Network Security Assignment No.7: Secure chat using openssl and MITM attacks

Group members:
Name: Hrishikesh Hemke
Roll No.: CS23MTECH14003

Name: Dindorkar Mayuresh Rajesh
Roll No.: CS23MTECH14007

Name: Shrenik Ganguli
Roll No.: CS23MTECH14014

Note:

1. The entire code is implemented in **C programming language** because it offers low-level access to networking APIs, socket operations, and network protocols in great detail.
2. C is more environment-friendly than C++.

TASK 1: Generate Keys and Certificates

Step 1: Generating the keys (.pem)

Root CA:

For generating private key:

openssl ecparam -genkey -name brainpoolP512r1 -out root_private_key.pem

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/root
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl ecparam -genkey -name brainpoolP512r1 -out root_private_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl ec -in root_private_key.pem -text -noout
read EC key
Private-Key: (512 bit)
priv:
    9c:b7:90:96:2e:be:08:c1:b1:ef:3d:d4:e2:c2:69:
    25:99:e6:e4:fb:ef:a8:c7:47:fc:e7:bb:dc:c9:6b:
    78:c8:48:cf:d2:fd:d6:b3:73:4c:65:b9:ff:f9:e8:
    05:c8:1c:43:e2:06:a1:5f:21:3f:3c:96:26:db:7f:
    af:8a:8a:6d
pub:
    04:2f:04:ea:e9:61:ae:29:35:9f:8f:14:64:5b:9a:
    3c:49:19:f2:e5:8b:a9:fe:60:bf:46:af:66:72:49:
    89:46:44:49:af:49:0f:b1:1e:25:33:1c:87:9b:0e:
    c5:c0:46:02:23:8f:3a:51:ab:96:1c:8d:d1:ef:9f:
    3c:a1:6b:08:79:a1:db:17:11:57:24:27:fb:50:b6:
    30:a2:61:5c:2f:3c:4c:d0:f8:c2:36:71:3f:61:64:
    38:9c:4c:a3:57:a3:d8:fb:3a:8c:f3:ae:d8:b5:f1:
    24:7f:e9:41:a1:f0:ee:1b:31:fd:63:f0:bf:6a:d5:
    dc:17:87:a0:af:9d:81:45:7c
ASN1 OID: brainpoolP512r1
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

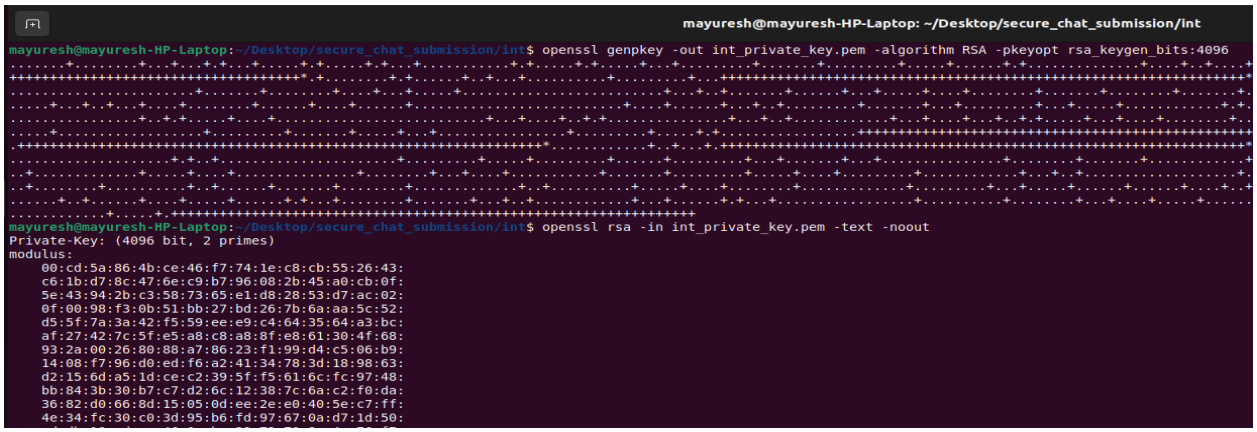
For extracting public key:

openssl ec -in root_private_key.pem -pubout -out root_public_key.pem

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl ec -in root_private_key.pem -pubout -out root_public_key.pem
read EC key
writing EC key
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ cat root_public_key.pem
-----BEGIN PUBLIC KEY-----
MIGbMBQGByqGSM49AgEGCSskAwMCCAEBDQOBggAELwTq6WGuKTWfjxRkW5o8SRny
5Yup/mC/Rq9mckmJRkRJr0kPsR4lMxyHmw7FwEYCI486UauWHI3R7588oWsIeaHb
FxFXJCf7ULYwomFcLzxM0PjCNnE/YWQ4nEyjV6PY+zqM867YtfEkf+lBofDuGzH9
Y/C/atXcF4egr52BRXw=
-----END PUBLIC KEY-----
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

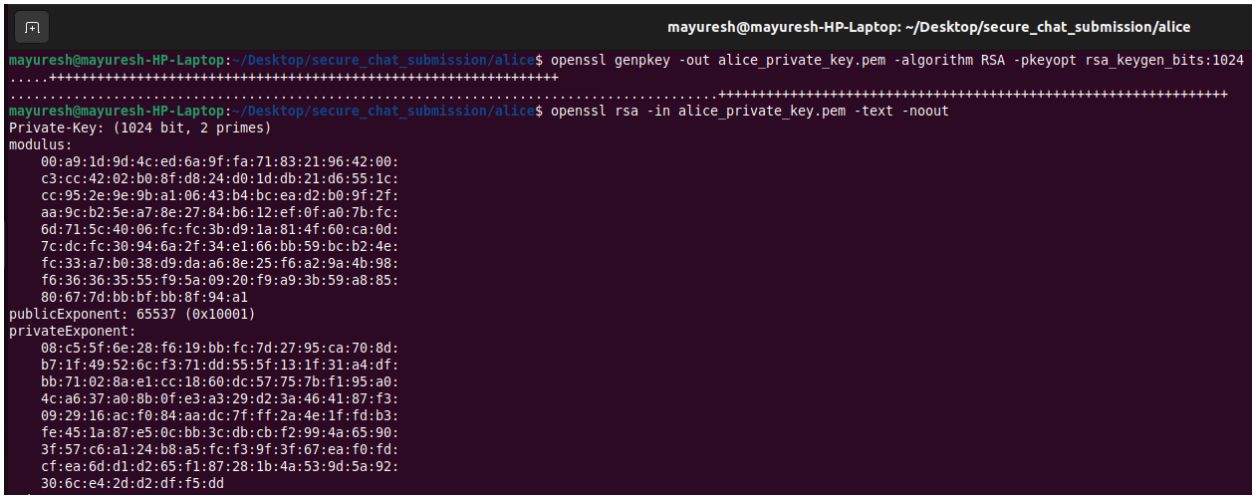
Intermediate CA:

openssl genpkey -out int_private_key.pem -algorithm RSA -pkeyopt rsa_keygen_bits:4096



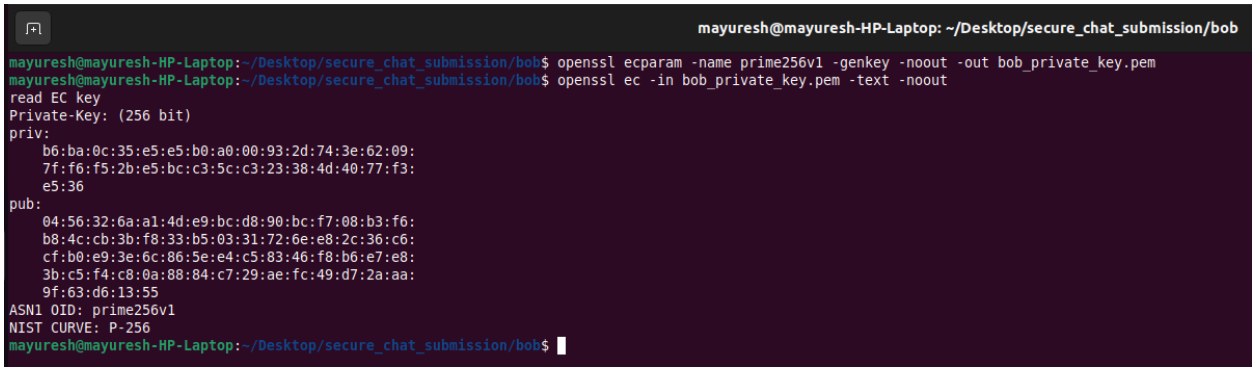
Alice:

openssl genpkey -out alice_private_key.pem -algorithm RSA -pkeyopt rsa_keygen_bits:1024



Bob:

openssl ecparam -name prime256v1 -genkey -noout -out bob_private_key.pem



Step 2: Creating the configuration(.cnf) and extension files(.txt)

RootCA:
Config file: root.cnf

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/root
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ cat root.cnf
[req]
prompt = no
distinguished_name = root_ca
input_password = root

[root_ca]
CN = iTS ROOT R1
emailAddress = rootCA@gmail.com
O = iTS ROOT R1
L = Hyderabad
C = IN
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

Extension file: root.txt

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ cat root.txt
authorityKeyIdentifier = keyid:always,issuer:always
basicConstraints       = CA:TRUE
keyUsage               = critical,keyCertSign,cRLSign
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

Intermediate CA:
Config file: int.cnf

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat int.cnf
[req]
prompt = no
distinguished_name = int_ca
input_password = int

[int_ca]
CN = iTS CA 1R3
emailAddress = intCA@gmail.com
O = iTS CA 1R3
L = Nashik
C = IN
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

Extension file: int.txt

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat int.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

Alice:

Config file: alice.cnf

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/alice
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$ cat alice.cnf
[req]
prompt = no
distinguished_name = alice_dn
input_password = alice

[alice_dn]
CN = Alice1.com
emailAddress = Alice1@gmail.com
O = Alice1
L = Nagpur
C = IN
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$
```

Extension file: alice.txt

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/alice
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$ cat alice.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints       = CA:FALSE
keyUsage               = digitalSignature,nonRepudiation,keyEncipherment,dataEncipherment

mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$
```

Bob:

Config file: bob.cnf

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/bob
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ cat bob.cnf
[req]
prompt = no
distinguished_name = bob_dn
input_password = bob

[bob_dn]
CN = Bob1.com
emailAddress = Bob1@gmail.com
O = Bob1
L = Kolkata
C = IN
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$
```

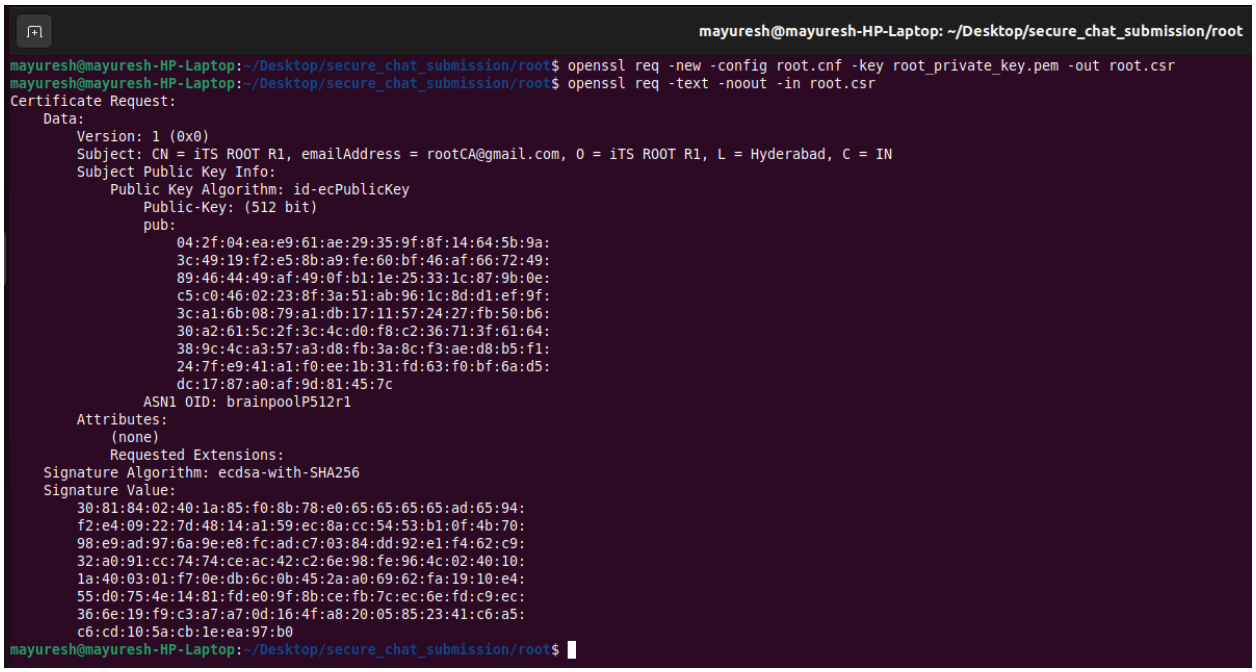
Extension file: bob.txt

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/bob
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ cat bob.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints       = CA:FALSE
keyUsage               = digitalSignature,nonRepudiation,keyEncipherment,dataEncipherment
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$
```

