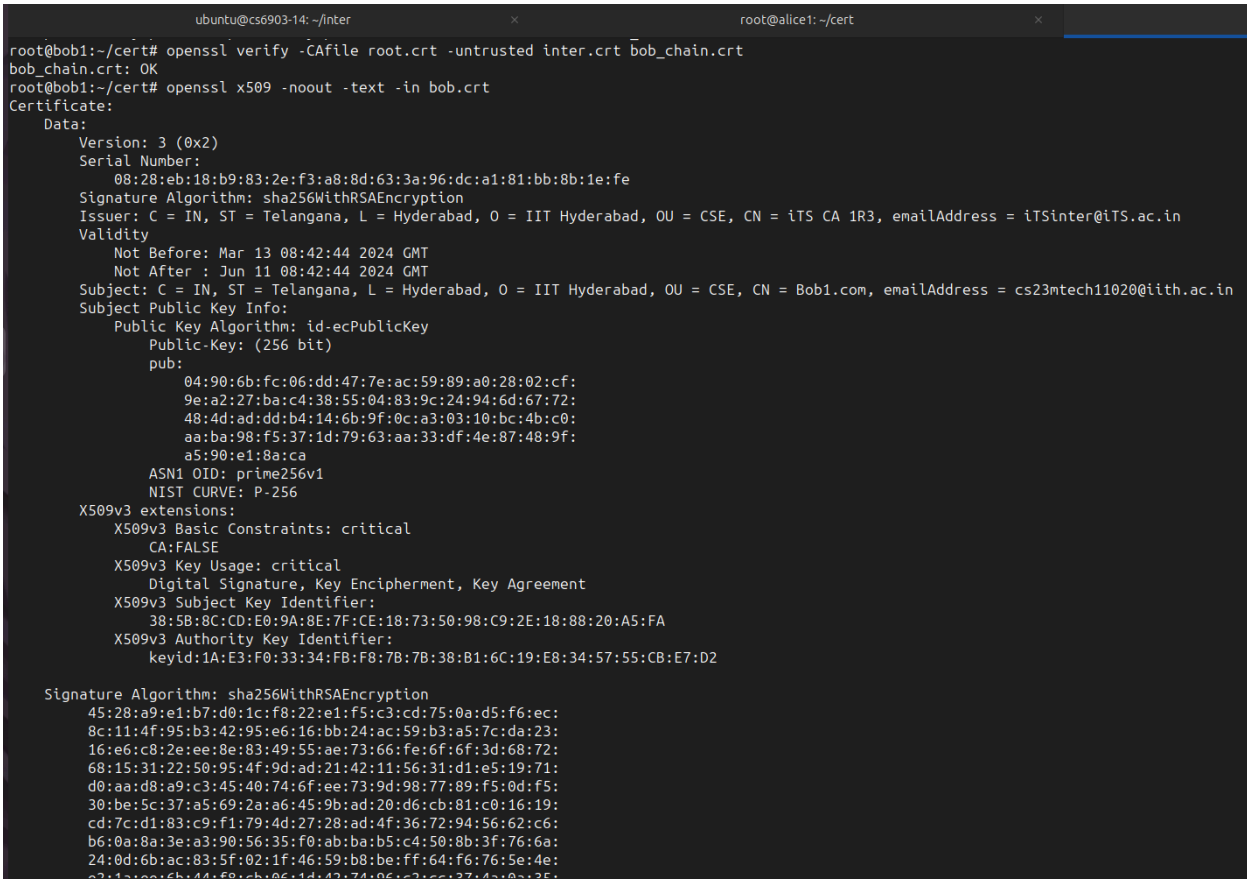
3) Configuration file for generating Bob's CSR. Here we specify the size of the public key to be of 256 bits.

```
root@bob1:~/home/ubuntu/secure_chat_app/Task_1/bob# cat bob.cnf
[req]
default_bits=256
distinguished_name=req_distinguished_name
x509_extensions=v3_ca

[req_distinguished_name]
countryName=IN
stateOrProvinceName=Telangana
localityName=Hyderabad
organizationName=IIT Hyderabad
organizationalUnitName=CSE
commonName=Bob1.com
emailAddress = cs23mtech11020@iith.ac.in

[v3_ca]
```

4) Bob’s CSR in human readable format.

```
root@bob1:~/cert# openssl req -new -key bob-private-key.pem -out bob.csr -config bob.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
IN []:IN
Telangana []:Telangana
Hyderabad []:Hyderabad
IIT Hyderabad []:IIT Hyderabad
CSE []:CSE
Bob1.com []:Bob1.com
cs23mtech11020@iith.ac.in []:cs23mtech11020@iith.ac.in
root@bob1:~/cert# openssl req -in bob.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Bob1.com, emailAddress = cs23mtech11020@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:90:6b:fc:06:dd:47:7e:ac:59:89:a0:28:02:cf:
                    9e:a2:27:ba:c4:38:55:04:83:9c:24:94:6d:67:72:
                    48:4d:ad:dd:b4:14:6b:9f:0c:a3:03:10:bc:4b:c0:
                    aa:ba:98:f5:37:1d:79:63:aa:33:df:4e:87:48:9f:
                    a5:90:e1:8a:ca
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            a0:00
    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:69:b0:ed:76:48:da:71:4e:4f:a9:8f:ca:46:8b:
         c1:5e:3e:c8:06:39:9a:b8:95:b7:e9:9a:58:1b:ec:48:be:9c:
         02:20:45:b2:2c:b4:c1:c0:99:b4:e9:72:dc:69:31:41:61:bd:
         8d:15:90:98:e4:d9:d9:97:79:68:fb:25:22:9d:5e:36
root@bob1:~/cert#
```

5) Bob’s certificate verified to be indeed signed by intermediate CA and in human readable format.



Task 2: Secure Chat App

Design overview:

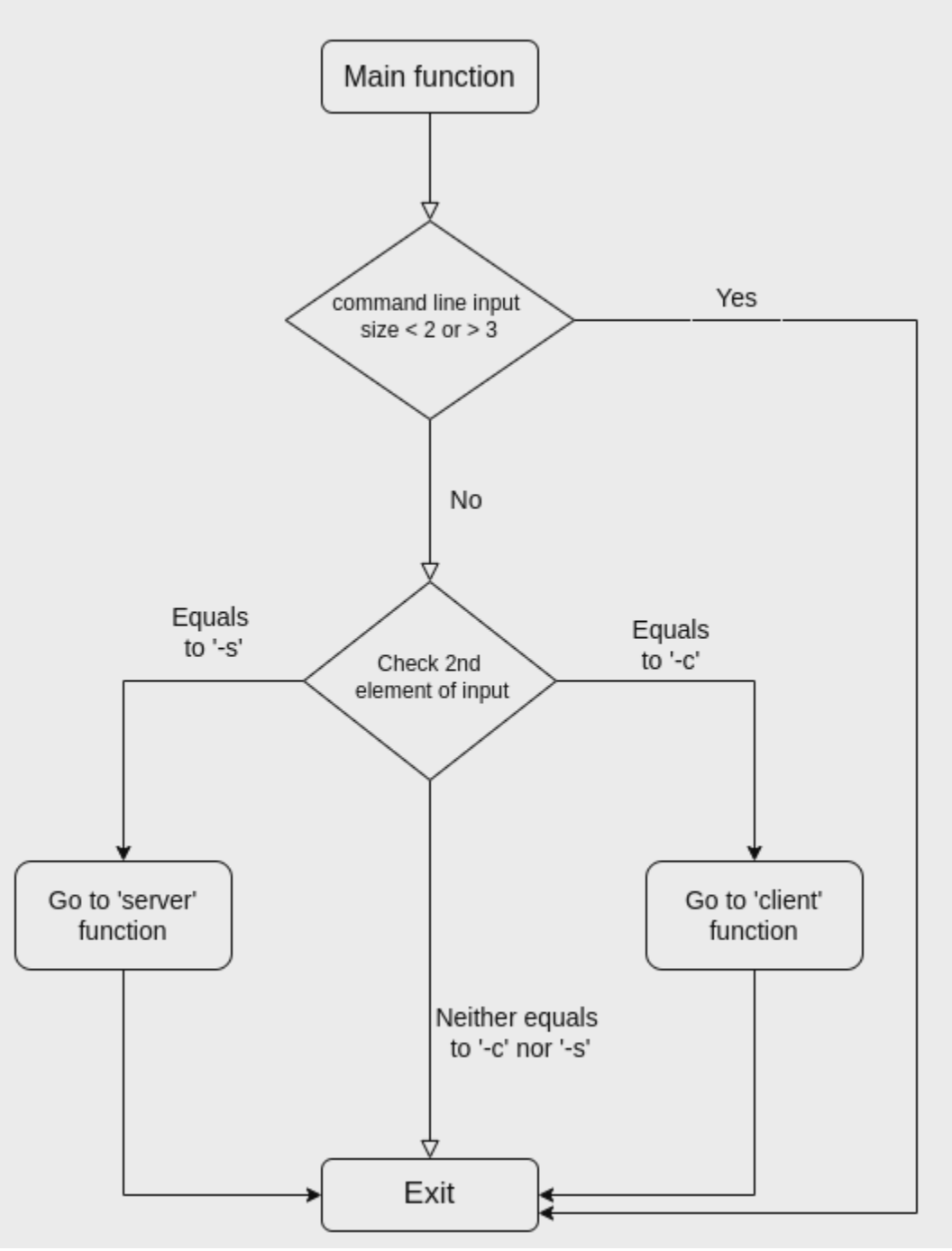
1) The main function takes command line argument:
    a. `secure_chat_app -s`: Program goes into server function.
    b. `secure_chat_app -c <serverhostname>`: Program goes into client function.
2) List of control messages:

	Client's message	Expected reply from server
a.	Chat_hello	chat_ok_reply
b.	chat_START_SSL	chat_START_SSL_ACK

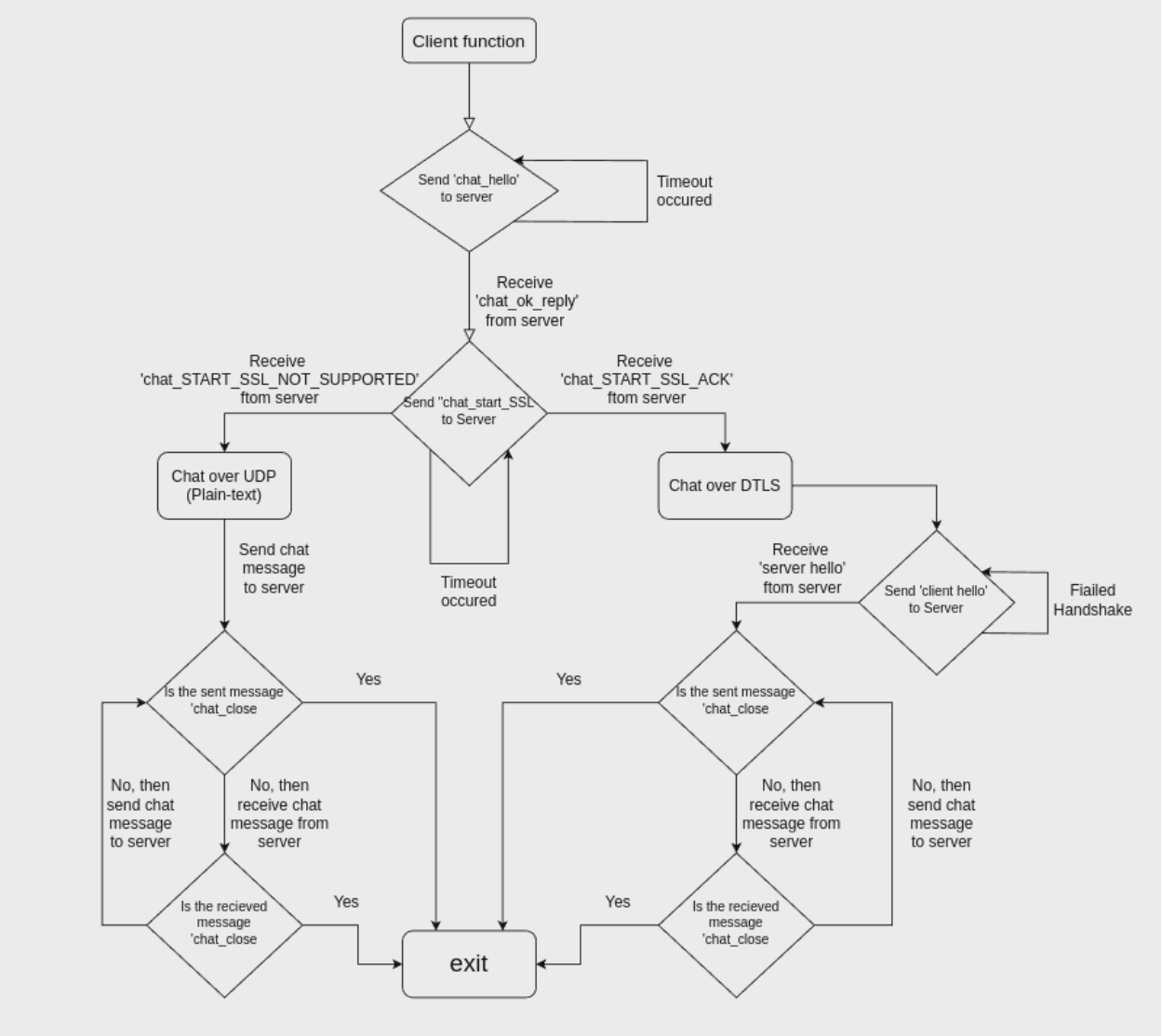
3) Once the client gets 'chat_START_SSL_ACK' response, she initiates DTLS handshake with the server where both exchange each other's certificate and verify it.
4) Before initiating the DTLS handshake, client sets up ciphersuites which provide perfect forward secrecy: ECDHE-RSA-AES256-GCM-SHA384 and ECDHE-ECDSA-AES256-GCM-SHA384
5) Reliability has been implemented for all the control messages including DTLS handshake using timeouts and retransmission. It will handle packet losses in UDP.
6) We have the SSL context in 'SSL_MODE_AUTO_RETRY' mode which specifies SSL read and SSL write to auto-retry when they are interrupted
7) A separate function has been implemented to handle errors and thus, the program handles errors gracefully.
8) The program cleans up everything and frees up memory when it is about to exit.