Step 3: Generating the Certificate Signing Request (.csr):

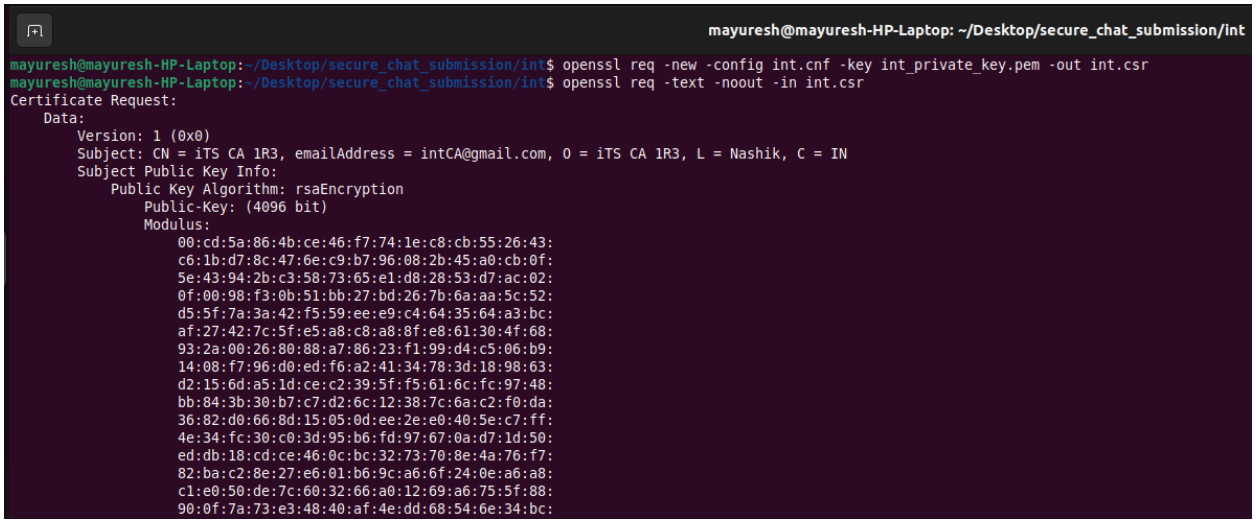
Root CA:

openssl req -new -config root.cnf -key root_private_key.pem -out root.csr



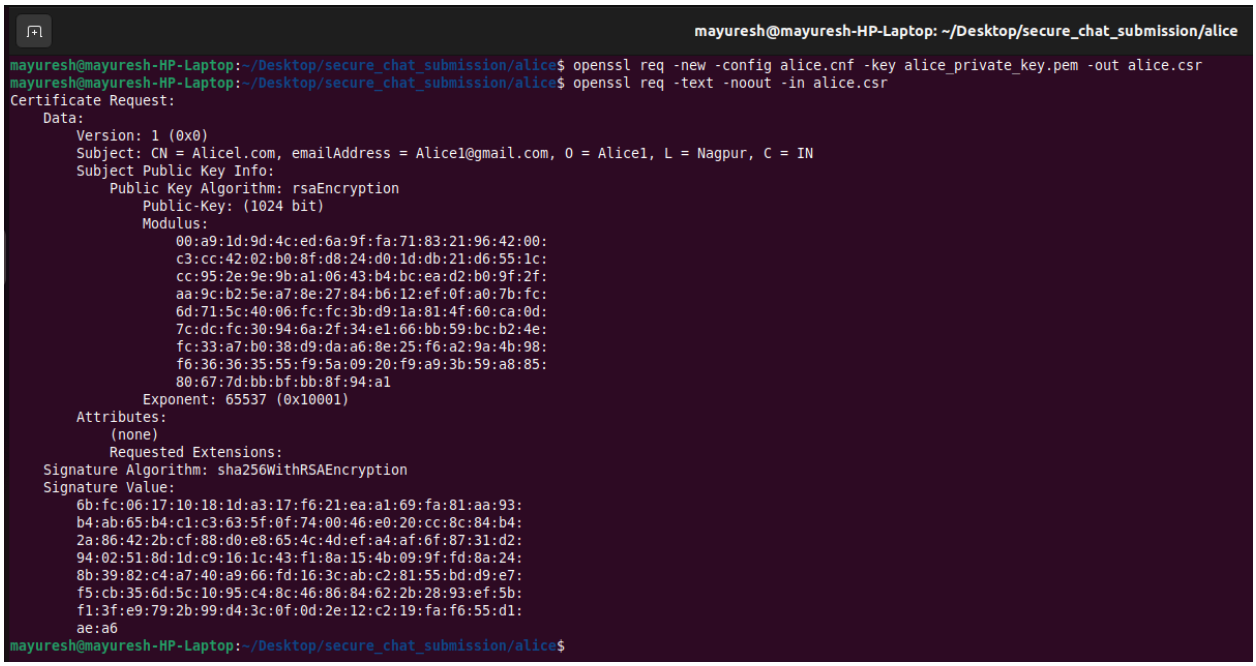
Intermediate CA:

openssl req -new -config int.cnf -key int_private_key.pem -out int.csr



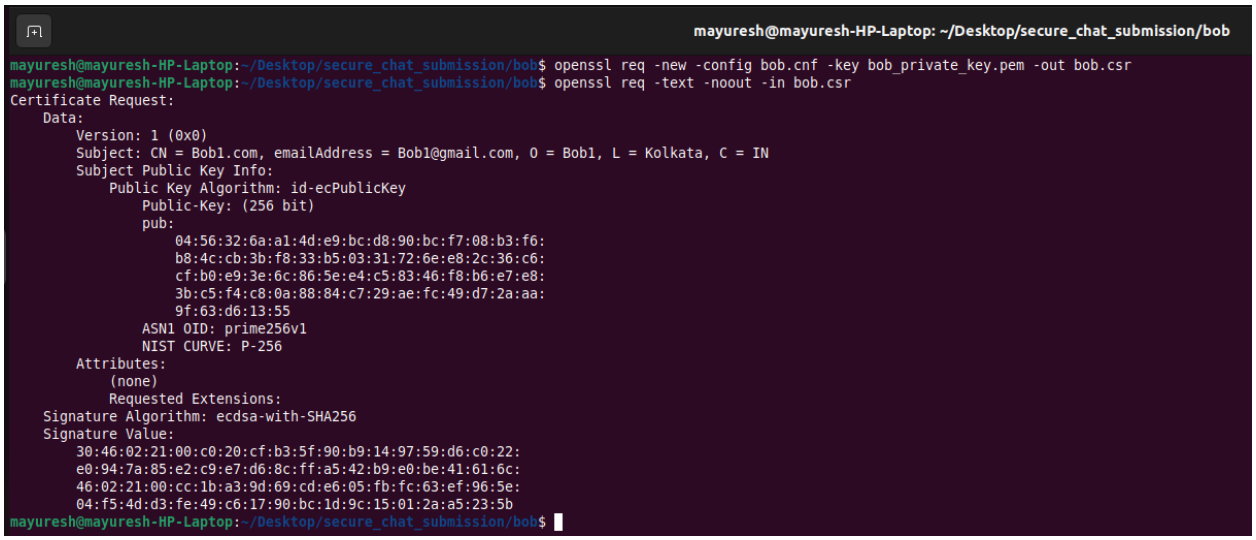
Alice:

openssl req -new -config alice.cnf -key alice_private_key.pem -out alice.csr



Bob:

openssl req -new -config bob.cnf -key bob_private_key.pem -out bob.csr



Step 4: Checking the Integrity of received CSR:

Intermediate CA:

a. **Creating digest of 'int.csr':**
openssl req -in int.csr -noout -text | openssl dgst -md5 -out int_csr_digest

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/int
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl req -in int.csr -noout -text | openssl dgst -md5 -out int_csr_digest
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat int_csr_digest
MD5(stdin)= 6d5f82e9809518503b789960dfabbdef
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

b. **Signing the digest with 'int_private_key.pem':**
openssl pkeyutl -sign -in int_csr_digest -out int_csr_sign -inkey int_private_key.pem

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl pkeyutl -sign -in int_csr_digest -out int_csr_sign -inkey int_private_key.pem
```

c. **Sent 'int.csr' and 'int_csr_sign' to Root CA**

d. **RootCA creates digest of 'int.csr'**
openssl req -in int.csr -noout -text | openssl dgst -md5 -out int_csr_digest_generated

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission/root
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl req -in int.csr -noout -text | openssl dgst -md5 -out int_csr_digest_generated
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ ls
int.csr  int_csr_digest_generated  int_csr_sign  root.cnf  root.csr  root_private_key.pem  root.txt
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

e. **RootCA extracts the public key from 'int.csr':**
openssl req -in int.csr -pubkey -noout > int_public_key.pem

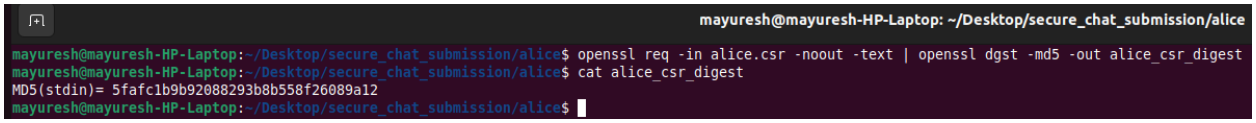
```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl req -in int.csr -pubkey -noout > int_public_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ cat int_public_key.pem
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAzVqGS85G93QeyMtVJkPG
G9eMR27Jt5YIK0Wgyw9eQ5Qrw1hzZeHYKFPXrAIPAJjzC1G7J70me2qqXFLVX3o6
QvVZ7unEZDVko7yvJ0J8X+WoyKiP6GEwT2iTKgAmgIinhiPxmdTFBrkUCPeW0O32
okE0eD0YmGPSFW2lHc7COV/1YWz8l0i7hDswt8fSbBI4fGrC8No2gtBmjRUFDe4u
4EBex/9ONPwwwD2Vtv2XZwrXHVDt2xjNzkYMvDJzcI5KdveCusKOJ+YBtpymbyQO
pqjB4FDefGAyZqASaaZ1X4iQD3pz40hAr07daFRuNLyln9CDX9BRWY3XsK0L0zSd
1BTjXrcJXYOtvvU8Q+34D5zh24vhvoEwnVyUE80aQ7KDgGA0pkCx2ROgYuw8pv27
1NbbPKo+VI8dHH6R0p4O6hOUtpNE/xqit5c46B7bG5x+ddVpV6URAoxn+7qdEshh
nQWi3DR5yqfKLxkHCF1+gxtafm/efLDj1qauMAXnQgUIxwp25U7IZr9HovJU5Rim
tVb0CMxf62q8yF6ufUZlRnoDWiJJTOnA4WHgKtHaqiiAUzD4EEwMriCx01EZDyP8
YwBVOIBVJ5/qdXQPmqHa5lrByPQfNY5iuvmvE5lARkIyXQewjPNrpObCcj2OPL2A
4UD16YwSmXF/URrfX6qiHOsCAwEAAQ==
-----END PUBLIC KEY-----
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

f. **RootCA verifying the authenticity of digest (i.e., verifying the authenticity of csr):**
openssl pkeyutl -verify -sigfile int_csr_sign -in int_csr_digest_generated -inkey int_public_key.pem -pubin