The flow of the program shown using flow charts

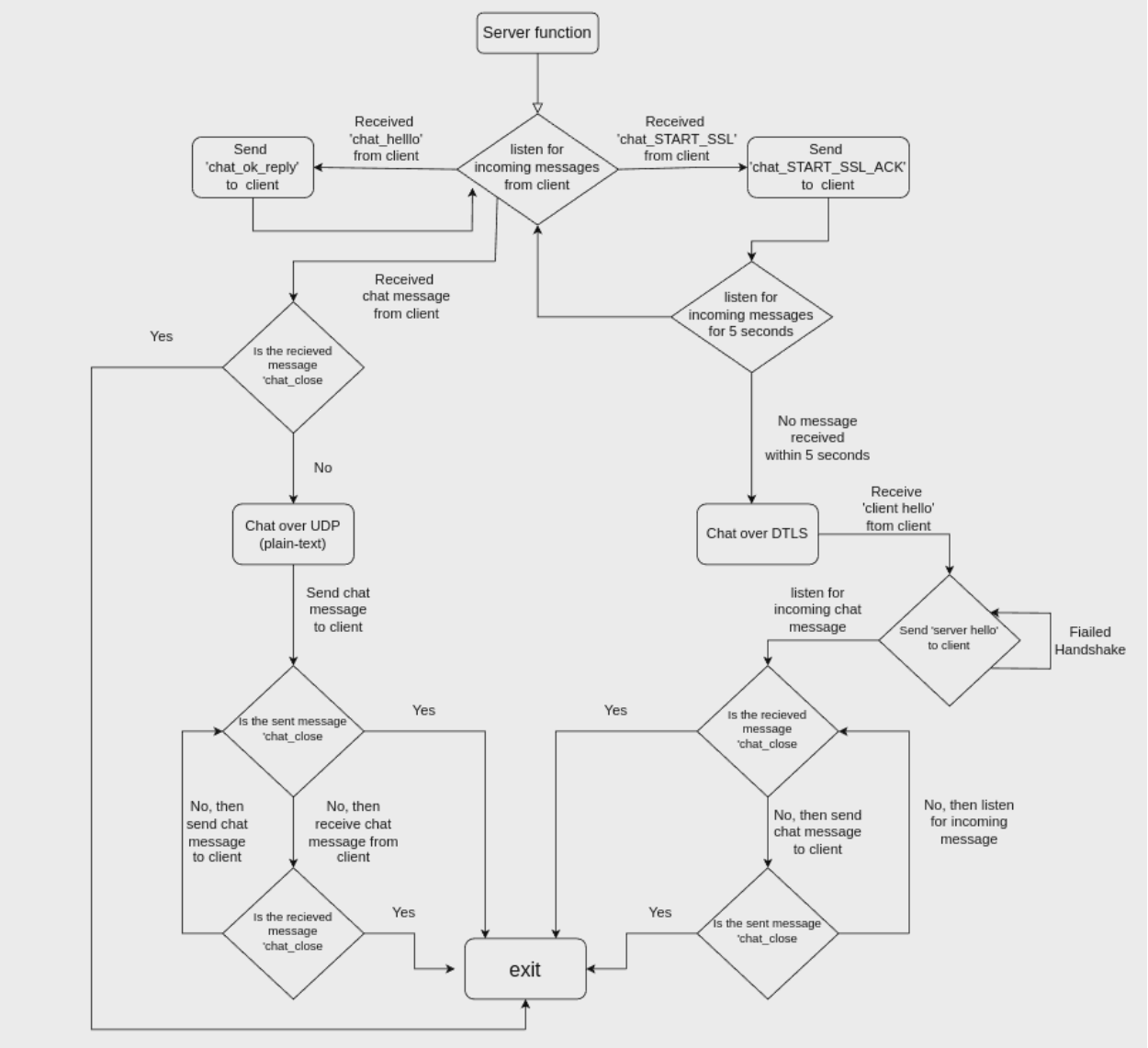
1. **Flowchart of Main function:**



2. Flowchart of client function:



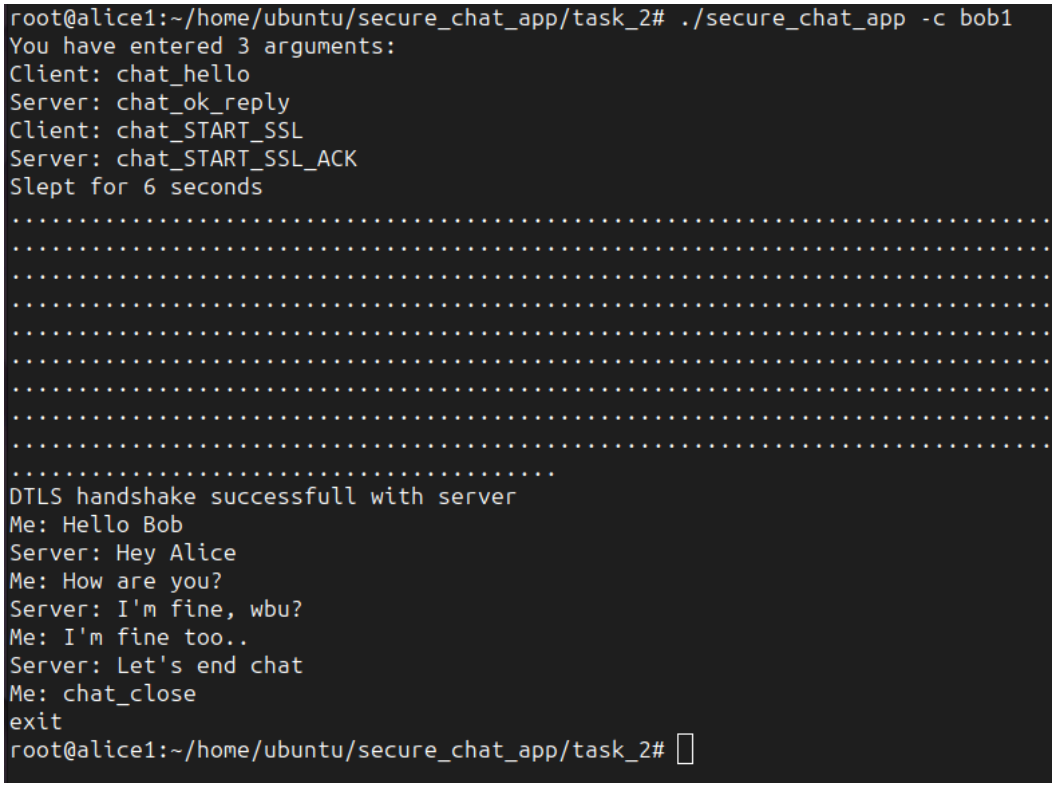
3. Flowchart of server function:



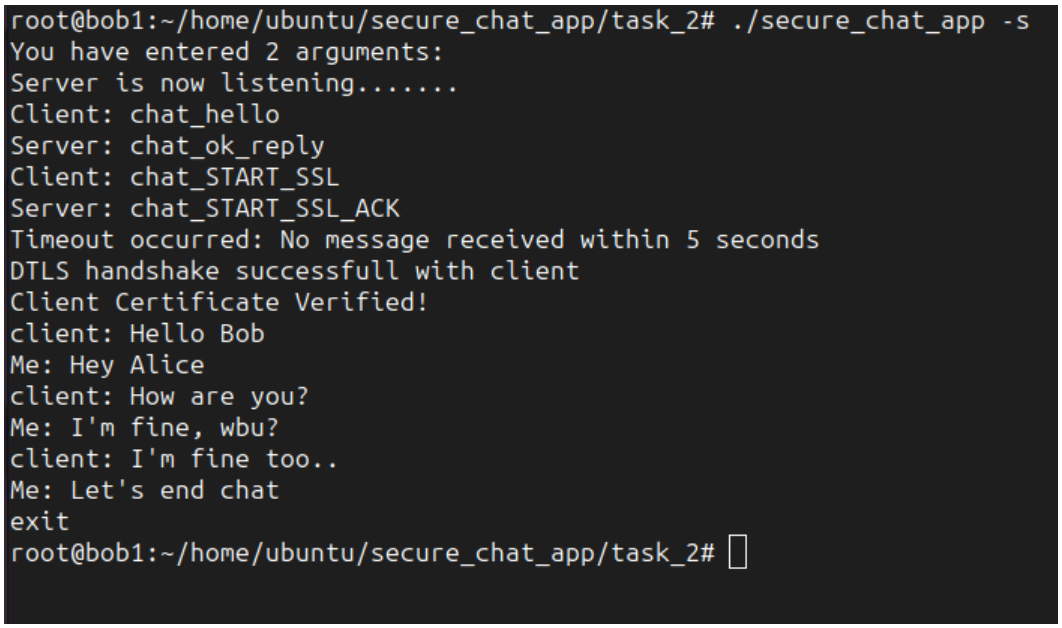
Working of the program demonstrated using screenshots:

1. **Without packet losses:**

Client side screenshot:

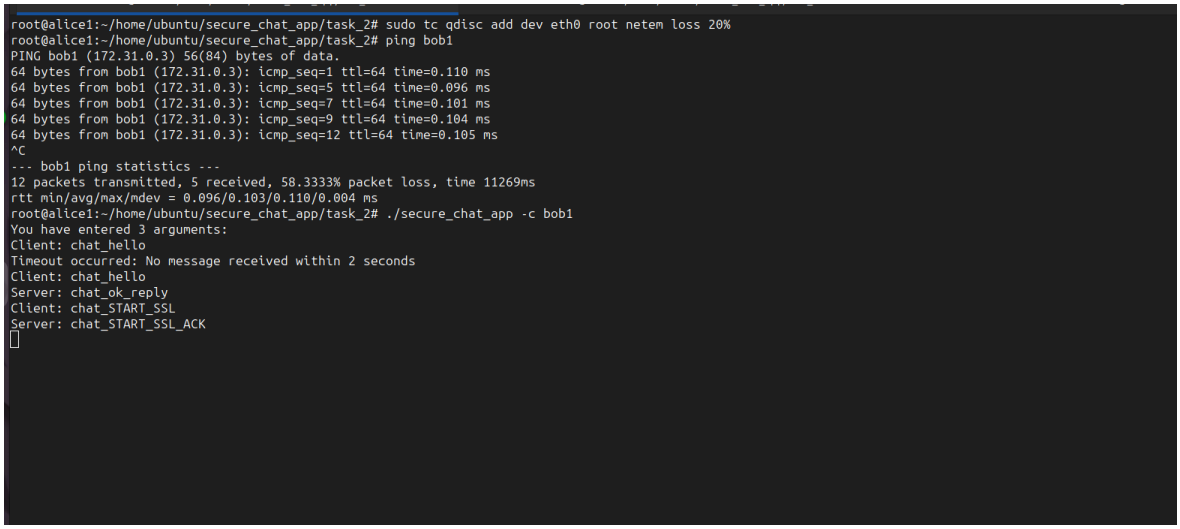


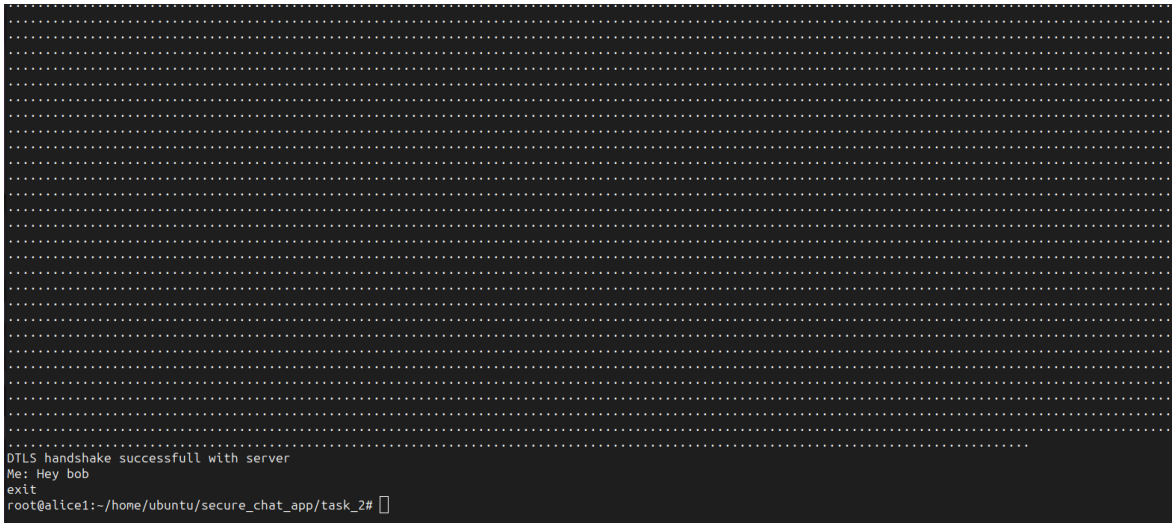
Server side screenshot:



2. With packet loss:

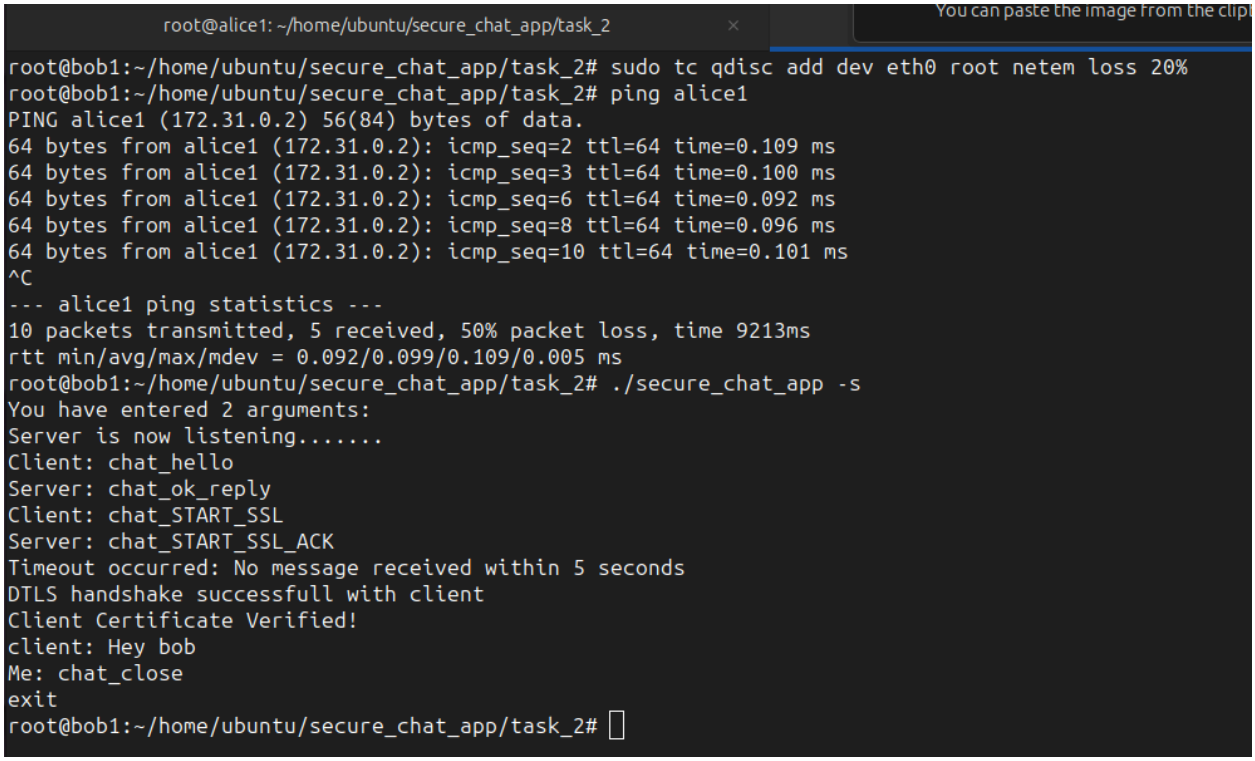
Client side screenshot:





a. We introduce 20% packet loss using command: *$ sudo tc qdisc add dev eth0 root netem loss 20%*
b. It was verified that packets are indeed being lost using ping, which showed 58.33 % packet loss as 20 % packet loss was introduced on both client and server side.
c. We can observe that the first ‘chat_hello’ was lost or the acknowledgement to it never reached the client, hence the timeout. As soon as the timeout occurred, ‘chat_hello’ was re-sent.
d. In the second screenshot above, each ‘.’ (period) represents a fragment of ‘client hello’ being sent, the number of ‘.’ here are more in comparison to screenshot in without loss case because whenever a fragment is lost in transmission, it is being re-sent.