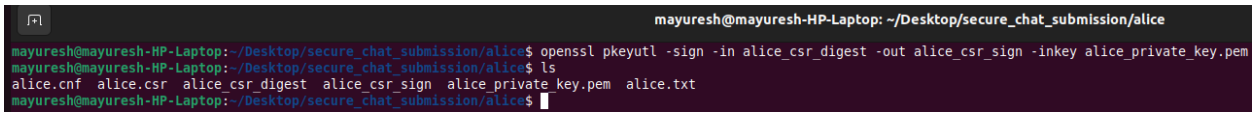
```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl pkeyutl -verify -sigfile int_csr_sign -in int_csr_digest_generated -inkey int_public_key.pem -pubin
Signature Verified Successfully
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

Alice:

a. **Creating digest of 'alice.csr':**
openssl req -in alice.csr -noout -text | openssl dgst -md5 -out alice_csr_digest

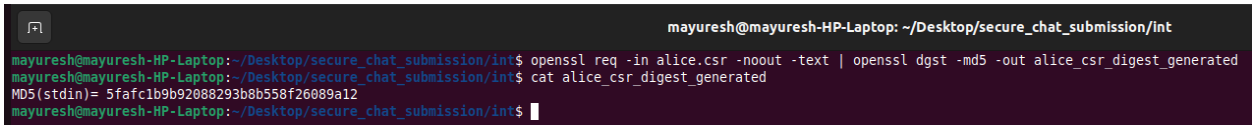


b. **Signing the digest with 'alice_private_key.pem':**
openssl pkeyutl -sign -in alice_csr_digest -out alice_csr_sign -inkey alice_private_key.pem

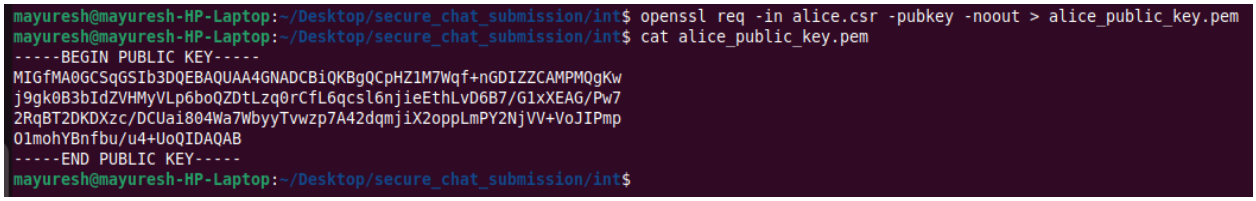


c. **Sent 'alice.csr' and 'alice_csr_sign' to Intermediate CA**

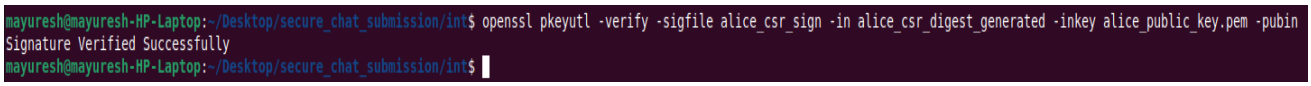
d. **Intermediate CA creates digest of 'alice.csr'**
openssl req -in alice.csr -noout -text | openssl dgst -md5 -out alice_csr_digest_generated



e. **Intermediate CA extracts the public key from 'alice.csr':**
openssl req -in alice.csr -pubkey -noout > alice_public_key.pem



f. **Intermediate CA verifying the authenticity of digest (i.e., verifying the authenticity of csr):**
openssl pkeyutl -verify -sigfile alice_csr_sign -in alice_csr_digest_generated -inkey alice_public_key.pem -pubin



Bob:

a. **Creating digest of 'bob.csr':**
openssl req -in bob.csr -noout -text | openssl dgst -md5 -out bob_csr_digest

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ openssl req -in bob.csr -noout -text | openssl dgst -md5 -out bob_csr_digest
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ cat bob_csr_digest
MD5(stdin)= d2b234b5eeb6e918319a6ebc322ae937
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$
```

b. **Signing the digest with 'bob_private_key.pem':**
openssl pkeyutl -sign -in bob_csr_digest -out bob_csr_sign -inkey bob_private_key.pem

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ openssl pkeyutl -sign -in bob_csr_digest -out bob_csr_sign -inkey bob_private_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ ls
bob.cnf  bob.csr  bob_csr_digest  bob_csr_sign  bob_private_key.pem  bob.txt
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$
```

c. **Sent 'bob.csr' and 'bob_csr_sign' to Intermediate CA**

d. **Intermediate CA creates digest of 'bob.csr'**
openssl req -in bob.csr -noout -text | openssl dgst -md5 -out bob_csr_digest_generated

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl req -in bob.csr -noout -text | openssl dgst -md5 -out bob_csr_digest_generated
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat bob_csr_digest_generated
MD5(stdin)= d2b234b5eeb6e918319a6ebc322ae937
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

e. **Intermediate CA extracts the public key from bob.csr':**
openssl req -in bob.csr -pubkey -noout > bob_public_key.pem

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl req -in bob.csr -pubkey -noout > bob_public_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat bob_public_key.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVjJqoU3pvNiQvPcIs/a4TMs7+DO1
AzFybugsNsbPsOk+bIZe5MWDRvi25+g7xfTICoiExymu/EnXKqqfY9YTVQ==
-----END PUBLIC KEY-----
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

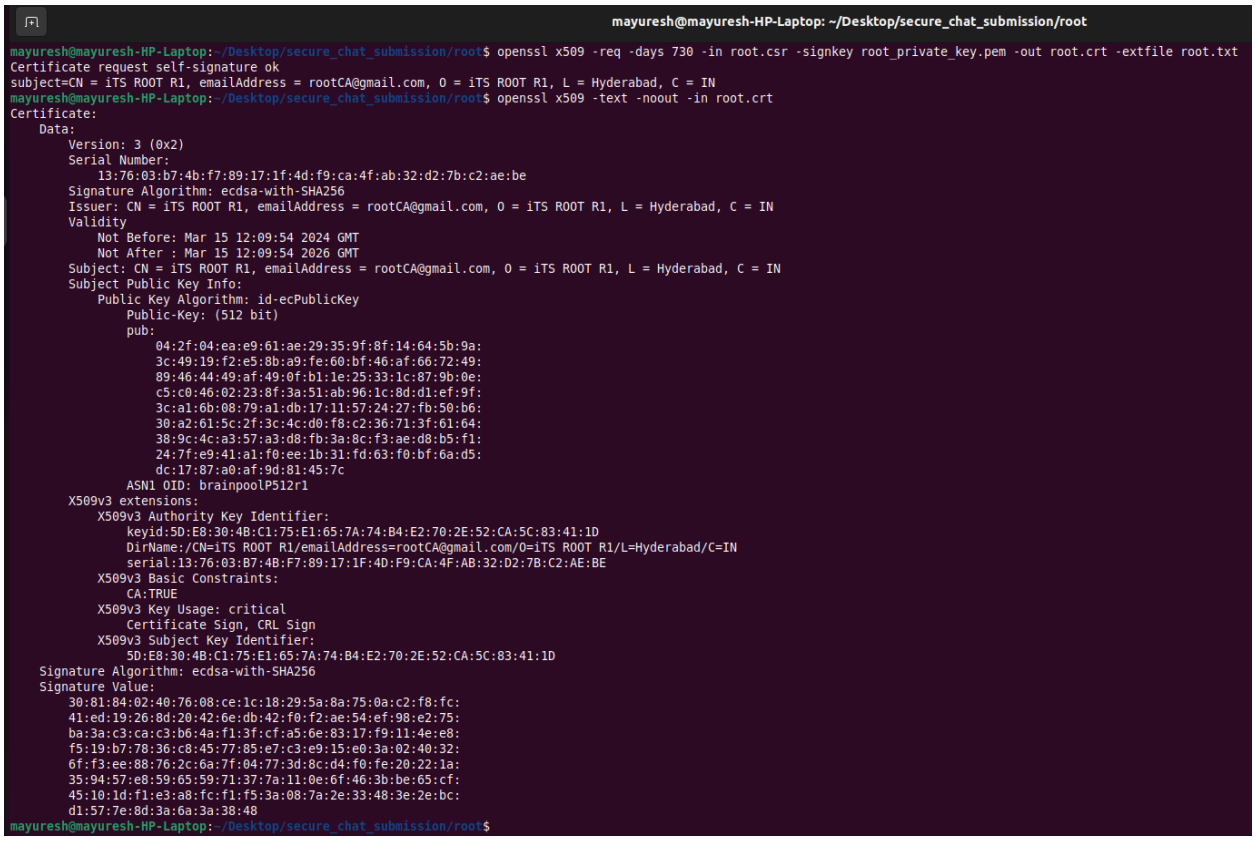
f. **Intermediate CA verifying the authenticity of digest (i.e., verifying the authenticity of csr):**
openssl pkeyutl -verify -sigfile bob_csr_sign -in bob_csr_digest_generated -inkey bob_public_key.pem -pubin

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl pkeyutl -verify -sigfile bob_csr_sign -in bob_csr_digest_generated -inkey bob_public_key.pem -pubin
Signature Verified Successfully
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

Step 5: Creating the Certificate (.crt) by signing CSR:

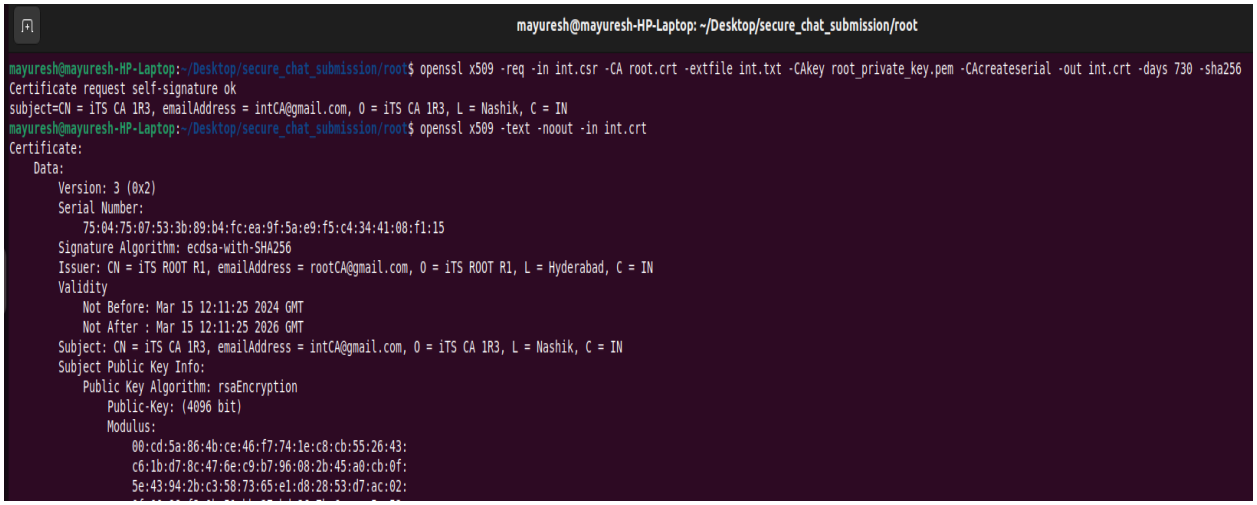
RootCA (self-signed):

openssl x509 -req -days 730 -in root.csr -signkey root_private_key.pem -out root.crt -extfile root.txt



IntermediateCA (signed by RootCA):

openssl x509 -req -in int.csr -CA root.crt -extfile int.txt -CAkey root_private_key.pem -CAcreateserial -out int.crt -days 730 -sha256



Alice (signed by IntermediateCA):

openssl x509 -req -in alice.csr -CA int.crt -extfile alice.txt -CAkey int_private_key.pem -CAcreateserial -out alice.crt -days 730 -sha256

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl x509 -req -in alice.csr -CA int.crt -extfile alice.txt -CAkey int_private_key.pem -CAcreateserial -out alice.crt -days 730 -sha256
Certificate request self-signature ok
subject=CN = Alice1.com, emailAddress = Alice1@gmail.com, O = Alice1, L = Nagpur, C = IN
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl x509 -text -noout -in alice.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            40:fb:c7:e9:9b:f7:0f:99:a5:f2:5c:07:94:89:4e:bd:51:a0:17:1f
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = iTS CA 1R3, emailAddress = intCA@gmail.com, O = iTS CA 1R3, L = Nashik, C = IN
        Validity
            Not Before: Mar 15 12:16:01 2024 GMT
            Not After : Mar 15 12:16:01 2026 GMT
        Subject: CN = Alice1.com, emailAddress = Alice1@gmail.com, O = Alice1, L = Nagpur, C = IN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:a9:1d:9d:4c:ed:6a:9f:fa:71:83:21:96:42:00:
                    c3:cc:42:02:b0:8f:d8:24:d0:1d:db:21:d6:55:1c:
                    cc:95:2e:9e:9b:a1:06:43:b4:bc:ea:d2:b0:9f:2f:
                    aa:9c:b2:5e:a7:8e:27:84:b6:12:ef:0f:a0:7b:fc:
                    6d:71:5c:40:06:fc:fc:3b:d9:1a:81:4f:60:ca:0d:
                    7c:dc:fc:30:94:6a:2f:34:e1:66:bb:59:bc:b2:4e:
                    fc:33:a7:b0:38:d9:da:a6:8e:25:f6:a2:9a:4b:98:
                    f6:36:36:35:55:f9:5a:09:20:f9:a9:3b:59:a8:85:
                    80:67:7d:bb:bf:bb:8f:94:a1
```

Bob (signed by IntermediateCA):

openssl x509 -req -in bob.csr -CA int.crt -extfile bob.txt -CAkey int_private_key.pem -CAcreateserial -out bob.crt -days 730 -sha256