Server side screenshot:



A packet loss of 20 % has also been introduced on the server side too using the same command used on the client side and verified by pinging Alice.

Explaining important code snippets:

1) Reliability for ‘chat_hello on client part

```
while(true){
    // Send initial message to server
    sendto(sockfd, "chat_hello", strlen("chat_hello"), 0, (struct sockaddr*)&server_addr, sizeof(server_addr));
cout << "Client: chat_hello" << endl;

    // Set receive timeout for socket
    timeout_handshake.tv_sec = 2; // 2 seconds timeout
    timeout_handshake.tv_usec = 0;
    if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout_handshake, sizeof(timeout_handshake)) < 0) {
        error_handler("Error setting socket receive timeout");
    }

    // Receive response from server
    int recvd_msg_size = recvfrom(sockfd, buffer, MAX_SIZE - 1, 0, (struct sockaddr*)&server_addr, &addr_len);
    if (recvd_msg_size < 0) {
        if(errno == EWOULDBLOCK) {
            cout << "Timeout occurred: No message received within 2 seconds" << endl;
            continue;
        }else{
            cout << "Failed response from server" << endl;
            continue;
        }
    }
    buffer[recvd_msg_size] = '\0';
    cout << "Server: " << buffer << endl;

    // If server acknowledges, initiate DTLS handshake
    if(strcmp(buffer, "chat_ok_reply") == 0) {
        break;
    }
}
```

a. We set a timeout of two seconds for receiving acknowledgement ‘chat_ok_reply’ from the server.
b. If the client gets ‘chat_ok_reply’ within 2 seconds he breaks out of the loop and sends ‘chat_START-SSL’ next.
c. If there is no acknowledgement within 2 seconds, the client resends ‘chat_hello’.
d. Similar thing is followed for ‘chat_START_SSL’ too.

2) Reliability for initial control message on server side

```
// Wait for client's initial message
while(true) {
    int recvd_msg_size;
    if(all_cm){
        timeout_handshake.tv_sec = 5; // 5 seconds timeout
        timeout_handshake.tv_usec = 0;
        if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout_handshake, sizeof(timeout_handshake)) < 0) {
            error_handler("Error setting socket receive timeout");
        }

        // Receive response for DTLS initiation
        recvd_msg_size = recvfrom(sockfd, buffer, MAX_SIZE - 1, 0, (struct sockaddr*)&client_addr, &addr_len);
        if (recvd_msg_size < 0) {
            if(errno == EWOULDBLOCK) {
                cout << "Timeout occurred: No message received within 5 seconds" << endl;
                break;
            }
        }
    }else{
        recvd_msg_size = recvfrom(sockfd, buffer, MAX_SIZE - 1, 0, (struct sockaddr*)&client_addr, &addr_len);
    }
    if (recvd_msg_size < 0) {
        error_handler("Failed handshake from client");
    }
    buffer[recvd_msg_size] = '\0';
    cout << "Client: " << buffer << endl;
    if(strcmp(buffer, "chat_hello") == 0) {
        sendto(sockfd, "chat_ok_reply", strlen("chat_ok_reply"), 0, (struct sockaddr*)&client_addr, addr_len);
        cout << "Server: chat_ok_reply" << endl;
    } else if(strcmp(buffer, "chat_START_SSL") == 0) {
        sendto(sockfd, "chat_START_SSL_ACK", strlen("chat_START_SSL_ACK"), 0, (struct sockaddr*)&client_addr, addr_len);
        cout << "Server: chat_START_SSL_ACK" << endl;
        ssl_check = true;
        all_cm = true;
        continue;
    }else{
        error_handler("Unrecognized message");
    }
}
```

a. Whenever the server receives a control message, he sends an appropriate acknowledgment message back.
b. After sending ‘chat_START_SSL_ACK’ to the client, the server goes into listen mode for 5 seconds in case the ‘chat_START_SSL_ACK’ was lost and the client has to resend the ‘chat_START_SSL’ message.

3) Reliability for DTLS handshake on the client side.

```
if (setSocketNonBlocking(sockfd) == -1) {
    cerr << "Error setting socket to non-blocking mode" << endl;
    SSL_CTX_free(ctx);
    SSL_shutdown(ssl);
    SSL_free(ssl);
    error_handler("");
}

int res;
do{
    res = SSL_connect(ssl);
cout << ".";
}while(res != 1);

cout << "" << endl;

cout << "DTLS handshake successfull with server" << endl;
```

```
int setSocketNonBlocking(int socket_fd) {
    int flags = fcntl(socket_fd, F_GETFL, 0);
    if (flags == -1) {
        perror("fcntl");
        return -1;
    }

    if (fcntl(socket_fd, F_SETFL, flags | O_NONBLOCK) == -1) {
        perror("fcntl");
        return -1;
    }

    return 0;
}
```

a. First we set the socket into non-blocking mode using the file descriptor
b. In non-blocking mode the program receives an immediate response from the operating system indicating whether the operation is completed successfully or not and hence we could retry if it failed.
c. We put the ‘ssl_connect()’ in a loop to keep on trying handshake with the server till it is successful.
d. In the setSocketNonBlocking function we first retrieve the current flag using ‘fcntl’ and the using the ‘|’ Bitwise OR operator combines the retrieved flag and ‘0_NONBLOCK’ using ‘fcntl’ again.

4) Verify certificate received.

```
// Verify client's certificate
if(SSL_get_peer_certificate(ssl)) {
    if( SSL_get_verify_result(ssl) == X509_V_OK) {
        cout << "Client Certificate Verified! \n";
    }
} else {
    error_handler("Failed to get peer certificate");
}
```

```
void configure_ctx(SSL_CTX *ctx) {
    SSL_CTX_set_security_level(ctx, 1);
    SSL_CTX_set_mode(ctx, SSL_MODE_AUTO_RETRY);
    SSL_CTX_set_session_cache_mode(ctx, SSL_SESS_CACHE_OFF);
    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
}
```

   a. In the second screenshot:
      1. SSL_VERIFY_PEER: The flag informs server to verify the client's certificate
      2. SSL_VERIFY_FAIL_IF_NO_PEER_CERT: The flag informs the server to fail verification if the client hasn't provided a certificate.
   b. In the first screenshot above:
      1. SSL_get_peer_certificate(ssl): This checks if the client has provided a certificate or not.
      2. SSL_get_verify_result(ssl) == X509_V_OK: This verifies the validity of the certificate and returns 'X509_V_OK' if valid.

5) Setting cipher suites which provide Perfect Forward Secrecy (PFS).

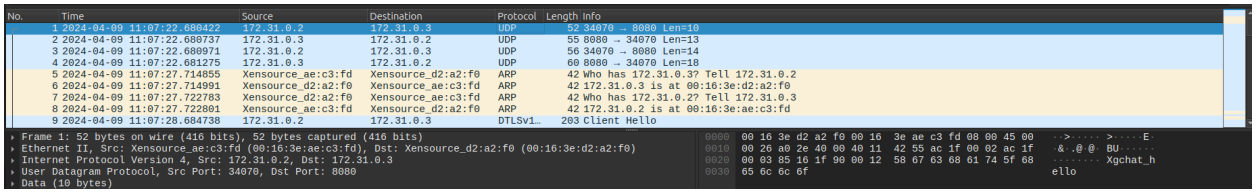
```
const char *pfs_ciphers = "ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384";
if (SSL_CTX_set_cipher_list(ctx, pfs_ciphers) != 1) {
    ERR_print_errors_fp(stderr);
    error_handler("Failed to set cipher suites");
}
```

   a. const char *pfs_ciphers = "ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384": This line saves two cipher suites separated by a colon, both of which provides PFS into a string pfs_ciphers.
   b. SSL_CTX_set_cipher_list(ctx, pfs_ciphers): This line sets the cipher suites to the SSL context called ctx

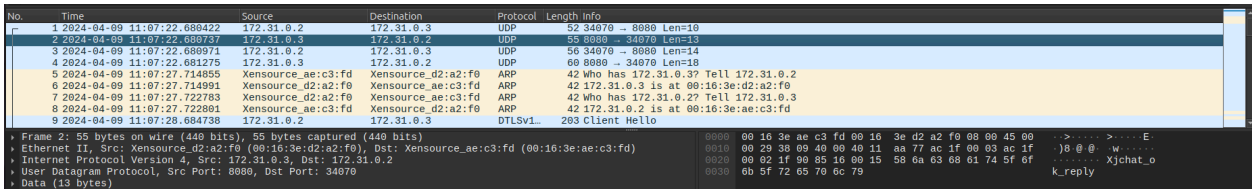
Screenshots of pcap files:

pcap file when no packet loss was induced:

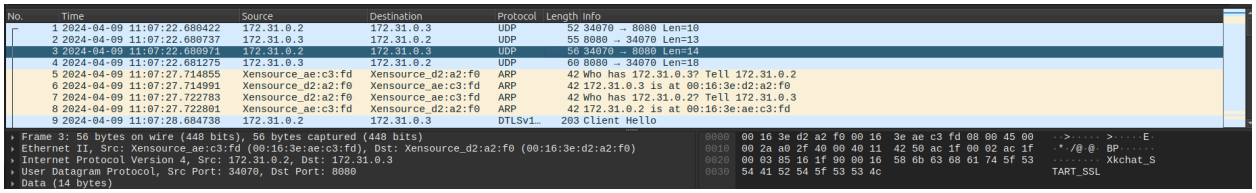
1. ‘chat_hello’ from Alice to Bob:



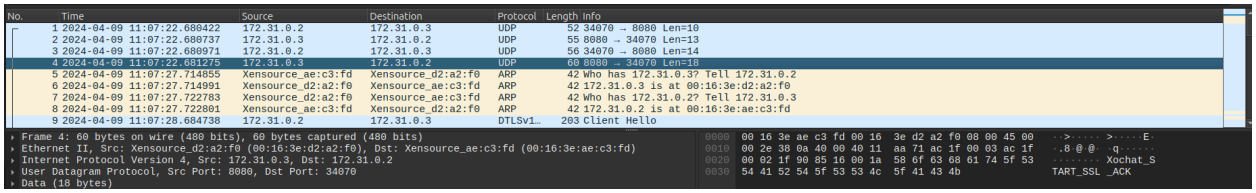
2. ‘chat_ok_reply’ from Bob to Alice:



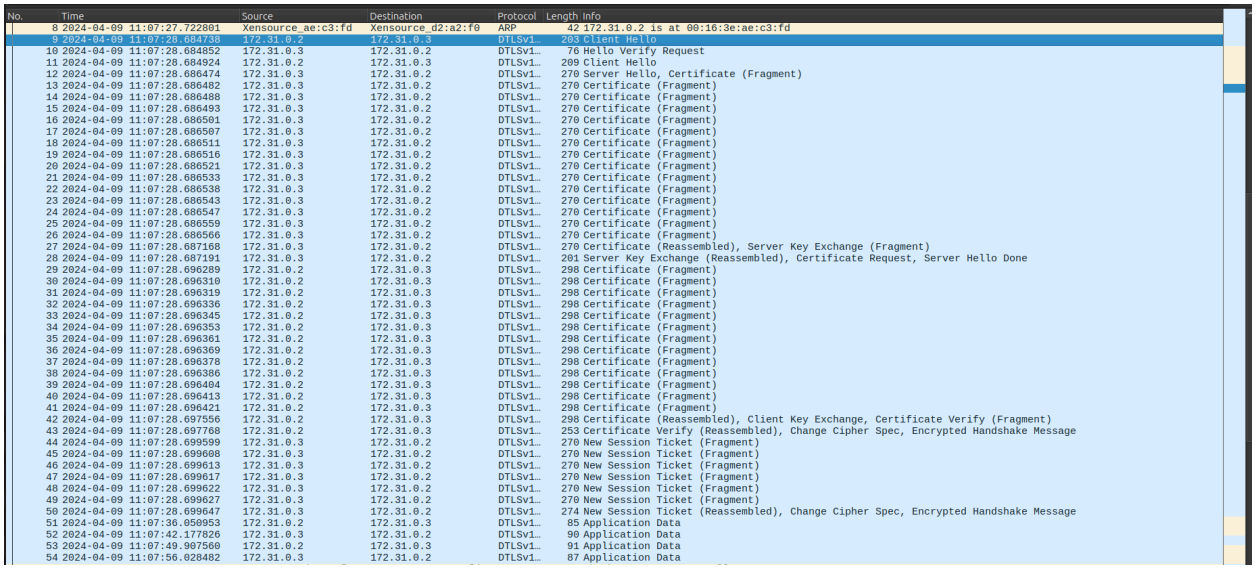
3. ‘chat_START_SSL’ from Alice to Bob.



4. ‘chat_START_SSL_ACK’ from Bob to Alice.