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl x509 -req -in bob.csr -CA int.crt -extfile bob.txt -CAkey int_private_key.pem -CAcreateserial -out bob.crt -days 730 -sha256
Certificate request self-signature ok
subject=CN = Bob1.com, emailAddress = Bob1@gmail.com, O = Bob1, L = Kolkata, C = IN
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl x509 -text -noout -in bob.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            47:4d:db:73:b3:57:12:2b:1f:22:c2:fb:11:2c:ac:f4:80:2c:01:4d
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = iTS CA 1R3, emailAddress = intCA@gmail.com, O = iTS CA 1R3, L = Nashik, C = IN
        Validity
            Not Before: Mar 15 12:17:16 2024 GMT
            Not After : Mar 15 12:17:16 2026 GMT
        Subject: CN = Bob1.com, emailAddress = Bob1@gmail.com, O = Bob1, L = Kolkata, C = IN
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:56:32:6a:a1:4d:e9:bc:d8:90:bc:f7:08:b3:f6:
                    b8:4c:cb:3b:f8:33:b5:03:31:72:6e:e8:2c:36:c6:
                    cf:b0:e9:3e:6c:86:5e:e4:c5:83:46:f8:b6:e7:e8:
                    3b:c5:f4:c8:0a:88:84:c7:29:ae:fc:49:d7:2a:aa:
                    9f:63:d6:13:55
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                75:43:D6:5C:11:5F:47:AF:92:87:D0:2B:DA:9B:E3:DF:CF:81:54:6B
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment, Data Encipherment
            X509v3 Subject Key Identifier:
                BE:57:0B:B9:84:DB:08:34:12:78:46:5F:00:1F:FE:33:0D:2D:D6:91
    Signature Algorithm: sha256WithRSAEncryption
```

Step 6: Checking the Integrity of received CRT:

Intermediate CA:

a. **RootCA creating digest of 'int.crt':**
openssl x509 -in int.crt -outform DER | openssl dgst -md5 > int_crt_digest

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl x509 -in int.crt -outform DER | openssl dgst -md5 > int_crt_digest
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ cat int_crt_digest
MD5(stdin)= 6d7038f80387961e7a26b76cd8c6dc0a
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

b. **RootCA signing the digest with 'root_private_key.pem':**
openssl pkeyutl -sign -in int_crt_digest -out int_crt_sign -inkey root_private_key.pem

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl pkeyutl -sign -in int_crt_digest -out int_crt_sign -inkey root_private_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ ls
int.crt  int_crt_digest  int_crt_sign  int.csr  int_csr_digest_generated  int_csr_sign  int_public_key.pem  int.txt  root.cnf  root.crt  root.csr  root_private_key.pem  root.txt
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

c. **RootCA sent 'int.crt' and 'int_crt_sign' to Intermediate CA**

d. **Intermediate CA creates digest of 'int.crt'**
openssl x509 -in int.crt -outform DER | openssl dgst -md5 > int_crt_digest_generated

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl x509 -in int.crt -outform DER | openssl dgst -md5 > int_crt_digest_generated
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat int_crt_digest_generated
MD5(stdin)= 6d7038f80387961e7a26b76cd8c6dc0a
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

e. **RootCA's public key is known to Intermediate CA**

f. **Verifying the authenticity of the certificate:**
openssl pkeyutl -verify -sigfile int_crt_sign -in int_crt_digest_generated -inkey root_public_key.pem -pubin

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl pkeyutl -verify -sigfile int_crt_sign  -in int_crt_digest_generated -inkey root_public_key.pem -pubin
Signature Verified Successfully
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

Alice:

a. **Intermediate CA creating digest of 'alice.crt':**
openssl x509 -in alice.crt -outform DER | openssl dgst -md5 > alice_crt_digest

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl x509 -in alice.crt -outform DER | openssl dgst -md5 > alice_crt_digest
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat alice_crt_digest
MD5(stdin)= 3456432a0c9f05ed097eb8b61c9de164
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

b. **Intermediate CA signing the digest with 'int_private_key.pem':**
openssl pkeyutl -sign -in alice_crt_digest -out alice_crt_sign -inkey int_private_key.pem

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl pkeyutl -sign -in alice_crt_digest -out alice_crt_sign -inkey int_private_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ ls | grep int_private_key.pem
int_private_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

c. **Intermediate CA sent 'alice.crt' and 'alice_crt_sign' to Alice**

d. **Alice creates digest of 'alice.crt':**
openssl x509 -in alice.crt -outform DER | openssl dgst -md5 > alice_crt_digest_generated

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$ openssl x509 -in alice.crt -outform DER | openssl dgst -md5 > alice_crt_digest_generated
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$ cat alice_crt_digest_generated
MD5(stdin)= 3456432a0c9f05ed097eb8b61c9de164
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$
```

e. **Intermediate CA's public key is known to Alice**
f. **Verifying the authenticity of the certificate:**
openssl pkeyutl -verify -sigfile alice_crt_sign -in alice_crt_digest_generated -inkey int_public_key.pem -pubin

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$ openssl pkeyutl -verify -sigfile alice_crt_sign  -in alice_crt_digest_generated -inkey int_public_key.pem -pubin
Signature Verified Successfully
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$
```

Bob:

a. **Intermediate CA creating digest of 'bob.crt':**
openssl x509 -in bob.crt -outform DER | openssl dgst -md5 > bob_crt_digest

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl x509 -in bob.crt -outform DER | openssl dgst -md5 > bob_crt_digest
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ cat bob_crt_digest
MD5(stdin)= 29428a6bcc7d6ddfbb57bc69f71ebd21
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

b. **Intermediate CA signing the digest with 'int_private_key.pem':**
openssl pkeyutl -sign -in bob_crt_digest -out bob_crt_sign -inkey int_private_key.pem

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl pkeyutl -sign -in bob_crt_digest -out bob_crt_sign -inkey int_private_key.pem
```

c. **Intermediate CA sent 'bob.crt' and 'bob_crt_sign' to Alice**
d. **Bob creates digest of 'bob.crt':**
openssl x509 -in bob.crt -outform DER | openssl dgst -md5 > bob_crt_digest_generated

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ openssl x509 -in bob.crt -outform DER | openssl dgst -md5 > bob_crt_digest_generated
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ cat bob_crt_digest_generated
MD5(stdin)= 29428a6bcc7d6ddfbb57bc69f71ebd21
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$
```

e. **Intermediate CA's public key is known to Alice**
f. **Verifying the authenticity of the certificate:**
openssl pkeyutl -verify -sigfile bob_crt_sign -in bob_crt_digest_generated -inkey int_public_key.pem -pubin

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ openssl pkeyutl -verify -sigfile bob_crt_sign  -in bob_crt_digest_generated -inkey int_public_key.pem -pubin
Signature Verified Successfully
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$
```

Step 7: Verifying the Certificate Chain for RootCA, InterCA, Alice and Bob

RootCA:

openssl verify -CAfile root.crt root.crt

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$ openssl verify -CAfile root.crt root.crt
root.crt: OK
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/root$
```

IntermediateCA:

openssl verify -CAfile root.crt int.crt

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$ openssl verify -CAfile root.crt int.crt
int.crt: OK
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/int$
```

Alice:

openssl verify -CAfile root.crt -untrusted int.crt alice.crt

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$ openssl verify -CAfile root.crt -untrusted int.crt alice.crt
alice.crt: OK
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/alice$
```

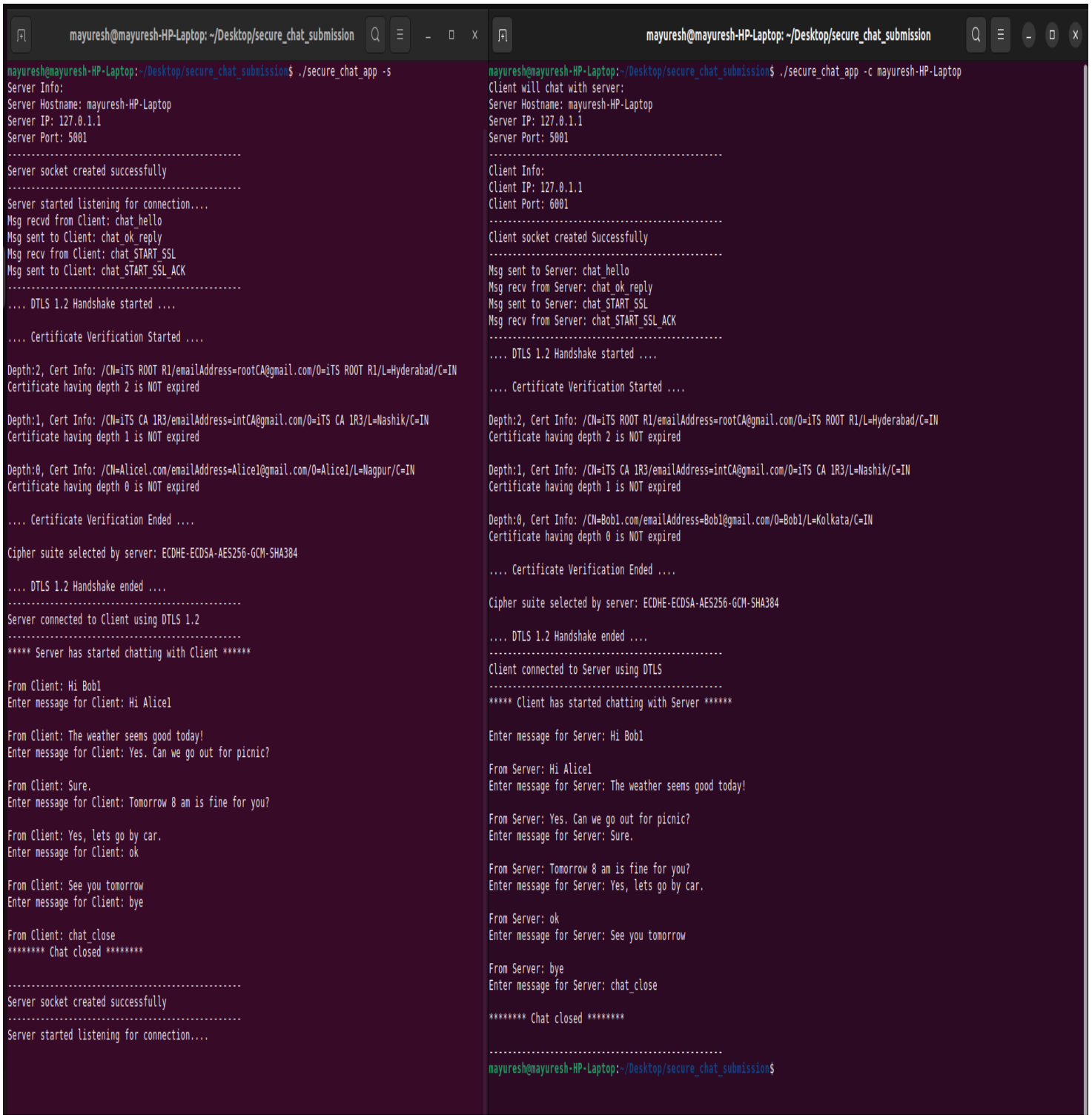
Bob:

openssl verify -CAfile root.crt -untrusted int.crt bob.crt

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$ openssl verify -CAfile root.crt -untrusted int.crt bob.crt
bob.crt: OK
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission/bob$
```

TASK 2: Secure Chat App

Secure Chat Output:

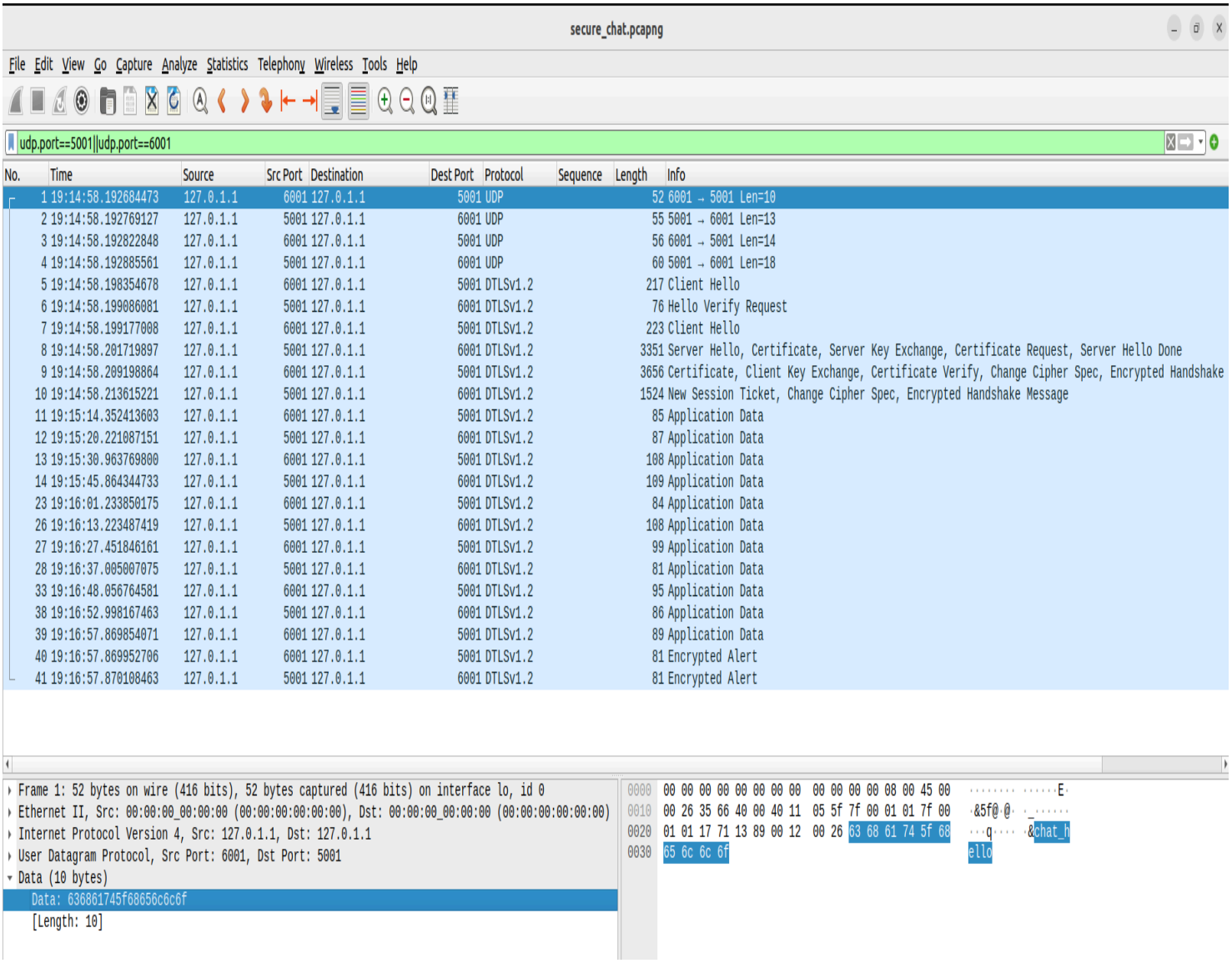


```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission

mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission$ ./secure_chat_app -s
Server Info:
Server Hostname: mayuresh-HP-Laptop
Server IP: 127.0.1.1
Server Port: 5001
------------------------------------------------
Server socket created successfully
------------------------------------------------
Server started listening for connection....
Msg recvd from Client: chat_hello
Msg sent to Client: chat_ok_reply
Msg recv from Client: chat_START_SSL
Msg sent to Client: chat_START_SSL_ACK
------------------------------------------------
.... DTLS 1.2 Handshake started ....

.... Certificate Verification Started ....

Depth:2, Cert Info: /CN=iTS ROOT R1/emailAddress=rootCA@gmail.com/O=iTS ROOT R1/L=Hyderabad/C=IN
Certificate having depth 2 is NOT expired

Depth:1, Cert Info: /CN=iTS CA 1R3/emailAddress=intCA@gmail.com/O=iTS CA 1R3/L=Nashik/C=IN
Certificate having depth 1 is NOT expired

Depth:0, Cert Info: /CN=Alicel.com/emailAddress=Alice1@gmail.com/O=Alice1/L=Nagpur/C=IN
Certificate having depth 0 is NOT expired

.... Certificate Verification Ended ....

Cipher suite selected by server: ECDHE-ECDSA-AES256-GCM-SHA384

.... DTLS 1.2 Handshake ended ....
------------------------------------------------
Server connected to Client using DTLS 1.2
------------------------------------------------
***** Server has started chatting with Client ******

From Client: Hi Bob1
Enter message for Client: Hi Alice1

From Client: The weather seems good today!
Enter message for Client: Yes. Can we go out for picnic?

From Client: Sure.
Enter message for Client: Tomorrow 8 am is fine for you?

From Client: Yes, lets go by car.
Enter message for Client: ok

From Client: See you tomorrow
Enter message for Client: bye

From Client: chat_close
******** Chat closed ********

------------------------------------------------
Server socket created successfully
------------------------------------------------
Server started listening for connection....
```

```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/secure_chat_submission

mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission$ ./secure_chat_app -c mayuresh-HP-Laptop
Client will chat with server:
Server Hostname: mayuresh-HP-Laptop
Server IP: 127.0.1.1
Server Port: 5001
------------------------------------------------
Client Info:
Client IP: 127.0.1.1
Client Port: 6001
------------------------------------------------
Client socket created Successfully
------------------------------------------------
Msg sent to Server: chat_hello
Msg recv from Server: chat_ok_reply
Msg sent to Server: chat_START_SSL
Msg recv from Server: chat_START_SSL_ACK
------------------------------------------------
.... DTLS 1.2 Handshake started ....

.... Certificate Verification Started ....

Depth:2, Cert Info: /CN=iTS ROOT R1/emailAddress=rootCA@gmail.com/O=iTS ROOT R1/L=Hyderabad/C=IN
Certificate having depth 2 is NOT expired

Depth:1, Cert Info: /CN=iTS CA 1R3/emailAddress=intCA@gmail.com/O=iTS CA 1R3/L=Nashik/C=IN
Certificate having depth 1 is NOT expired

Depth:0, Cert Info: /CN=Bob1.com/emailAddress=Bob1@gmail.com/O=Bob1/L=Kolkata/C=IN
Certificate having depth 0 is NOT expired

.... Certificate Verification Ended ....

Cipher suite selected by server: ECDHE-ECDSA-AES256-GCM-SHA384

.... DTLS 1.2 Handshake ended ....
------------------------------------------------
Client connected to Server using DTLS
------------------------------------------------
***** Client has started chatting with Server ******

Enter message for Server: Hi Bob1

From Server: Hi Alice1
Enter message for Server: The weather seems good today!

From Server: Yes. Can we go out for picnic?
Enter message for Server: Sure.

From Server: Tomorrow 8 am is fine for you?
Enter message for Server: Yes, lets go by car.

From Server: ok
Enter message for Server: See you tomorrow

From Server: bye
Enter message for Server: chat_close

******** Chat closed ********

------------------------------------------------
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission$
```

Flow diagram depicting DTLS 1.2 message flow:

Wireshark Trace of entire chat:



a) Running the secure chat application:

We start the server using the below command:

In the screenshot below, Bob is the server running on **Port:5001**.

$ secure_chat_app -s

We start the client using the below command:

In the screenshot below, Alice is the client running on **Port:6001**.

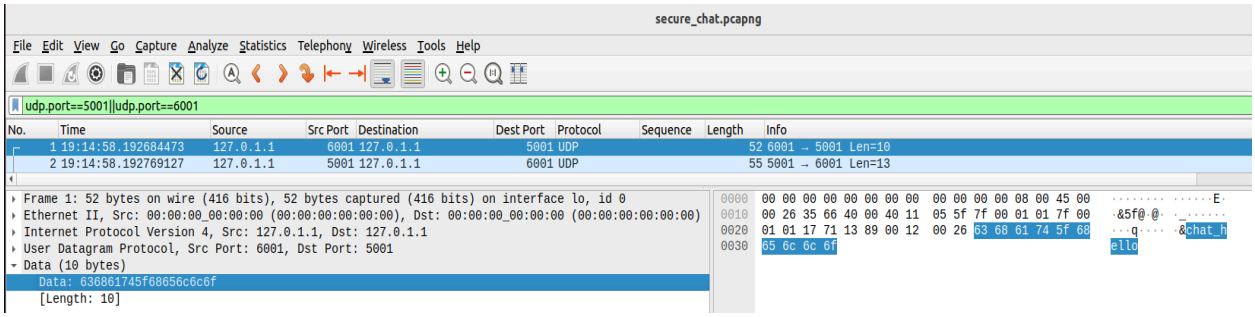
$ secure_chat_app -c <server_hostname>

b) Exchanging the 'hello control messages':

We can observe that these messages are exchanged in plain text.

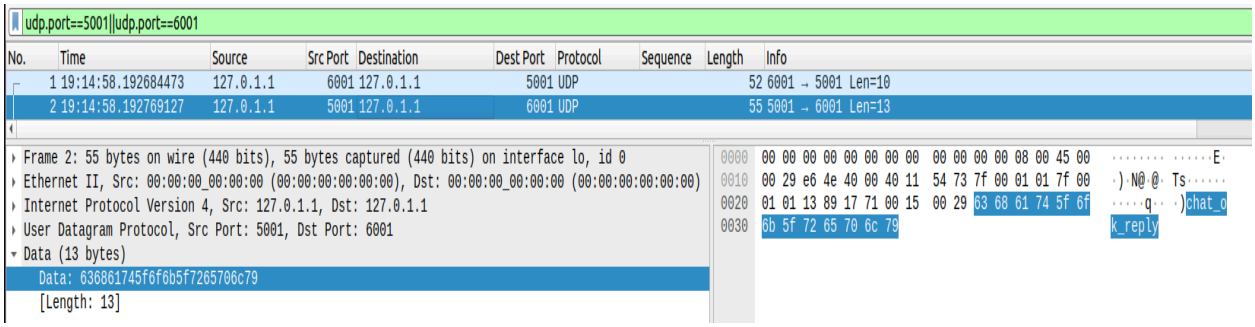
Messages sent by Client to Server:

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission$ ./secure_chat_app -c mayuresh-HP-Laptop
Client will chat with server:
Server Hostname: mayuresh-HP-Laptop
Server IP: 127.0.1.1
Server Port: 5001
-------------------------------------------------
Client Info:
Client IP: 127.0.1.1
Client Port: 6001
-------------------------------------------------
Client socket created Successfully
-------------------------------------------------
Msg sent to Server: chat_hello
Msg recv from Server: chat_ok_reply
```



Messages sent by Server to the Client:

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission$ ./secure_chat_app -s
Server Info:
Server Hostname: mayuresh-HP-Laptop
Server IP: 127.0.1.1
Server Port: 5001
-------------------------------------------------
Server socket created successfully
-------------------------------------------------
Msg recvd from Client: chat_hello
Msg sent to Client: chat_ok_reply
```

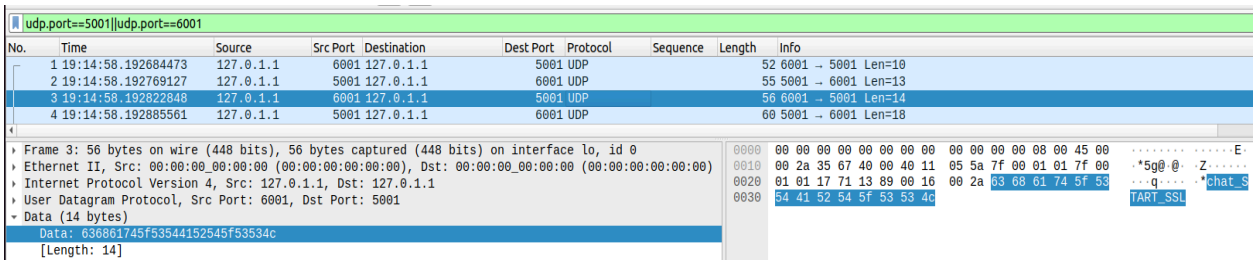


c) Exchanging the ‘control messages’:

We can observe that these messages are exchanged in plain text.

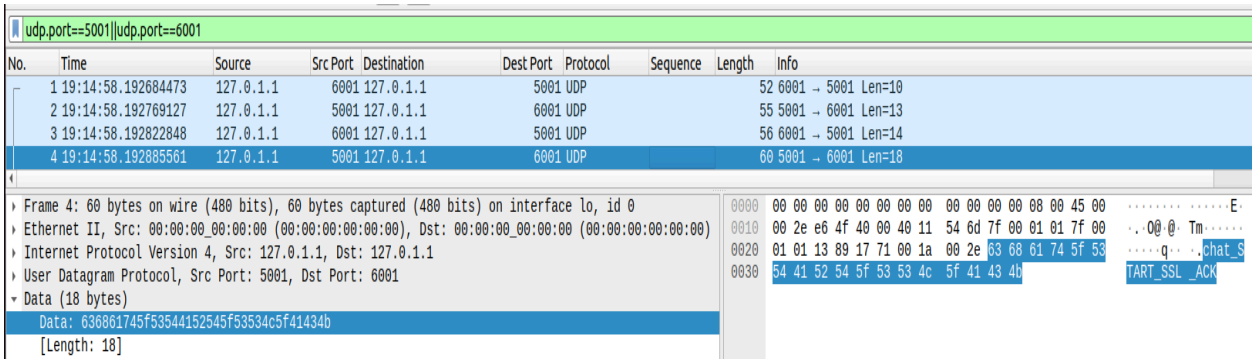
Messages sent by Client to Server:

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission$ ./secure_chat_app -c mayuresh-HP-Laptop
Client will chat with server:
Server Hostname: mayuresh-HP-Laptop
Server IP: 127.0.1.1
Server Port: 5001
-------------------------------------------------
Client Info:
Client IP: 127.0.1.1
Client Port: 6001
-------------------------------------------------
Client socket created Successfully
-------------------------------------------------
Msg sent to Server: chat_hello
Msg recv from Server: chat_ok_reply
Msg sent to Server: chat_START_SSL
Msg recv from Server: chat_START_SSL_ACK
-------------------------------------------------
```



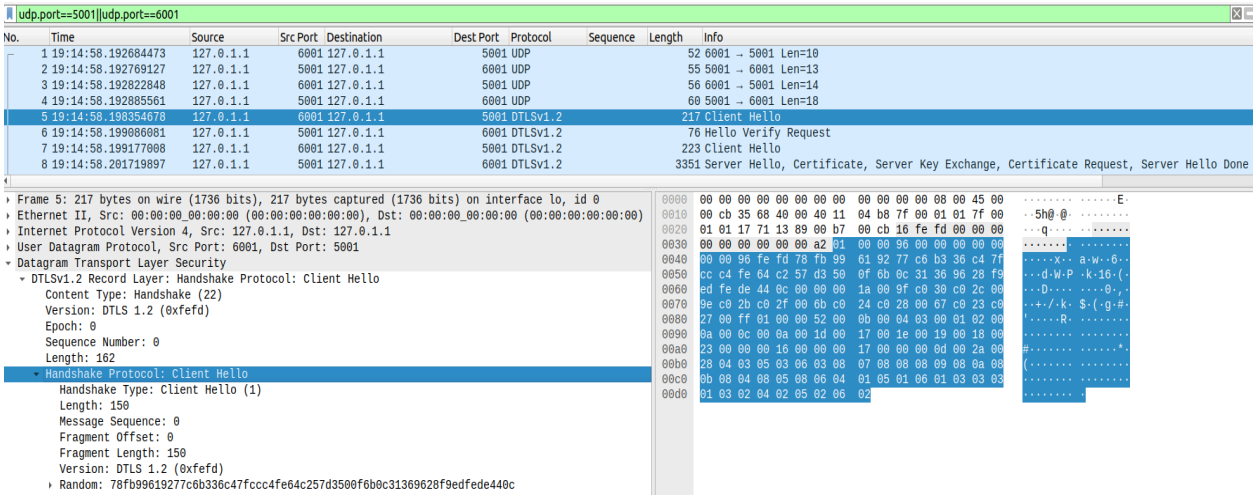
Messages sent by Server to the Client:

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/secure_chat_submission$ ./secure_chat_app -s
Server Info:
Server Hostname: mayuresh-HP-Laptop
Server IP: 127.0.1.1
Server Port: 5001
-------------------------------------------------
Server socket created successfully
-------------------------------------------------
Msg recvd from Client: chat_hello
Msg sent to Client: chat_ok_reply
Msg recv from Client: chat_START_SSL
Msg sent to Client: chat_START_SSL_ACK
-------------------------------------------------
```



c-i) Client sends ‘Client Hello’ message to server:

When the client receives ‘chat_START_SSL_ACK’ from the server, it initiates the DTLS 1.2 handshake by sending a ‘Client Hello’ message to Server.

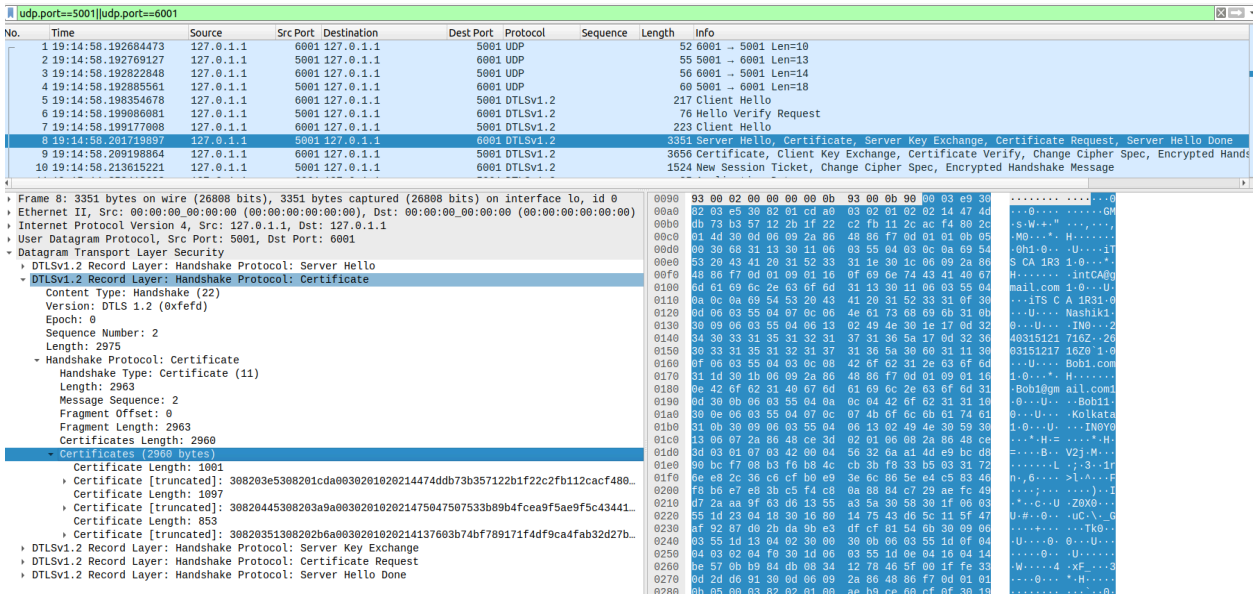


c-ii) Certificate Exchange and Verification:

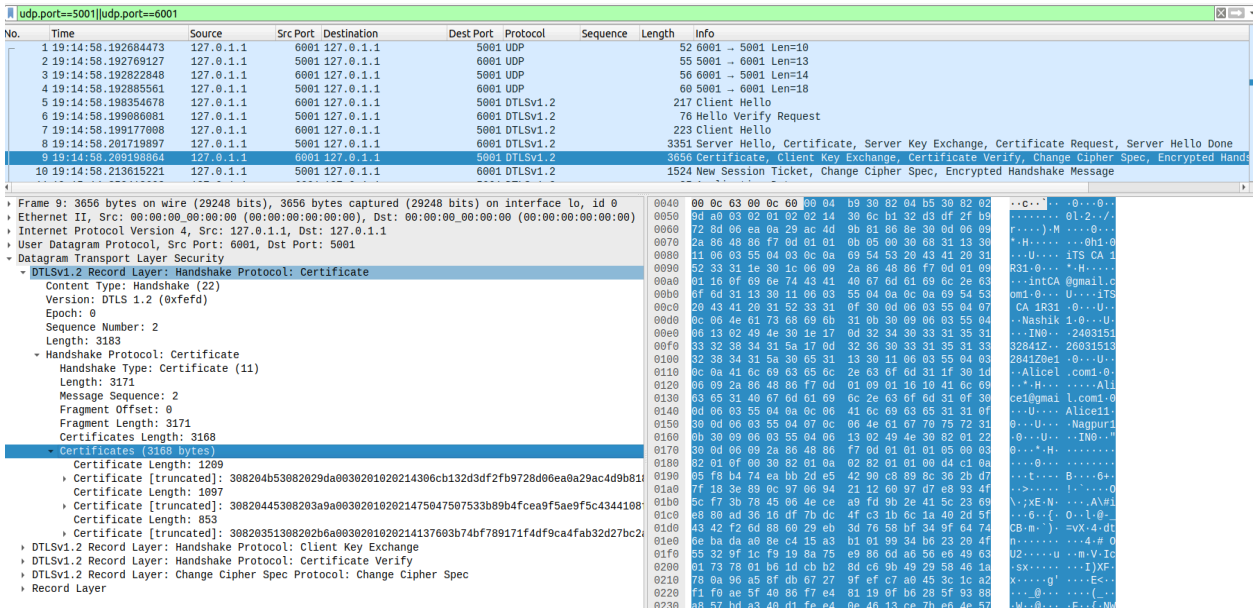
Certificate Exchange:

- Server replies to Client by a ‘Server Hello’ message which contains Server’s Digital Certificate.
- This certificate contains the server's public key.
- We can observe that, in the ‘Server Hello’ message, the server also requests for the Client's digital certificate.

Server sends its certificate to Client:



Client sends its certificate to Server: Client provides its certificate to Server in subsequent message.



Certificate Verification:

The C code used for 'Certificate Verification' is as below:

```c
// Function to perform certificate expiry date checking
int verify_certificate_expiry_date(X509 *cert)
{
    ASN1_TIME *not_after = X509_get_notAfter(cert);
    time_t now = time(NULL);
    struct tm tm_val;
    ASN1_TIME_to_tm(not_after, &tm_val);
    time_t expiry_time = mktime(&tm_val);
    return (now > expiry_time) ? 0 : 1;
}

// This function is called for each certificate in the 'Trust Chain'
int verify_certificate_callback(int preverify_ok, X509_STORE_CTX *ctx)
{
    char buf[256];
    X509 *err_cert;
    int err, depth;

    // Extracting the certificate from SSL/DTLS context
```

```
    err_cert = X509_STORE_CTX_get_current_cert(ctx);
    err = X509_STORE_CTX_get_error(ctx);
    depth = X509_STORE_CTX_get_error_depth(ctx);

    // Printing the depth and Subject name of certificate
    X509_NAME_oneline(X509_get_subject_name(err_cert), buf, 256);
    printf("Depth:%d, Cert Info: %s\n", depth, buf);
    if (!preverify_ok) {
        printf("Certificate Verification Error: num=%d:%s:%s\n", err,
X509_verify_cert_error_string(err), buf);
        exit(1);
    }
    if (preverify_ok) {
        // Verifying the expiry date of certificate
        if (verify_certificate_expiry_date(err_cert)) {
            printf("Certificate having depth %d is NOT expired\n\n",
depth);
        } else {
            printf("Error: Certificate is Expired\n");
            exit(1);
        }
    }
    return preverify_ok;
}
```

1. verify_certificate_callback() function is called to verify each certificate in the 'Trust Chain'.
2. It extracts the digital certificate from the SSL/TLS context object.
3. It prints the certificate subject name and its depth in the trust chain.
4. It verifies the expiry date of the certificate.
5. If the certificate is expired then it throws an error message and stops the DTLS 1.2 handshake.

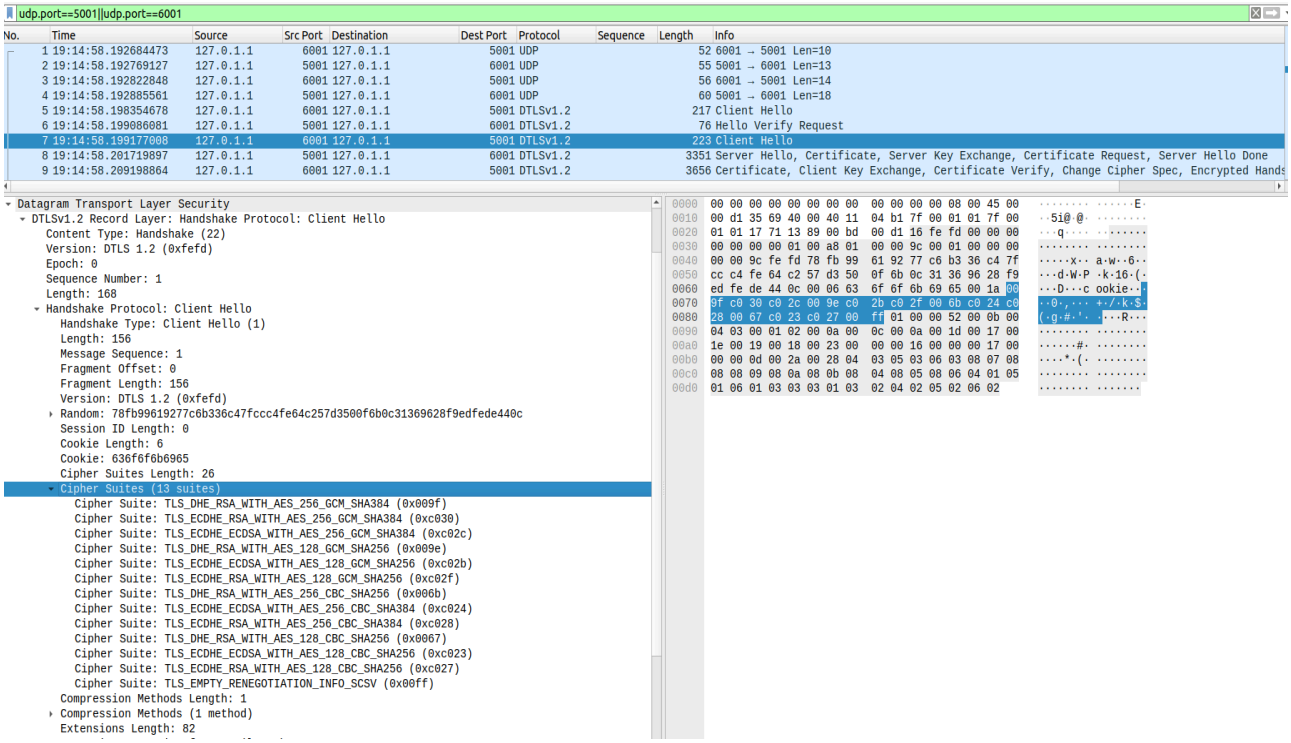
c-iii) Cipher Suite Negotiation:

- Client advertises a list of PFS (Perfect Forward Secrecy) supported cipher suites to Server through the 'Client Hello' message.