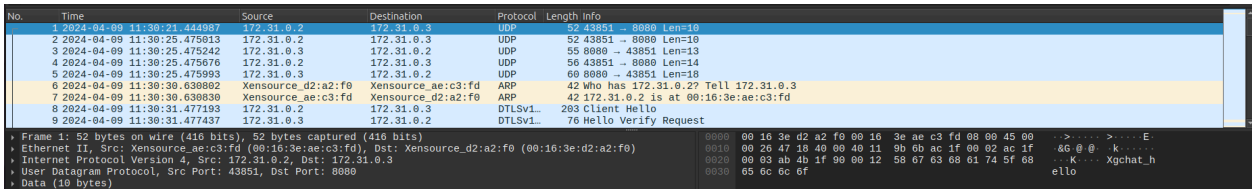


5. client_hello, server_hello, certificate exchange and session ticket generation.

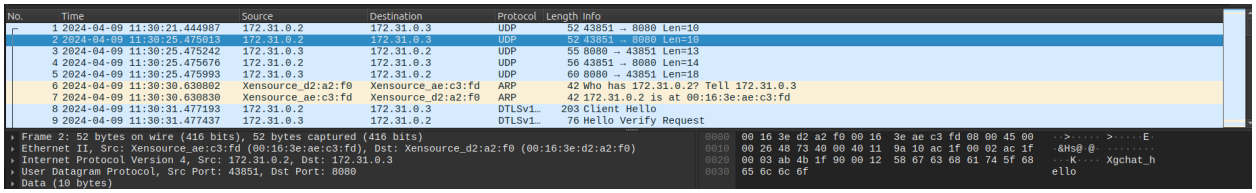


pcap file with 20% packet loss induced on both Alice and Bob side:

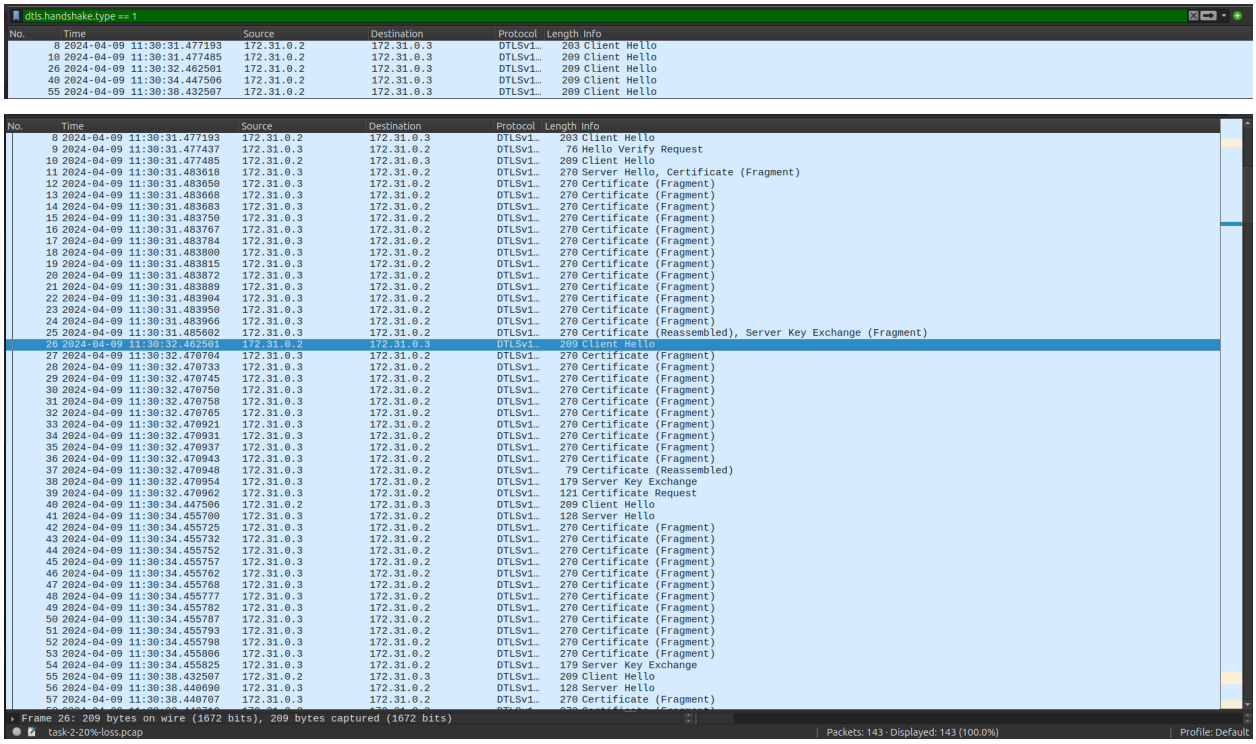
1. First ‘chat_hello’ message sent from Alice to Bob.



2. Second “chat_hello’ message sent from Alice to Bob as acknowledgement for the first was lost.



3. Screenshot showing ‘clent_hello’ being retransmitted multiple times as fragments of it got lost because of packet loss.



Comparison between DTLS1.2 handshake and TLS1.2 handshake.

Similarities between DTLS1.2 handshake and TLS1.2 handshake:

1. Same flow of handshake messages: Both DTLS1.2 and TLS1.2 have the same flow of initial message exchanges like client hello, server hello, certificate exchange (optional) for authentication, key exchange and finished messages.
2. Cipher suites: Both support a wide range of cipher suites which include schemes for key exchange, authentication, encryption, and message authentication and these cipher suites are negotiated during the handshake process.
3. Certificate based authentication: Both support X.509 digital certificates for certificate based authentication.
4. Perfect forward secrecy: Both support cipher suites with PFS so that even if the long term private key is compromised, the past sessions are safe.
5. Key exchange and shared secret derivation: Both support different key exchange algorithms like RSA, Diffie-Hellman, and Elliptic Curve Diffie-Hellman (ECDH) etc, which are used to derive shared secrets between client and server. These shared secrets are later used to generate keys for encryption to secure communication channels.
6. Security goals: The primary security goals of both are similar and include: confidentiality, integrity, authentication, perfect forward secrecy, key exchange security.

Differences between DTLS1.2 handshake and TLS1.2 handshake:

DTLS1.2 handshake	**TLS1.2 handshake**
DTLS is over UDP	TLS is over TCP
Replay detection is optional in DTLS	Replay detection is a required feature of TLS
DTLS doesn't reorder or re-transmit packets	TLS reorders and re-transmits packets
DTLS has sequence numbers for handshake messages.	TLS handshake messages doesn't have sequence numbers

Task 3: START_SSL downgrade attack for eavesdropping

Design overview of secure_chat_interceptor:

1. The main function takes command line argument:
   a) `secure_chat_interceptor -d <clienthostname> <serverhostname>`: Program works in downgrading interceptor mode.
2. List of control messages:

	Client's message	Reply from Interceptor
a.	chat_START_SSL	chat_START_SSL_NOT_SUPPORTED

3. Once the client gets 'chat_START_NOT_SUPPORTED' response, she assumes server lacks DTLS support.
4. Reliability has been taken care of by client and server end.
5. A separate function has been implemented to handle errors and thus, the program handles errors gracefully.
6. The program cleans up everything and frees up memory when it is about to exit.

a)

```
204     while(true){
205         sendto(sockfd, "chat_START_SSL", strlen("chat_START_SSL"), 0, (struct sockaddr*)&server_addr, addr_len);
206
207         // Set receive timeout for socket
208         timeout_handshake.tv_sec = 2; // 2 seconds timeout
209         timeout_handshake.tv_usec = 0;
210         if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout_handshake, sizeof(timeout_handshake)) < 0) {
211             error_handler("Error setting socket receive timeout");
212         }
213
214         // Receive response for DTLS initiation
215         int recvd_msg_size = recvfrom(sockfd, buffer, MAX_SIZE - 1, 0, (struct sockaddr*)&server_addr, &addr_len);
216         if (recvd_msg_size < 0) {
217             if(errno == EWOULDBLOCK) {
218                 cout << "Timeout occurred: No message received within 2 seconds" << endl;
219                 continue;
220             }else{
221                 cout << "Failed response from server" << endl;
222                 continue;
223             }
224         }
225         buffer[recvd_msg_size] = '\0';
226         cout << "Server: " << buffer << endl;
227
228         if(strcmp(buffer, "chat_START_SSL_ACK") == 0){
229             ssl_check = true;
230             break;
231         }else if(strcmp(buffer, "chat_START_SSL_NOT_SUPPORTED") == 0){
232             break;
233         }
234     }
```

Fig: Code snippet for handling chat_START_SSL_NOT_SUPPORTED message on alice's side

In the above figure we can see that at the time of DTLS handshake initiation we are sending chat_START_SSL message on the client's side and on receiving the chat_START_SSL_NOT_SUPPORTED message from the interceptor we assume that server does not support SSL. As a result we keep the ssl_check flag

at its initialized value of false.

```
325     else if(!ssl_check){
326         // Chat loop
327         while(true) {
328             cout << "Me: ";
329             string inp;
330             getline(cin, inp);
331             const char* msg;
332             msg = &inp[0];
333             sendto(sockfd, msg, strlen(msg), 0, (struct sockaddr*)&server_addr, addr_len);
334             if(strcmp(msg, "chat_close") == 0) {
335                 cout << "exit" << endl;
336                 break;
337             }
338 
339             int recvd_msg_size = recvfrom(sockfd, buffer, MAX_SIZE - 1, 0, (struct sockaddr*)&server_addr, &addr_len);
340             if (recvd_msg_size < 0) {
341                 error_handler("Failed response from server");
342             }
343             buffer[recvd_msg_size] = '\0';
344             cout << "Server: " << buffer << endl;
345             if(strcmp(buffer, "chat_close") == 0) {
346                 cout << "exit" << endl;
347                 break;
348             }
349 
350         }
```

Fig: Code snippet for continuing to chat with server on receiving chat_START_SSL_NOT_SUPPORTED message over UDP connection.

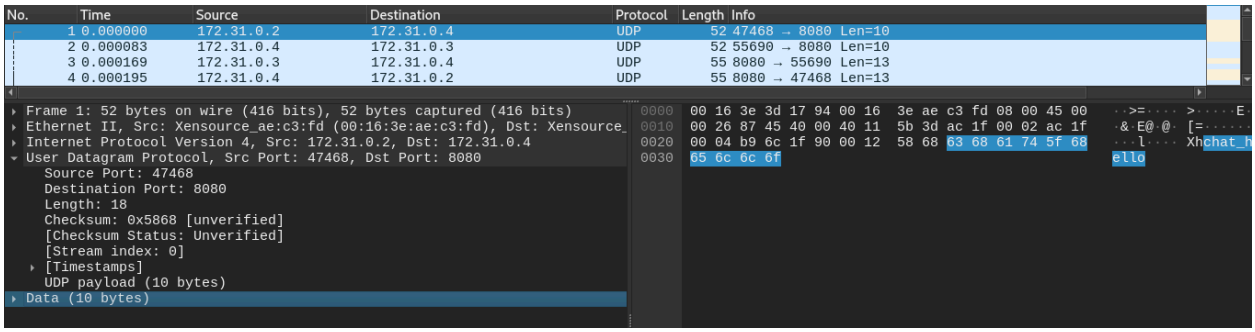
In the above figure we see that the Client continues communicating with the server though it doesn't establish a DTLS connection and continues over plain UDP.

b) **To be done: pcap**

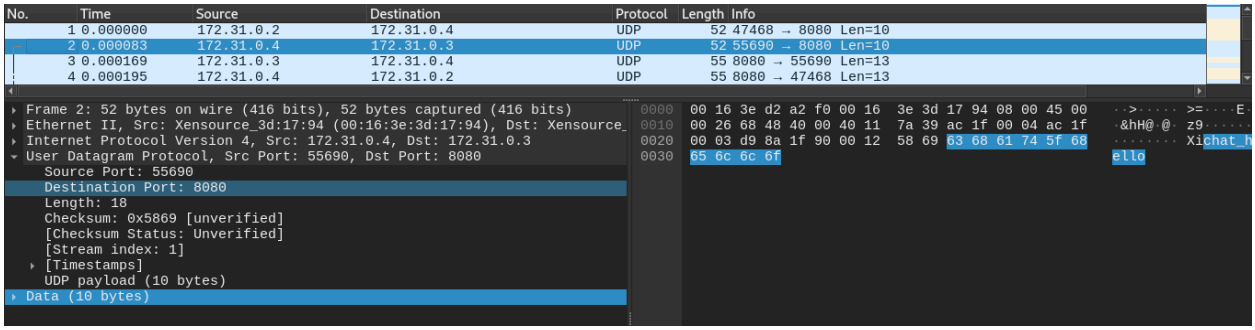
Screenshots of pcap files;

No packet loss was induced:

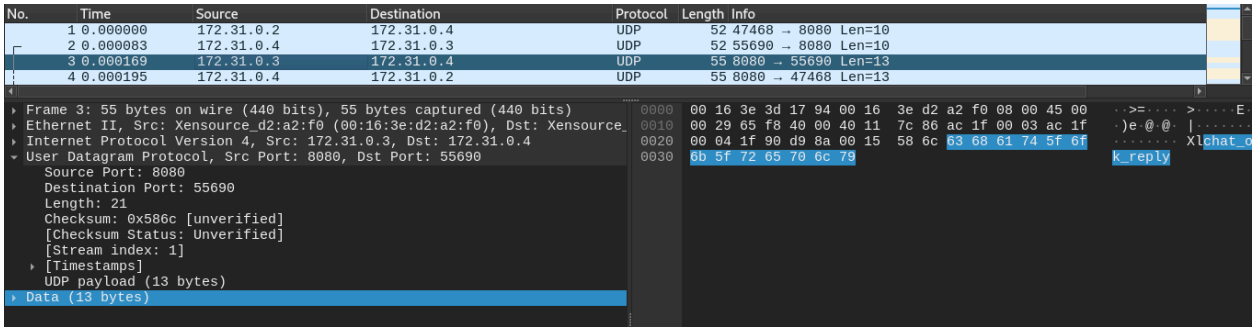
1. 'chat_hello' from Alice to Trudy:



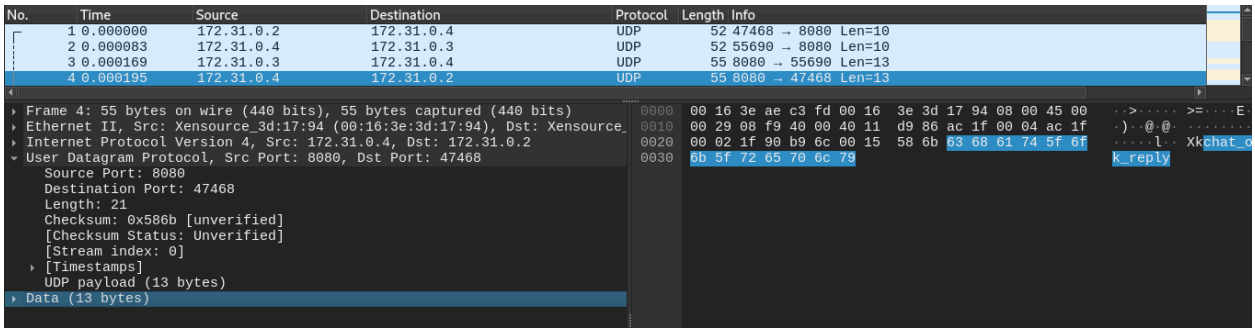
2. 'Chat_hello' forwarded from Trudy to Bob:



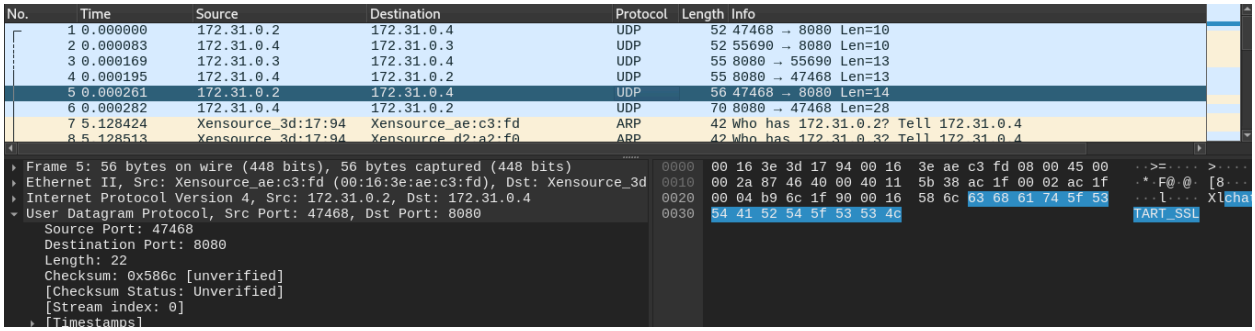
3. ‘Chat_ok_reply’ from Bob to Trudy:



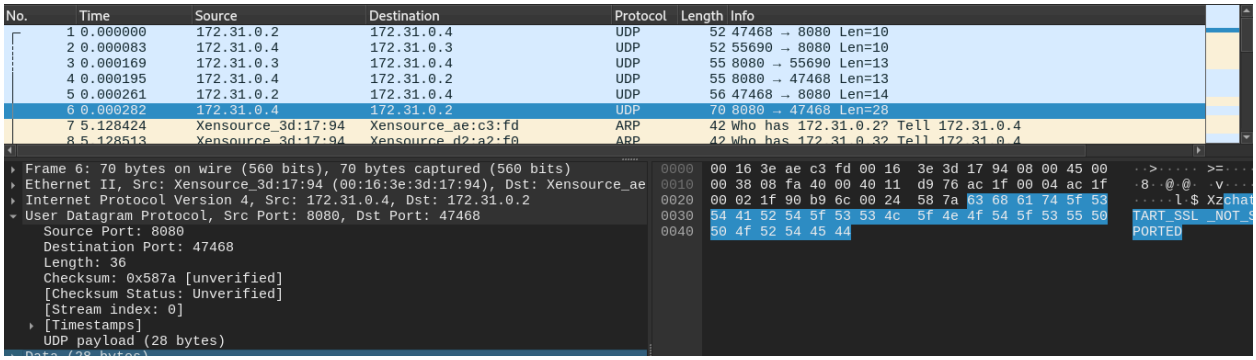
4. Chat_ok_reply’ forwarded from Trudy to Alice:



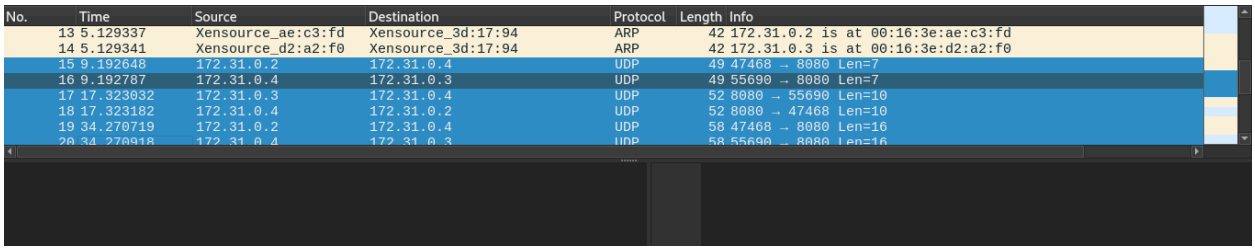
5. ‘chat_START_SSL’ from Alice to Trudy.



6. ‘chat_START_SSL_NOT_SUPPORTED’ from Bob to Alice.



7. Trudy eavesdropping all the messages over UDP.



c)

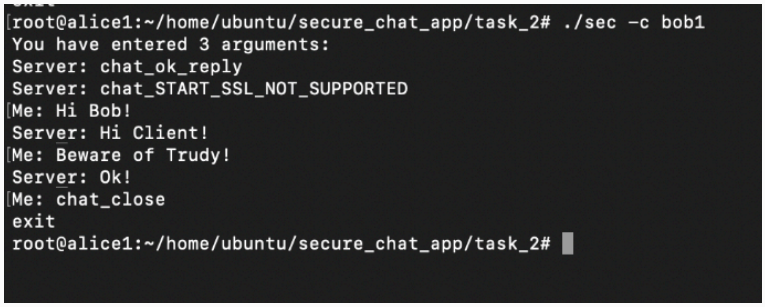


Fig: client end of the chat application running on alice1 container

In the above figure we can see that we run the chat application compiled as file **sec** using the command:

./sec -c bob1

We can see in the sequence of messages received in the terminal that Alice receives chat_START_SSL_NOT_SUPPORTED message. This prevents the DTLS handshake as Alice assumes that server does not support DTLS. So it just continues with the UDP connection.

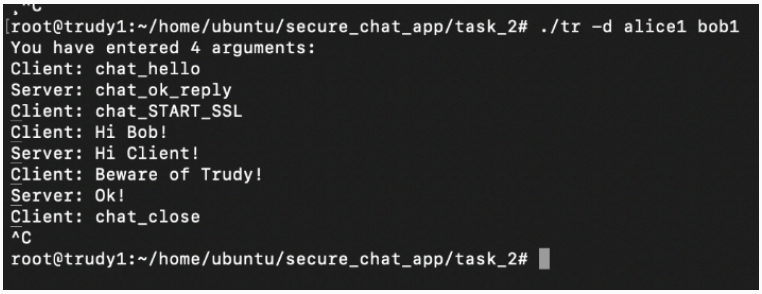


Fig: interceptor app running on trudy1 container

In the above figure we can see that we run the interceptor application compiled as file **tr** using the command:

./tr -d alice1 bob1

We can see in the sequence of messages received in the terminal that when Alice sends chat_START_SSL message no message is intercepted from server because Trudy has already sent back the chat_START_SSL_NOT_SUPPORTED.

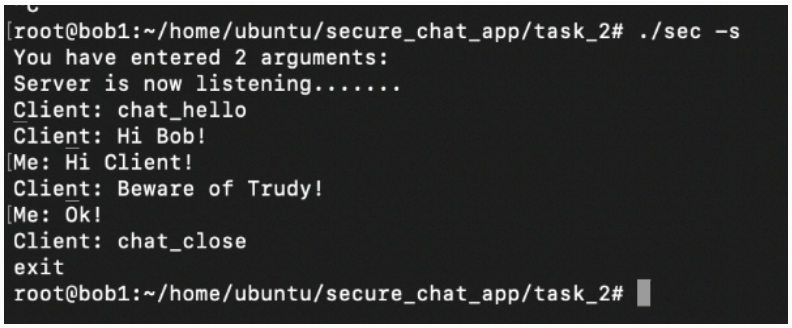


Fig: server end of the chat application running on bob1 container

In the above figure we can see that we run the chat application compiled as file **sec** using the command:

./sec -s

We can see in the sequence of messages received in the terminal that Bob does not receive chat_START_SSL message as it has already been intercepted by the server. This prevents the DTLS handshake as Alice assumes that server does not support DTLS. So it just continues with the UDP connection.

```
189     while(true) {
190         int recvd_msg_size = recvfrom(sockfd_client, buffer, MAX_SIZE - 1, 0, (struct sockaddr*)&client_addr, &addr_len_c);
191         if (recvd_msg_size < 0) {
192             error_handler("Failed handshake from client");
193         }
194         buffer[recvd_msg_size] = '\0';
195         cout << "Client: " << buffer << endl;
196         if(strcmp(buffer, "chat_START_SSL") == 0) {
197             sendto(sockfd_client, "chat_START_SSL_NOT_SUPPORTED", strlen("chat_START_SSL_NOT_SUPPORTED"), 0, (struct sockaddr*)&client_addr, sizeof(client_addr));
198             continue;
199         }else{
200             sendto(sockfd_server, buffer, strlen(buffer), 0, (struct sockaddr*)&server_addr, sizeof(server_addr));
201         }
202 
203         recvd_msg_size = recvfrom(sockfd_server, buffer, MAX_SIZE - 1, 0, (struct sockaddr*)&server_addr, &addr_len_s);
204         if (recvd_msg_size < 0) {
205             error_handler("Failed response from server");
206         }
207         buffer[recvd_msg_size] = '\0';
208         cout << "Server: " << buffer << endl;
209 
210         sendto(sockfd_client, buffer, strlen(buffer), 0, (struct sockaddr*)&client_addr, sizeof(client_addr));
211 
212     }
```

Fig: code snippet from interceptor which enables it to downgrade the connection

In the above figure we can see that trudy will receive messages from Client(Alice) and pass on these messages to Bob unless a chat_START_SSL message is received. On receiving the chat_START_SSL message interceptor will directly send chat_START_SSL_NOT_SUPPORTED message to the server, not letting it move on to the DTLS handshake phase and successfully downgrading the connection to a simple UDP one.

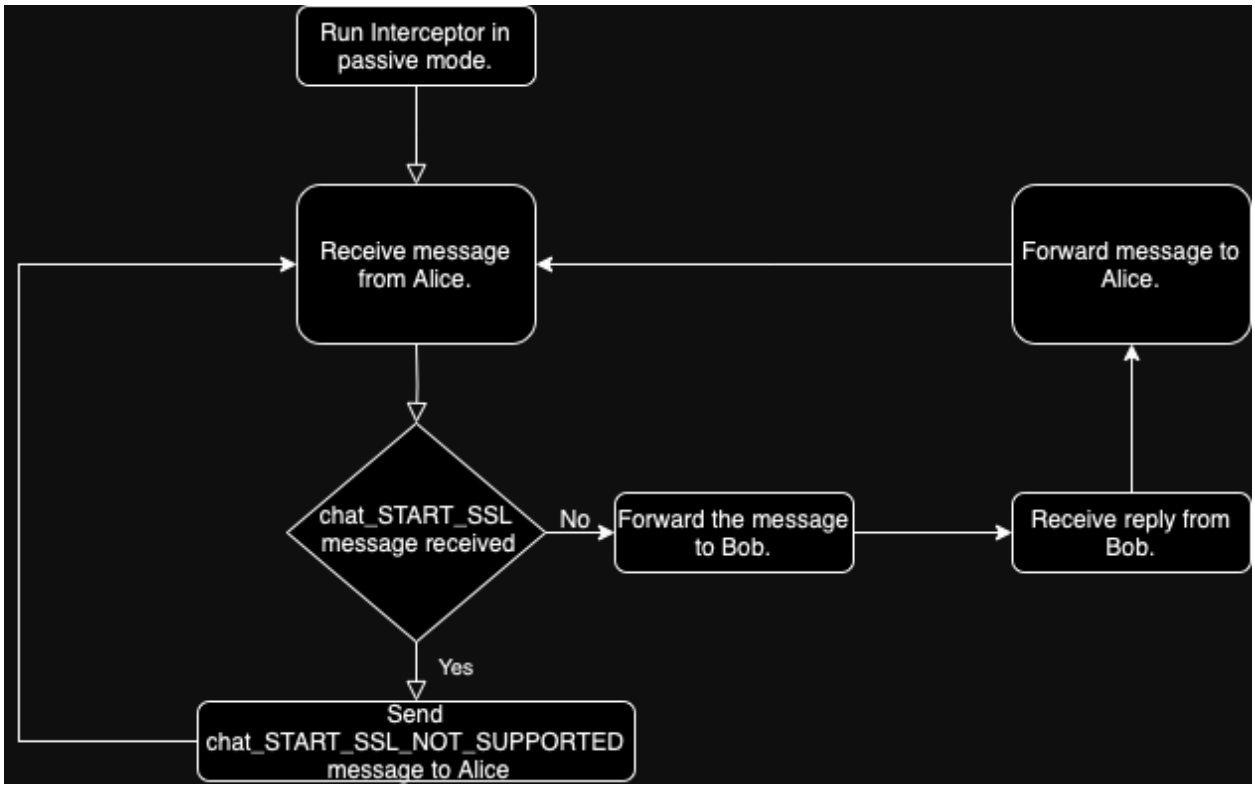


Fig: Flow Chart showing how our interceptor works

Task 4: Active MITM attack for tampering chat messages and dropping DTLS handshake messages

Design overview of secure_chat_active_interceptor:

1) The main function takes command line argument:
    a. `secure_chat_active_interceptor -m <clienthostname> <serverhostname>:` Program works in downgrading interceptor mode.
2) List of control messages:

	Client's message	Expected reply from server
a.	chat_hello	chat_ok_reply
b.	chat_START_SSL	chat_START_SSL_ACK

3) Once the client gets 'chat_START_SSL_ACK' response, she initiates DTLS handshake with the interceptor where both exchange each other's certificate and verify it.
4) Before initiating the DTLS handshake, client sets up ciphersuites which provide perfect forward secrecy: ECDHE-RSA-AES256-GCM-SHA384 and ECDHE-ECDSA-AES256-GCM-SHA384
5) Trudy initiates DTLS handshake with the server where both exchange each other's certificate and verify it.
6) Before initiating the DTLS handshake, trudy sets up ciphersuites which provide perfect forward secrecy: ECDHE-RSA-AES256-GCM-SHA384 and ECDHE-ECDSA-AES256-GCM-SHA384.
7) Reliability has been implemented for all the control messages including DTLS handshake using timeouts and retransmission. It will handle packet losses in UDP.
8) We have the SSL context in 'SSL_MODE_AUTO_RETRY' mode which specifies SSL read and SSL write to auto-retry when they are interrupted
9) A separate function has been implemented to handle errors and thus, the program handles errors gracefully.
10) The program cleans up everything and frees up memory when it is about to exit.

a)

```
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl genrsa -out alice-fake-private-key.pem 1024
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl rsa -in alice-fake-private-key.pem -pubout -out alice-fake-public-key.pem
writing RSA key
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ cat alice-fake-private-key.pem
-----BEGIN PRIVATE KEY-----
MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBALEnCxs+jKoE/Am5
tua590KqazFG/A5ek2BIT3oCkV13iEobzBSa+puQNr8fpju9Nr6CkVfL1LMS9h9N
Y3E2ZJ/WhKHaoAbEwp7t4ijAha8+pvqnDdQ1p1F4vCRbd3ON1AaJpDIyrL+fakLW
YHIpUmhxzOHaizqCrRun7tN7xspJAgMBAAECgYEAk1ybDIq3kCadUam+z6caW8ZP
bKpTudBKX+lG04OXUFzCboM4/Fr+5LDOGaZGLaN0Cn2qUZIo2N2JBvjlI6PPhxQA
xM7N+jNUpobUOlenQE0b27OFxZvoVLS366Deb0zfT3DYUh88kBd2XEXxxrV/pNfA
hxzRNiNzcrutSe9+2tECQQDbkVl1PkuosXNaxaCiKbwaDaC1vQlWvbIlxx4A6FBH
RzIGRGYtn9U0iHAGgT3LTCDEAWIA51xD7D/oj0HchOPrAkEAzowB51Tg1ullmWjm
WxjzB1CZYJnMbAFEk36Eqz5GLbTDKBGf/ktpVlvq1aqWxv6Mir0PbLtj1IGt5rSw
H26hmwJBAKMFiSkexqgKkL5dq4WbfRjO83rcTxnDIIkiDOnZkvlJKCA1GrRXTsKB
ZsO9o7bU4CMHb8+B+CRKPa0a6Gilll8CQHEWAuzPchL40uG9p3H1JQq6PF/BlUUF
JLDEsoq3m+DmeXt0uu6WB2R7CwZVUhw9vmEvxkQ4DR6KqdYI0+1Bu0UCQCOasqVE
X4OFsYvchdfLkW0XMOWoccuQiDNhQ0Z1LOy5N6Ue1Tex57iZBpaicGL2Tvppdovm
UOu9cdQ8ZDtce98=
-----END PRIVATE KEY-----
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ cat alice-fake-public-key.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCxJwsbPoyqBPwJubbmufdCqmsx
RvwOXpNgSE96ApFdd4hKG8wUmvqbkDa/H6Y7vTa+gpFXy9SzEvYfTWNxNmSf1oSh
2qAGxMKe7eIowIWvPqb6pw3UNadReLwkW3dzjdQGiaQyMqy/n2pC1mByKVJocczh
2os6gq0bp+7Te8bKSQIDAQAB
-----END PUBLIC KEY-----
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$
```

Fig: Generating fake private key for Alice

```
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl req -new -key alice-fake-private-key.pem -out alice-fake.csr -config alice.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
IN []:IN
Telangana []:Telangana
Hyderabad []:Hyderabad
IIT Hyderabad []:IIT Hyderabad
CSE []:CSE
Alice1.com []:Alice1.com
cs23mtech11004@iith.ac.in []:cs23mtech11004@iith.ac.in
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl req -in alice-fake.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Alice1.com, emailAddress = cs23mtech11004@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:b1:27:0b:1b:3e:8c:aa:04:fc:09:b9:b6:e6:b9:
                    f7:42:aa:6b:31:46:fc:0e:5e:93:60:48:4f:7a:02:
                    91:5d:77:88:4a:1b:cc:14:9a:fa:9b:90:36:bf:1f:
                    a6:3b:bd:36:be:82:91:57:cb:d4:b3:12:f6:1f:4d:
                    63:71:36:64:9f:d6:84:a1:da:a0:06:c4:c2:9e:ed:
                    e2:28:c0:85:af:3e:a6:fa:a7:0d:d4:35:a7:51:78:
                    bc:24:5b:77:73:8d:d4:06:89:a4:32:32:ac:bf:9f:
                    6a:42:d6:60:72:29:52:68:71:cc:e1:da:8b:3a:82:
                    ad:1b:a7:ee:d3:7b:c6:ca:49
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        6f:b6:57:98:01:d1:d7:34:f0:7e:af:12:05:92:b5:be:39:c0:
        0a:b4:7e:4e:e7:ef:d8:3b:96:e5:e2:5f:2a:24:46:82:7f:73:
        f9:40:87:f8:28:93:4f:86:8c:81:10:fc:4a:35:b7:2b:ec:0d:
        61:82:24:ad:70:1e:80:b2:4e:14:be:0e:5b:ad:04:c1:cb:8b:
        7a:b9:04:1f:df:6d:aa:26:a6:d0:05:3a:f0:f0:1d:6a:e5:4f:
        65:35:4e:f8:b0:37:4b:1f:2d:9e:e6:f1:d2:4c:0d:4f:cd:03:
```

Fig: Generating CSR for fake alice certificate

```
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl x509 -req -in alice-fake.csr -CA inter.crt -CAkey private-key-inter.pem -CAcreates
erial -out alice-fake.crt -days 90 -extfile cert.cnf -extensions v3_ca
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Alice1.com, emailAddress = cs23mtech11004@iith.ac.in
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl x509 -noout -text -in alice-fake.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:61:c4:33:f0:5f:d7:07:b4:d6:aa:63:8d:fe:17:4f:d0:28:38:9a
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = iTS CA 1R3, emailAddress = iTSinter@iTS.ac.in
        Validity
            Not Before: Apr 10 12:13:31 2024 GMT
            Not After : Jul  9 12:13:31 2024 GMT
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Alice1.com, emailAddress = cs23mtech11004@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:b1:27:0b:1b:3e:8c:aa:04:fc:09:b9:b6:e6:b9:
                    f7:42:aa:6b:31:46:fc:0e:5e:93:60:48:4f:7a:02:
                    91:5d:77:88:4a:1b:cc:14:9a:fa:9b:90:36:bf:1f:
                    a6:3b:bd:36:be:82:91:57:cb:d4:b3:12:f6:1f:4d:
                    63:71:36:64:9f:d6:84:a1:da:a0:06:c4:c2:9e:ed:
                    e2:28:c0:85:af:3e:a6:fa:a7:0d:d4:35:a7:51:78:
                    bc:24:5b:77:73:8d:d4:06:89:a4:32:32:ac:bf:9f:
                    6a:42:d6:60:72:29:52:68:71:cc:e1:da:8b:3a:82:
                    ad:1b:a7:ee:d3:7b:c6:ca:49
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Key Agreement
            X509v3 Subject Key Identifier:
                B6:C7:5D:E6:AF:FA:0D:C9:8B:BD:1F:B4:03:1B:05:CA:33:EA:14:62
            X509v3 Authority Key Identifier:
                2E:33:0D:B3:13:47:4B:45:F7:59:F7:81:56:7B:42:1F:7F:B9:F7:7F
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        50:05:23:e6:90:e0:ee:25:64:2b:4a:5a:8e:ea:ef:fa:4d:65:
        81:bd:d3:46:9b:a6:a5:72:e9:08:17:76:09:1e:da:71:bd:cf:
        12:eb:40:ec:52:4a:4a:16:64:3f:46:68:37:93:8f:2b:b8:85:
        0c:06:4c:f1:22:c1:51:1f:66:87:e8:f0:70:f6:f0:ad:27:6e:
        0f:b2:e7:10:d8:05:20:47:70:9e:63:91:d9:1d:ce:1b:bb:5c:
```

Fig: Using Intermediate certificate to generate fake Alice certificate

```
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl ecparam -genkey -name prime256v1 -out bob-fake-private-key.pem
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl ec -in bob-fake-private-key.pem -pubout -out bob-fake-public-key.pem
read EC key
writing EC key
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ cat bob-fake-private-key.pem
-----BEGIN EC PARAMETERS-----
BggqhkjOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIGp+dgxU7D+eso4OBPpE/4F43PpNcFRn4j0VeY4AMv/aoAoGCCqGSM49
AwEHoUQDQgAEd46t3j3jyzfw6/xNxQj9Z+i/9j9q4dmErgXivP5ZOL7de1WR6HFY
TPHMYOYeFmWizC6Jwc2Tam/RhnVpqRNYkw==
-----END EC PRIVATE KEY-----
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ cat bob-fake-public-key.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEd46t3j3jyzfw6/xNxQj9Z+i/9j9q
4dmErgXivP5ZOL7de1WR6HFYTPHMYOYeFmWizC6Jwc2Tam/RhnVpqRNYkw==
-----END PUBLIC KEY-----
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$
```

Fig: Generating fake private key for Bob

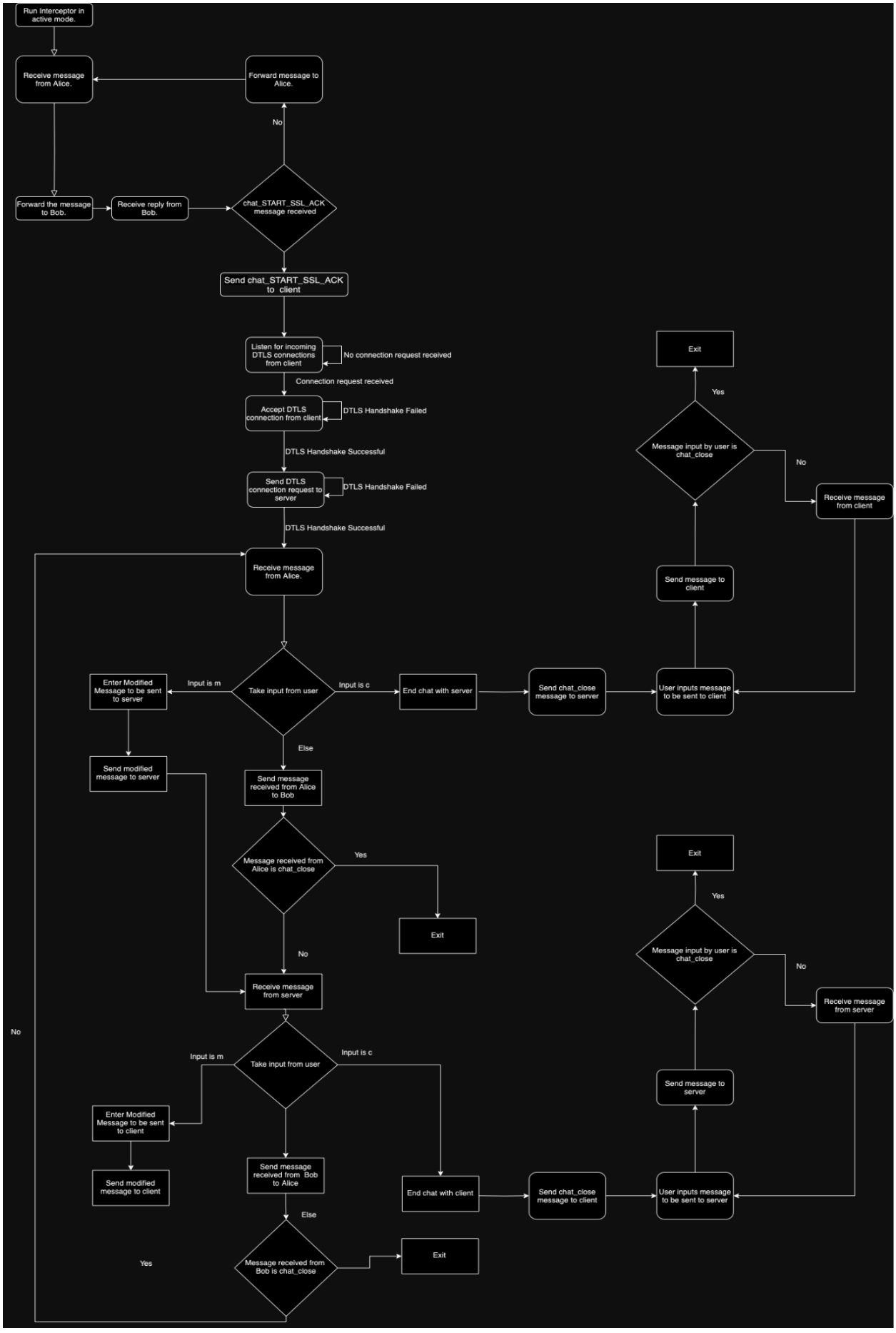
```
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl req -new -key bob-fake-private-key.pem -out bob-fake.csr -config bob.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
IN []:IN
Telangana []:Telangana
Hyderabad []:Hyderabad
IIT Hyderabad []:IIT Hyderabad
CSE []:CSE
Bob1.com []:Bob1.com
cs23mtech11020@iith.ac.in []:cs23mtech11020@iith.ac.in
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl req -in bob-fake.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Bob1.com, emailAddress = cs23mtech11020@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:77:8e:ad:de:3d:e3:cb:37:f0:eb:fc:4d:c5:08:
                    fd:67:e8:bf:f6:3f:6a:e1:d9:84:ae:05:e2:bc:fe:
                    59:38:be:dd:7b:55:91:e8:71:58:4c:f1:cc:60:e6:
                    1e:16:65:a2:cc:2e:89:c1:cd:93:6a:6f:d1:86:75:
                    69:a9:13:58:93
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:81:35:0d:ec:e6:e8:ab:eb:21:db:71:da:49:
        a5:52:e0:30:bf:f5:1b:53:79:ed:76:60:3c:29:1a:75:c9:00:
        f8:02:20:1e:8a:d2:75:75:91:d9:87:be:ac:fd:48:d3:be:ea:
        4d:6c:03:34:3c:b4:d5:06:55:f1:28:b2:84:76:d2:a3:67
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$
```