- Server selects one cipher from this list and conveys it to the Client using 'Server Hello' message.

- We can observe the same thing in the screenshot below.

Client sends list of PFS supported cipher suites to Server using ‘Client Hello’ message:

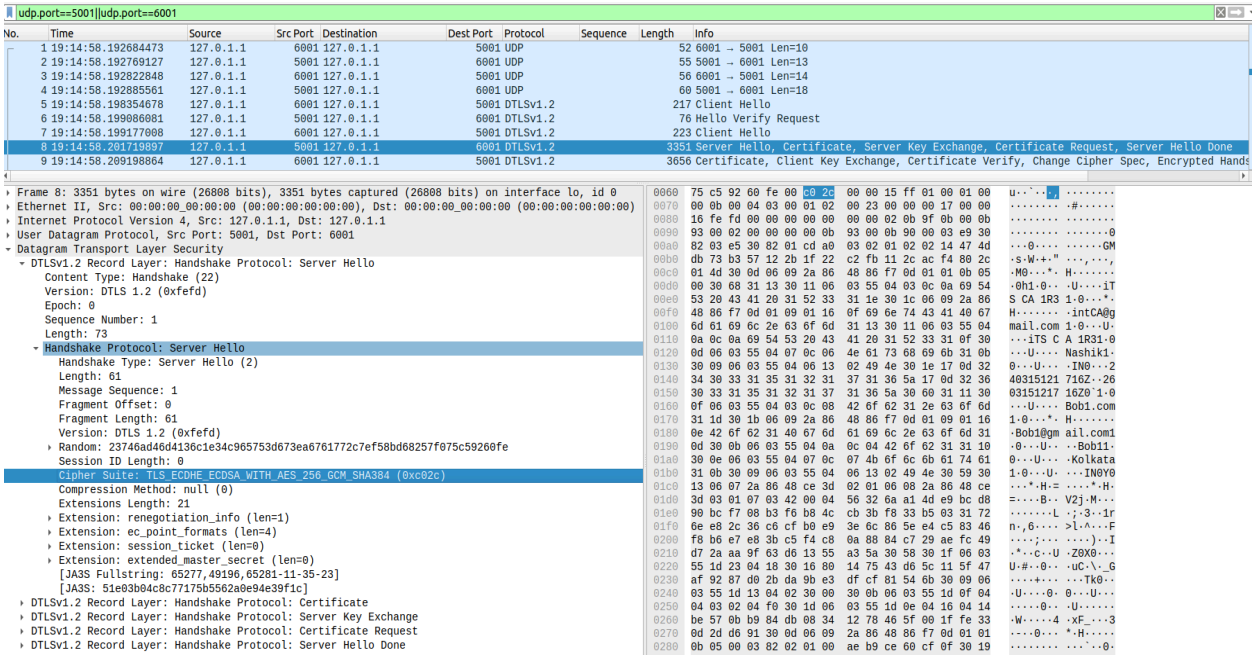


Part of the code used for sending the PFS supported Cipher suites list to Server through ‘Client Hello’:

```
SSL_CTX *create_client_context() // alice
{
   // Creating context and filling certs and private key in it
   SSL_CTX *ctx = SSL_CTX_new(DTLSv1_2_client_method());

   // Setting the cipher suits list of "12 Cipher Suites that support PFS"
   SSL_CTX_set_cipher_list(ctx,
"DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-
GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-R
SA-AES128-GCM-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE
-RSA-AES256-SHA384:DHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-R
SA-AES128-SHA256");
```

This is Client side code, which sets the list of PFS supported cipher suites in the DTLS 1.2 context of client.

Server coveys the cipher suite to Client through ‘Server Hello’ message:



Part of the code used for conveying the selected cipher suite to Client through ‘Server Hello’:

```
SSL_CTX *create_server_context() // bob
{
   // Creating context and filling certs and private key in it
   SSL_CTX *ctx = SSL_CTX_new(DTLSv1_2_server_method());

   // Setting the cipher suits list
   SSL_CTX_set_cipher_list(ctx, "ECDHE-ECDSA-AES256-GCM-SHA384");
```

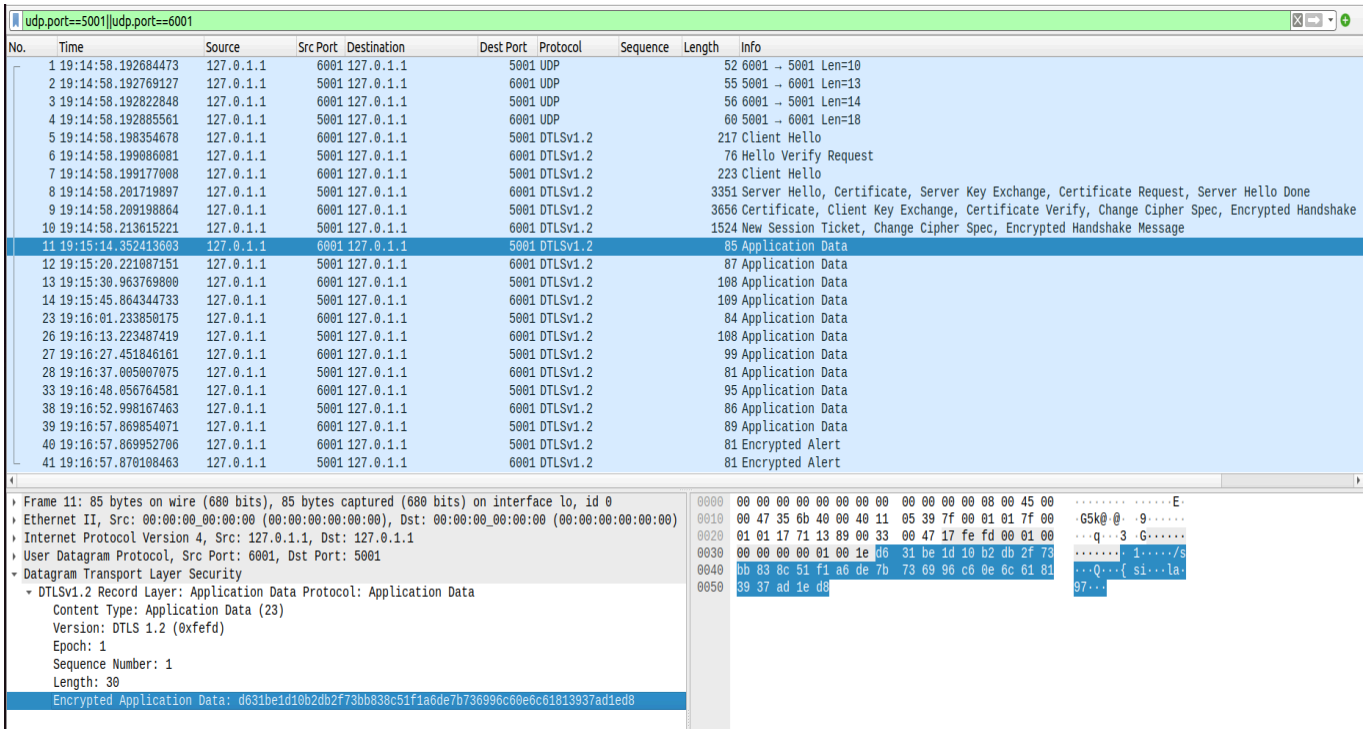
This is Server side code, which sets the selected cipher suites in the DTLS 1.2 context of server.

d) Encrypted chatting using DTLS 1.2:

- We can observe in the below screenshot, chat messages between Client and Server are exchanged in the encrypted form.

- As can be seen from the protocol column in wireshark, the client and server uses DTLS version 1.2 protocol for chatting

- The ‘Application data’ packets, contains the encrypted chat messages.



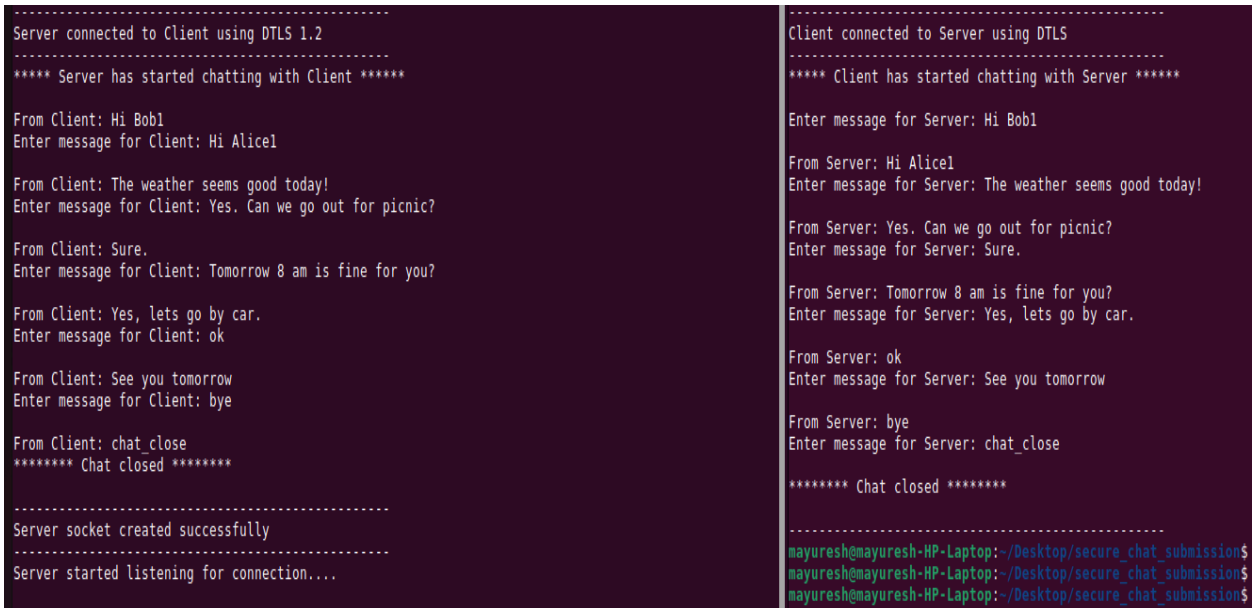
e) Difference between DTLS 1.2 handshake and TLS 1.2 handshake:

Comparison	DTLS 1.2	TLS 1.2
1) Client Hello	In DTLS 1.2 two “ClientHello” messages are exchanged: ● For the first “ClientHello” message, the server’s response includes a “HelloVerifyRequest” message and a cookie. ● Client then sends the second “ClientHello” message along with the received cookie, which then the server verifies and responds with the “ServerHello” message	In this case, there is only one “ClientHello” message sent by client to the server, which in turn responds with the “ServerHello” message (containing the Server Certificate, ServerKeyExchange, CertificateRequest).
2) Connection Type	It uses UDP as the underlying transmission protocol.	It uses TCP as the underlying transmission protocol.
3) Protection Against IP Spoofing	Uses cookies to prevent IP spoofing	Performs communications only after establishing the TCP handshake. This makes it difficult to spoof.
4) Reliability and ordered delivery	Does not guarantee reliable or ordered delivery of packets	Provides reliable, ordered, and error-checked delivery of packets
5) Fragmentation and Retransmission	It handles fragmentation and retransmission at the application layer because UDP lacks built-in support for these features. DTLS include mechanisms such as message retransmission, ack and timeout handling to ensure handshake reliability.	It relies on TCP for fragmentation and retransmission of lost packets, or packets arriving out of order.
6) Latency	Reduces latency and overhead. Hence it is more suitable for real time applications such as VoIP, etc.	Used when data integrity and ordering is more important than reducing latency.

g) Chat closing scenarios:

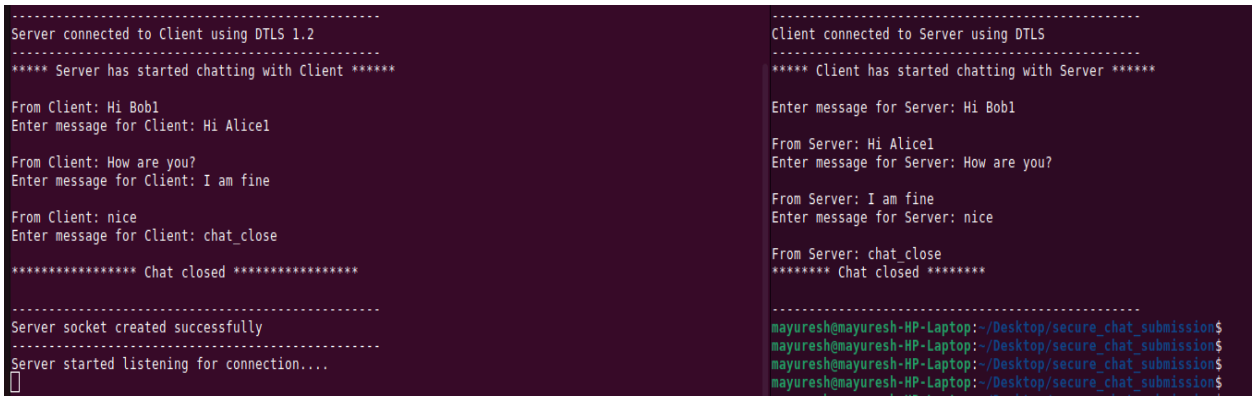
Scenario 1: When client closes the chat

- Client can close the chatting by sending the ‘chat_close’ message to Server.
- When the Client closes the chat, the Server keeps on listening for incoming connection requests from newer Clients.