Fig: Generating CSR for fake Bob certificate

```
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl x509 -req -in bob-fake.csr -CA inter.crt -CAkey private-key-inter.pem -CAcreateser
ial -out bob-fake.crt -days 90 -extfile cert.cnf -extensions v3_ca
Certificate request self-signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Bob1.com, emailAddress = cs23mtech11020@iith.ac.in
ashish@ashish-Lenovo-Legion-5-15ARH05:~/CS23MTECH11004|CS23MTECH11020|CS23MTECH13001/Task_3$ openssl x509 -noout -text -in bob-fake.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:61:c4:33:f0:5f:d7:07:b4:d6:aa:63:8d:fe:17:4f:d0:28:38:9b
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = iTS CA 1R3, emailAddress = iTSinter@iTS.ac.in
        Validity
            Not Before: Apr 10 12:19:28 2024 GMT
            Not After : Jul  9 12:19:28 2024 GMT
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IIT Hyderabad, OU = CSE, CN = Bob1.com, emailAddress = cs23mtech11020@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:77:8e:ad:de:3d:e3:cb:37:f0:eb:fc:4d:c5:08:
                    fd:67:e8:bf:f6:3f:6a:e1:d9:84:ae:05:e2:bc:fe:
                    59:38:be:dd:7b:55:91:e8:71:58:4c:f1:cc:60:e6:
                    1e:16:65:a2:cc:2e:89:c1:cd:93:6a:6f:d1:86:75:
                    69:a9:13:58:93
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Key Agreement
            X509v3 Subject Key Identifier:
                98:50:7E:B7:3A:C0:AA:D2:26:DC:4C:22:D1:E5:2B:13:1F:5B:43:C1
            X509v3 Authority Key Identifier:
                2E:33:0D:B3:13:47:4B:45:F7:59:F7:81:56:7B:42:1F:7F:B9:F7:7F
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        8c:9f:45:37:7c:ea:3d:b9:c2:c5:26:77:d7:e8:08:9c:e0:f7:
        9b:c6:48:a3:60:8f:77:c3:db:52:fd:48:2d:24:5e:92:b6:ba:
        a8:bf:a1:fd:96:43:06:20:b8:c1:04:11:e1:a4:24:73:ba:e3:
        4f:9b:e3:42:1d:f1:c0:89:17:cc:57:20:7e:b6:43:34:fc:a9:
        c5:8c:da:1a:14:82:d0:11:16:3b:6d:62:f9:2d:45:e0:81:cd:
        e3:5a:9e:d6:f9:30:1a:e6:fc:6d:7f:6e:0b:73:e4:0f:01:e2:
        c7:05:79:0d:43:fa:58:41:7b:3e:fc:30:ef:82:11:4b:46:b8:
        df:14:0c:b6:d5:a8:2e:74:72:68:0e:2b:ae:95:61:78:22:57:
```

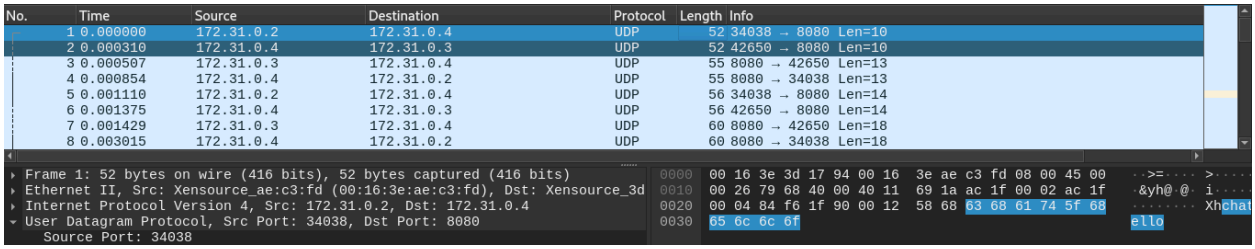
Fig: Using Intermediate certificate to generate fake Bob certificate

b) Flow chart showcasing how the working of our secure_chat_active_interceptor:

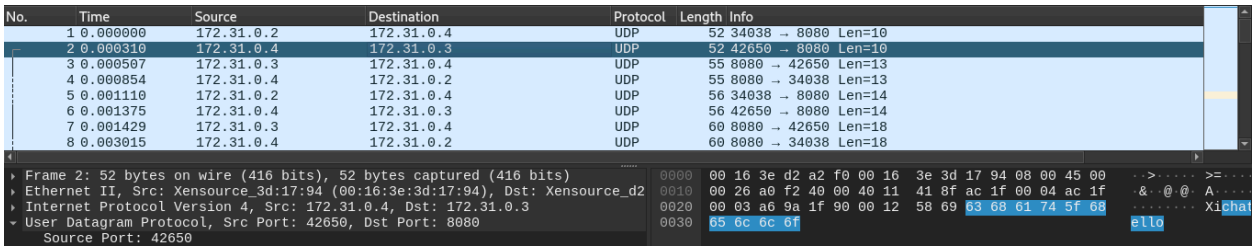

Run Interceptor in active mode.
Receive message from Alice.
Forward message to Alice.
No
Forward the message to Bob.
Receive reply from Bob.
chat_START_SSL_ACK message received
Send chat_START_SSL_ACK to client
Listen for incoming DTLS connections from client
No connection request received
Connection request received
Accept DTLS connection from client
DTLS Handshake Failed
DTLS Handshake Successful
Send DTLS connection request to server
DTLS Handshake Failed
DTLS Handshake Successful
Receive message from Alice.
Take input from user
Input is m
Enter Modified Message to be sent to server
Send modified message to server
Input is c
End chat with server
Send chat_close message to server
User inputs message to be sent to client
Else
Send message received from Alice to Bob
Message received from Alice is chat_close
Yes
Exit
No
Receive message from server
Take input from user
Input is m
Enter Modified Message to be sent to client
Send modified message to client
Input is c
End chat with client
Send chat_close message to client
User inputs message to be sent to server
Else
Send message received from Bob to Alice
Message received from Bob is chat_close
Yes
Exit
No
Exit
Yes
Message input by user is chat_close
No
Receive message from client
Send message to client
Exit
Yes
Message input by user is chat_close
No
Receive message from server
Send message to server

Screenshots of pcap files;

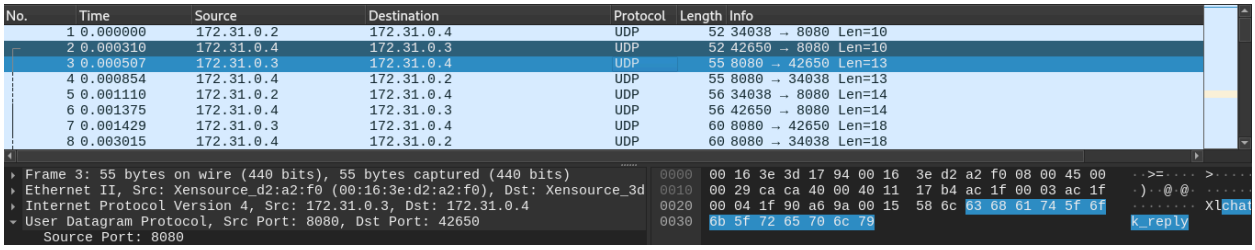
1. ‘chat_hello’ from Alice to Trudy:



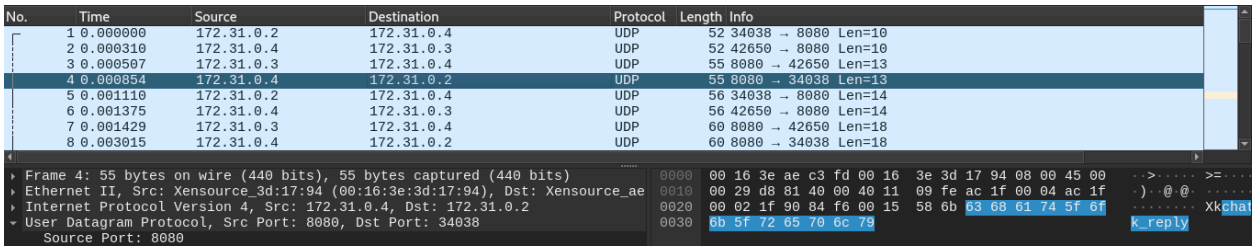
2. ‘Chat_hello’ forwarded from Trudy to Bob:



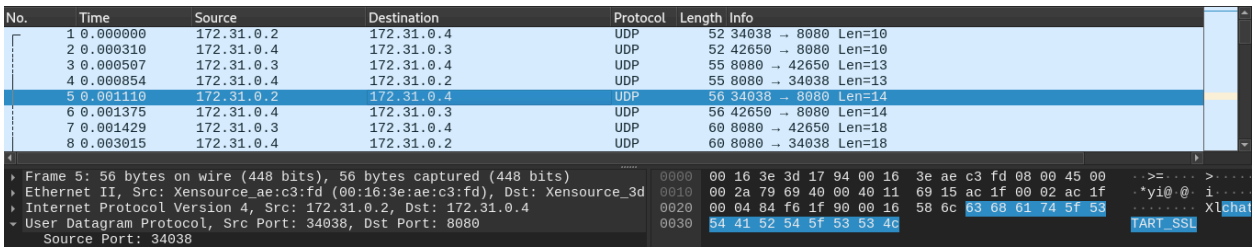
3. ‘Chat_ok_reply’ from Bob to Trudy:



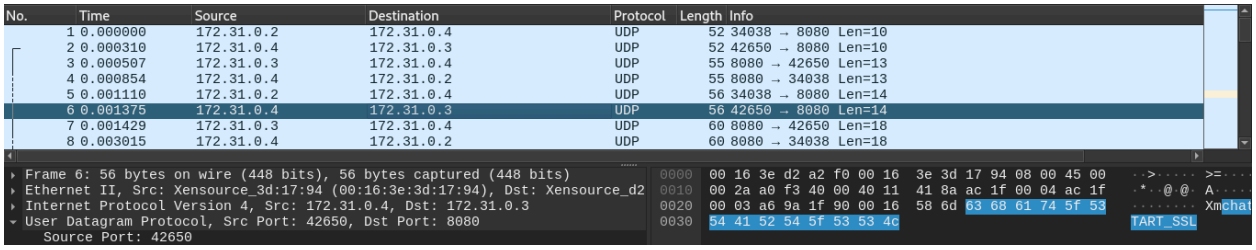
4. Chat_ok_reply’ forwarded from Trudy to Alice:



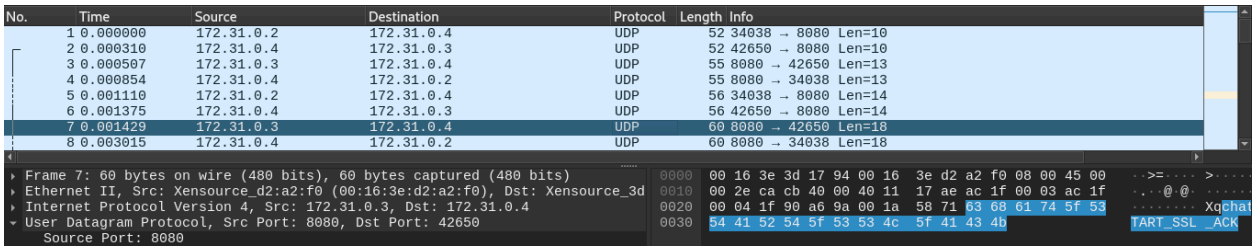
5. ‘chat_START_SSL’ from Alice to Trudy.



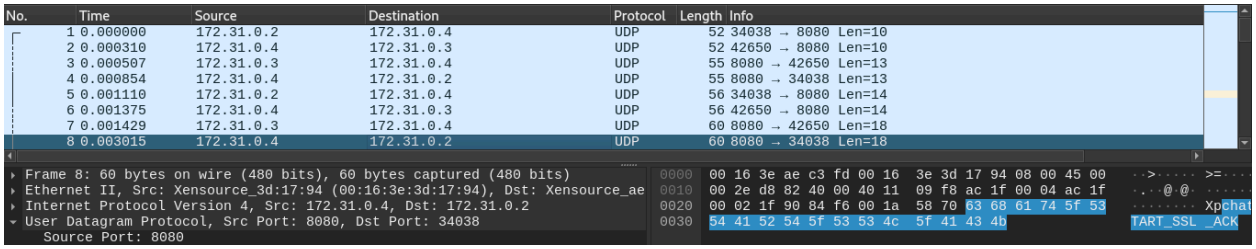
6. ‘Chat_START_SSL’ forwarded from Trudy to Bob:



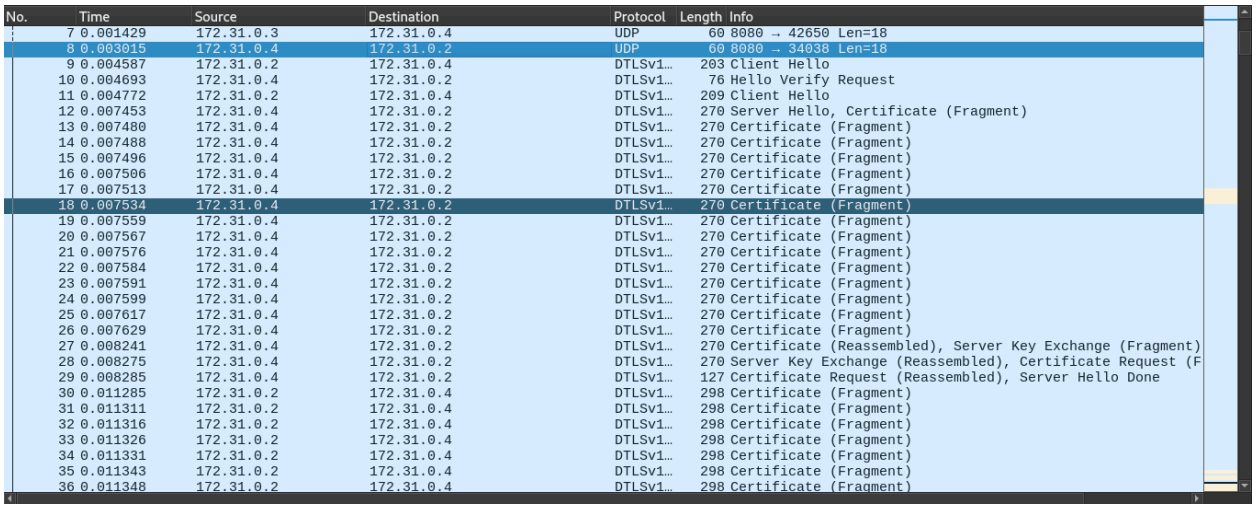
7. ‘chat_START_SSL_ACK’ from Bob to Trudy.



8. ‘chat_START_SSL_ACK’ forwarded from Trudy to Alice:



8. client_hello, server_hello, certificate exchange and session ticket generation.



9. Screenshot showing ‘clent_hello’ being retransmitted multiple times on Trudy’s end as fragments of it got lost because of packet loss.

No.	Time	Source	Destination	Protocol	Length	Info
9	0.004587	172.31.0.2	172.31.0.4	DTLSv1…	203	Client Hello
11	0.004772	172.31.0.2	172.31.0.4	DTLSv1…	209	Client Hello
176	14.950231	172.31.0.4	172.31.0.3	DTLSv1…	203	Client Hello
178	14.950423	172.31.0.4	172.31.0.3	DTLSv1…	209	Client Hello

No.	Time	Source	Destination	Protocol	Length	Info
166	14.948836	172.31.0.4	172.31.0.2	DTLSv1…	121	Certificate Request
167	14.948855	172.31.0.4	172.31.0.2	DTLSv1…	67	Server Hello Done
168	14.949120	172.31.0.4	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
169	14.949154	172.31.0.4	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
170	14.949174	172.31.0.4	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
171	14.949194	172.31.0.4	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
172	14.949213	172.31.0.4	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
173	14.949231	172.31.0.4	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
174	14.949279	172.31.0.4	172.31.0.2	DTLSv1…	274	New Session Ticket (Reassembled), Change Cipher Spec, Enc
175	14.949364	172.31.0.2	172.31.0.4	DTLSv1…	98	Application Data
176	14.950231	172.31.0.4	172.31.0.3	DTLSv1…	203	Client Hello
177	14.950395	172.31.0.3	172.31.0.4	DTLSv1…	76	Hello Verify Request
178	14.950423	172.31.0.4	172.31.0.3	DTLSv1…	209	Client Hello
179	14.955920	172.31.0.3	172.31.0.4	DTLSv1…	270	Server Hello, Certificate (Fragment)
180	14.955943	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
181	14.955959	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
182	14.955974	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
183	14.955990	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
184	14.956005	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
185	14.956020	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
186	14.956035	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
187	14.956049	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
188	14.956075	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
189	14.956089	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
190	14.956110	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
191	14.956124	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
192	14.956161	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
193	14.956176	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Fragment)
194	14.957050	172.31.0.3	172.31.0.4	DTLSv1…	270	Certificate (Reassembled), Server Key Exchange (Fragment)
195	14.957095	172.31.0.3	172.31.0.4	DTLSv1…	202	Server Key Exchange (Reassembled), Certificate Request, S

c) Working of active interceptor (with packet loss introduced at the client and server end):

```
...ssh ubuntu@10.200.33.219   ...ssh ubuntu@10.200.33.219   ...ss
[root@alice1:~# sudo tc qdisc change dev eth0 root netem loss 10%
root@alice1:~#
```

Fig: Change the packet loss in eth0 interface at alice1 container to 10%

```
...ssh ubuntu@10.200.33.219   ...ssh ubuntu@10.200.33.219   ...
[root@bob1:~# sudo tc qdisc change dev eth0 root netem loss 10%
[root@bob1:~# ping alice1
PING alice1 (172.31.0.4) 56(84) bytes of data.
64 bytes from alice1 (172.31.0.4): icmp_seq=1 ttl=64 time=0.335 ms
64 bytes from alice1 (172.31.0.4): icmp_seq=3 ttl=64 time=0.111 ms
64 bytes from alice1 (172.31.0.4): icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from alice1 (172.31.0.4): icmp_seq=5 ttl=64 time=0.071 ms
64 bytes from alice1 (172.31.0.4): icmp_seq=6 ttl=64 time=0.085 ms
64 bytes from alice1 (172.31.0.4): icmp_seq=7 ttl=64 time=0.078 ms
64 bytes from alice1 (172.31.0.4): icmp_seq=8 ttl=64 time=0.081 ms

64 bytes from alice1 (172.31.0.4): icmp_seq=10 ttl=64 time=0.085 ms
64 bytes from alice1 (172.31.0.4): icmp_seq=11 ttl=64 time=0.087 ms
^C
--- alice1 ping statistics ---
11 packets transmitted, 9 received, 18.1818% packet loss, time 10238ms
rtt min/avg/max/mdev = 0.071/0.112/0.335/0.079 ms
root@bob1:~#
```

Fig: Change the packet loss in eth0 interface at bob1 container to 10%

In the above figures we can see how we introduce loss in the containers using the **tc utility.**

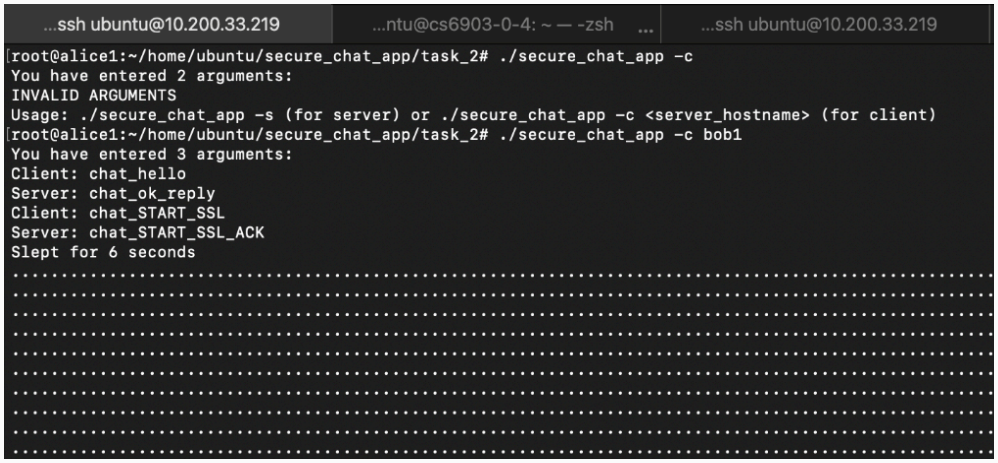


Fig: Chat app running as client end on the alice1 container

In the above figures we can see that we run the chat application compiled as file **secure_chat_app** using the command:
./secure_chat_app -c bob1
The dots in the figure represent fragments of client_hello messages. **Having a long series of dots represents DTLS handshake is being retried.** This shows how we ensure **reliability** of the DTLS handshake in the face of packet loss.

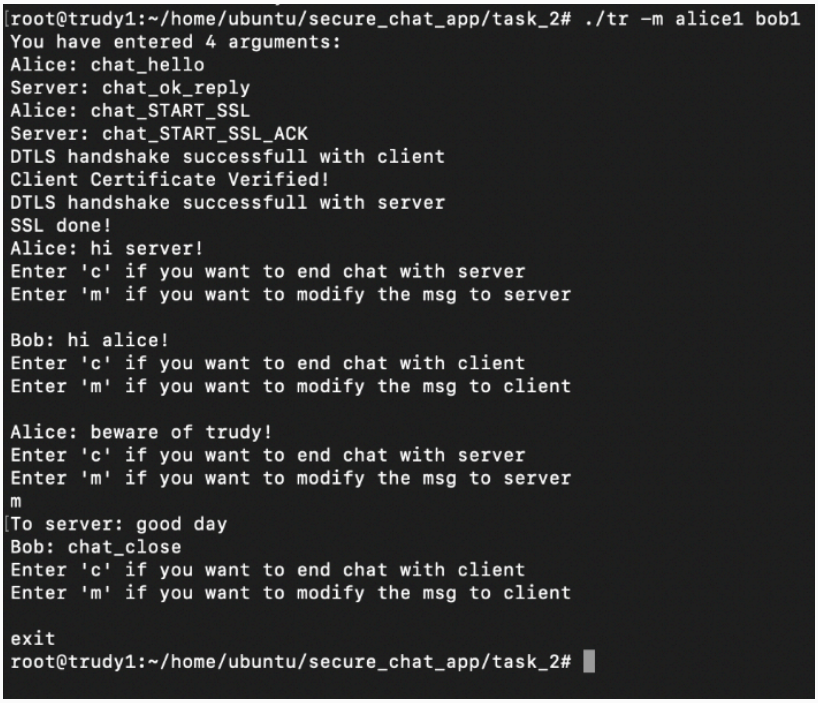


Fig: interceptor app running on trudy1 container

In the above figure we can see that we run the interceptor application compiled as file **tr** using the command:
./tr -m alice1 bob1

We also see the “DTLS handshake successful with client” and “DTLS handshake successful with client” messages. These show the separate DTLS connections Trudy establishes with Alice and Bob respectively.
We can see here that we modify the “beware of trudy” message to “good day” message on the client end.

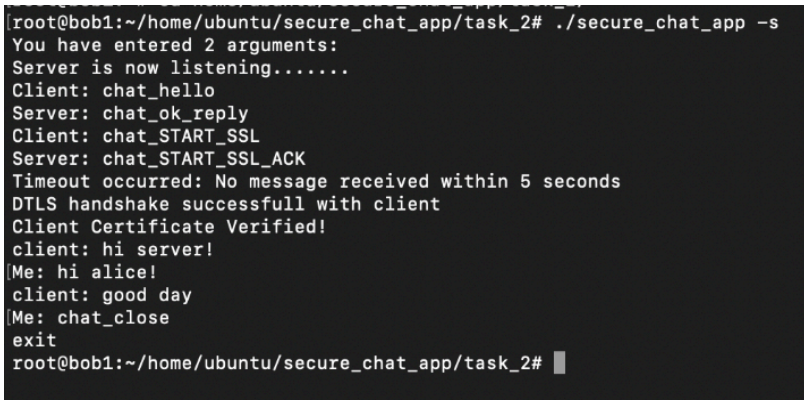
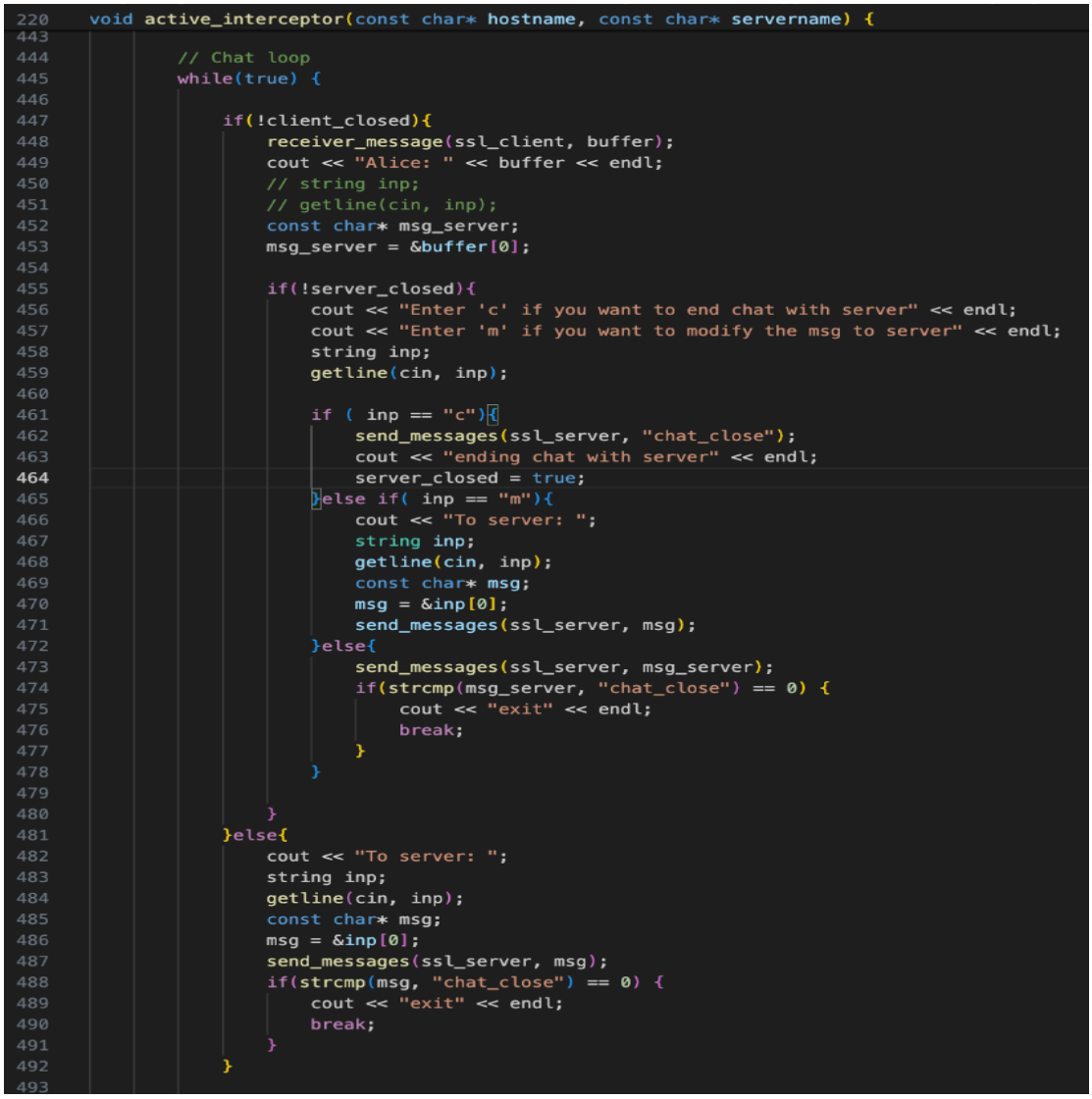


Fig: Chat app running as server end on the bob1 container

In the above figure we can see that we run the chat application compiled as file **secure_chat_app** using the command:
./secure_chat_app -s
We can see the sequence of messages received in the terminal. The messages received by the server are the ones which are finally forwarded/modified and forwarded by Trudy (“good day” message).

Explaining modified code snippet:

1.



```
220  void active_interceptor(const char* hostname, const char* servername) {
443
444          // Chat loop
445          while(true) {
446
447              if(!client_closed){
448                  receiver_message(ssl_client, buffer);
449                  cout << "Alice: " << buffer << endl;
450                  // string inp;
451                  // getline(cin, inp);
452                  const char* msg_server;
453                  msg_server = &buffer[0];
454
455                  if(!server_closed){
456                      cout << "Enter 'c' if you want to end chat with server" << endl;
457                      cout << "Enter 'm' if you want to modify the msg to server" << endl;
458                      string inp;
459                      getline(cin, inp);
460
461                      if ( inp == "c"){
462                          send_messages(ssl_server, "chat_close");
463                          cout << "ending chat with server" << endl;
464                          server_closed = true;
465                      }else if( inp == "m"){
466                          cout << "To server: ";
467                          string inp;
468                          getline(cin, inp);
469                          const char* msg;
470                          msg = &inp[0];
471                          send_messages(ssl_server, msg);
472                      }else{
473                          send_messages(ssl_server, msg_server);
474                          if(strcmp(msg_server, "chat_close") == 0) {
475                              cout << "exit" << endl;
476                              break;
477                          }
478                      }
479
480                  }
481              }else{
482                  cout << "To server: ";
483                  string inp;
484                  getline(cin, inp);
485                  const char* msg;
486                  msg = &inp[0];
487                  send_messages(ssl_server, msg);
488                  if(strcmp(msg, "chat_close") == 0) {
489                      cout << "exit" << endl;
490                      break;
491                  }
492              }
493
```

Fig: Code snippet showcasing how Trudy uses the ‘m’ and ‘c’ user inputs to modify the message coming from Alice.

When Trudy receives the message from Alice. User can enter “m” to modify the message and then input their own message which will be sent to server. If user enters “c” then trudy will send chat_close message to server so the server will close its connection with trudy and further communication will only take place between trudy and alice.

Similar code is used to intercept and modify the messages received from Server.

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: Ashish B Emmanuel, Suryansh Gautam, Anil Kumar Sharma
Date: 10/04/2024
Signature: ABE, SG, AKS