Scenario 2: When server closes the chat

- Server can also close the chatting by sending the ‘chat_close’ message to Client.
- When the Server closes the chat with one client, it keeps on listening for incoming connection requests from newer Clients.



Packet Loss:

1. **Command to see all connected interfaces, used below command:**
   $ ifconfig

```
root@alice1:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.31.0.2  netmask 255.255.255.0  broadcast 172.31.0.255
        inet6 fe80::216:3eff:feae:c3fd  prefixlen 64  scopeid 0x20<link>
        ether 00:16:3e:ae:c3:fd  txqueuelen 1000  (Ethernet)
        RX packets 114587  bytes 537006277 (537.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 82479  bytes 7202690 (7.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1599  bytes 142528 (142.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1599  bytes 142528 (142.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@alice1:~#
```

2. **Command used to View the Configured Network Conditions:**
   $ sudo tc qdisc show dev <interface_name>

```
root@alice1:~# sudo tc qdisc show dev eth0
qdisc noqueue 0: root refcnt 2
```

3. **Command used to inject the packet loss:**
   $ sudo tc qdisc add dev <interface_name> root netem loss 33%

```
root@alice1:~# sudo tc qdisc add dev eth0 root netem loss 55%
root@alice1:~# sudo tc qdisc show dev eth0
qdisc netem 8026: root refcnt 3 limit 1000 loss 55%
root@alice1:~#
```

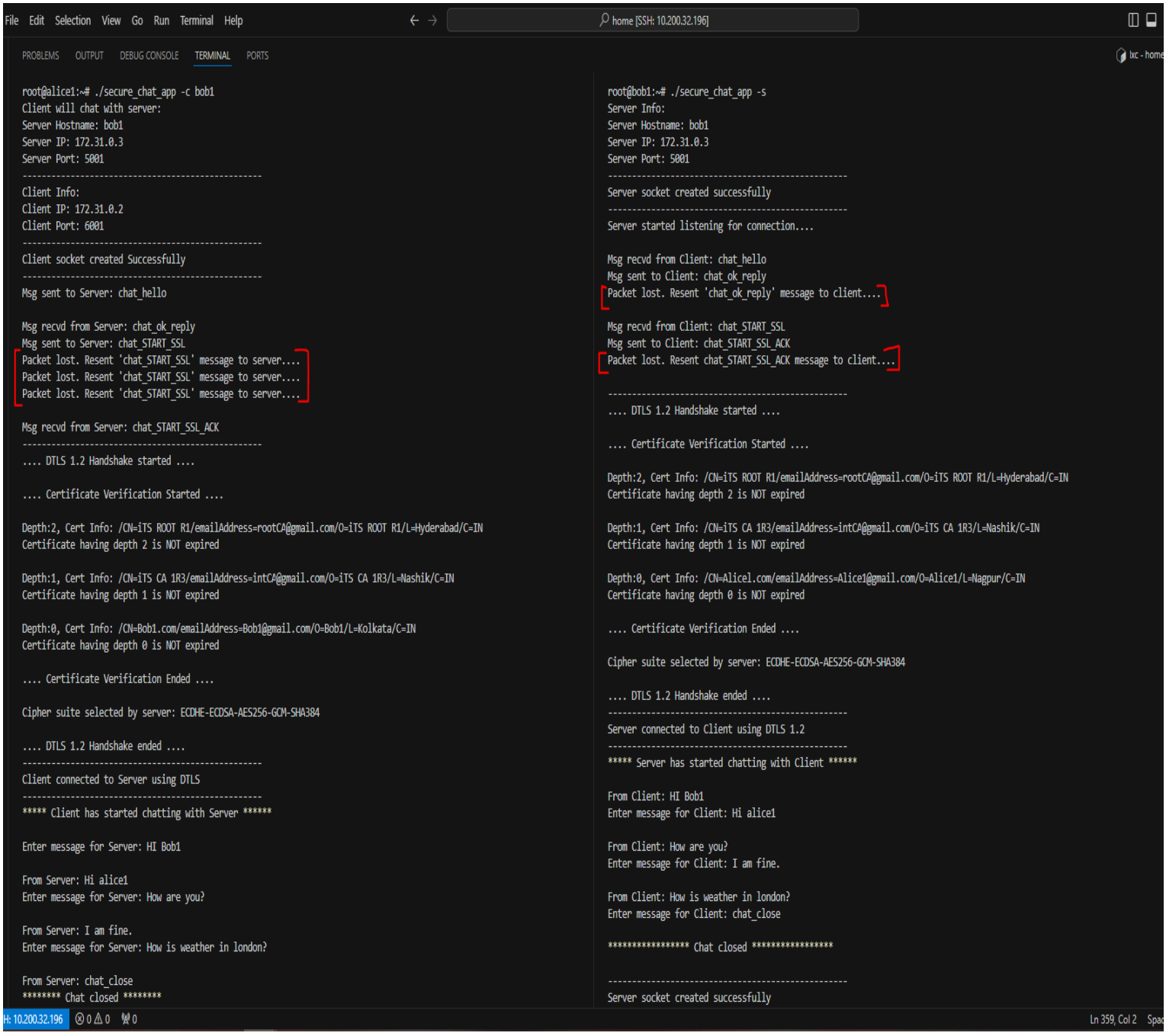
```
root@bob1:~# sudo tc qdisc add dev eth0 root netem loss 55%
root@bob1:~# sudo tc qdisc show dev eth0
qdisc netem 8027: root refcnt 3 limit 1000 loss 55%
root@bob1:~#
```

4. **Command used to delete the packet loss configuration:**
   $ sudo tc qdisc del dev <interface_name> root netem

```
root@bob1:~# sudo tc qdisc del dev eth0 root netem
root@bob1:~#
```

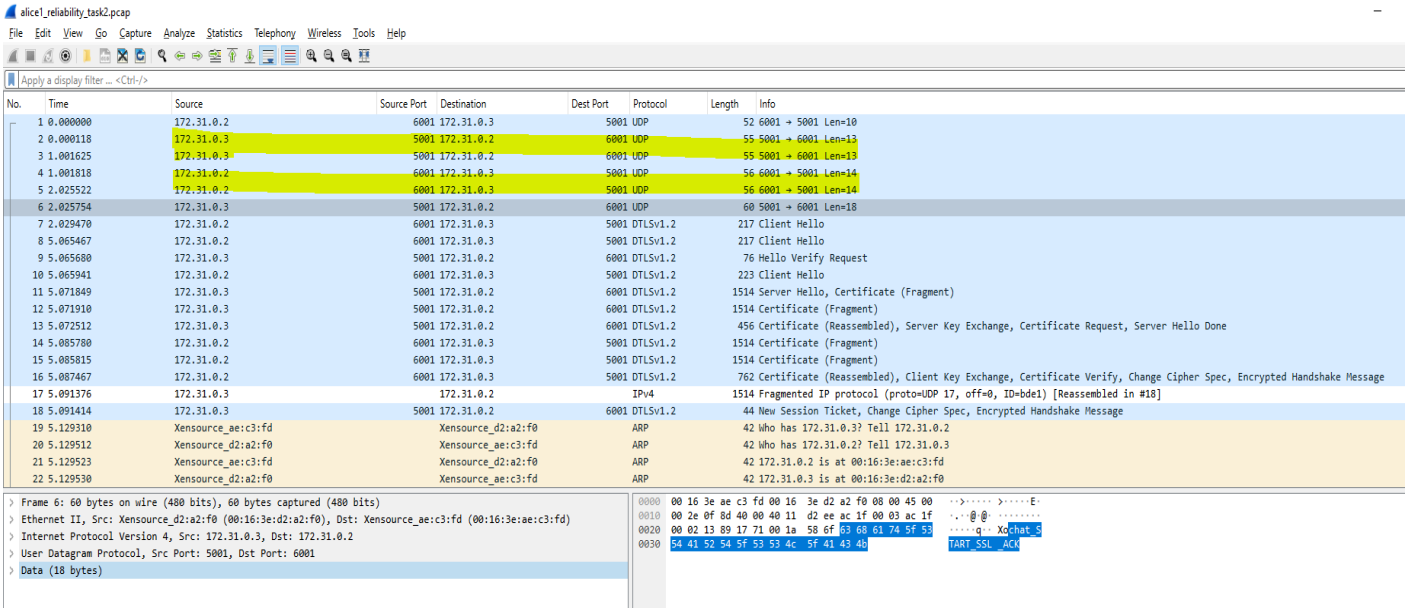
Secure chat after injecting packet loss at alice1 and bob1:

- We can observe that, after injecting 55% packet loss in the application layer, some paintext control messages get lost, and are **retransmitted to achieve reliability.**
- Retransmitted control messages are highlighted using red color.

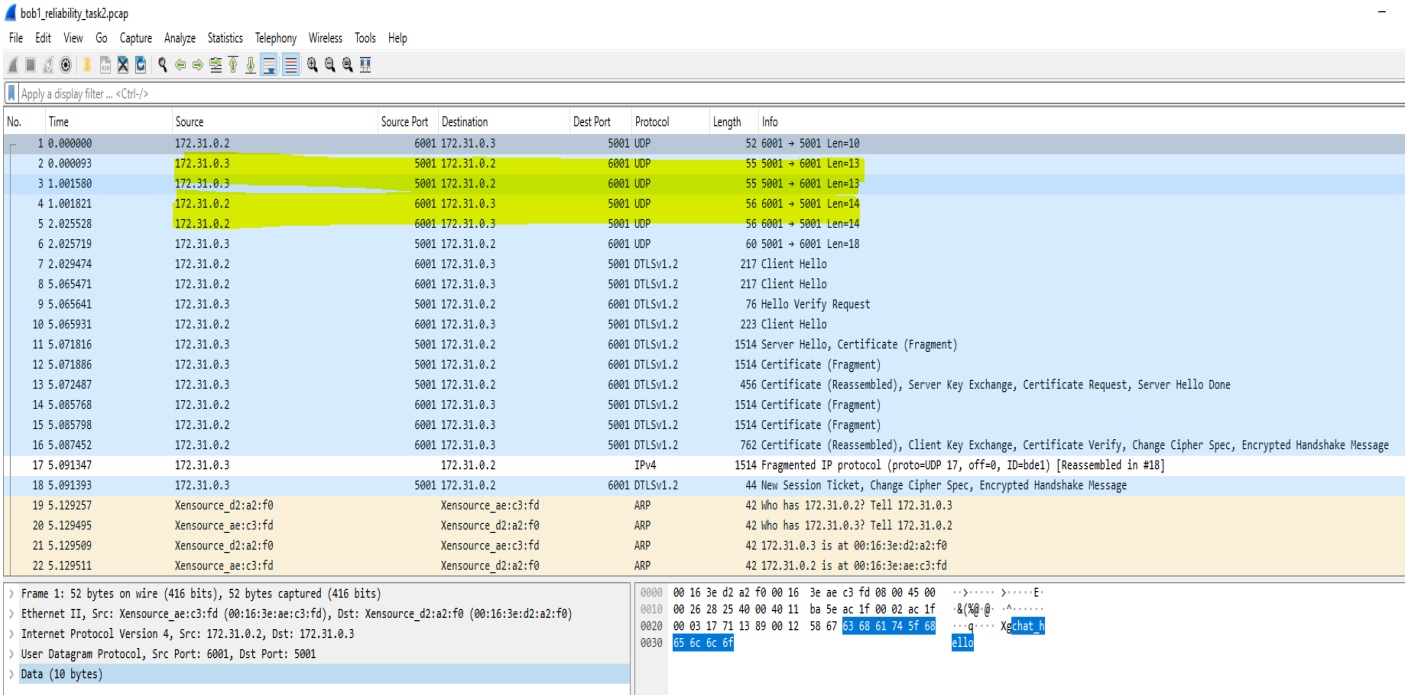


Wireshark trace after injecting packet loss:

Tcpdump trace at alice1:



Tcpdump trace at bob1:



From the above pcap trace at alice1 and bob1, we can see that plain text control messages are being retransmitted due to packet loss.

Challenges faced during the implementation of this secure_chat_app:

The private key generated for Alice was a 1024-bit RSA key. However, when we were trying to load this private key during the code execution we encountered an error “ee key too small”. Therefore for finally running the code we have used a 2048-bit RSA key. This has been kept in the “alice_2048” directory.

TASK 3: START SSL downgrade attack for eavesdropping

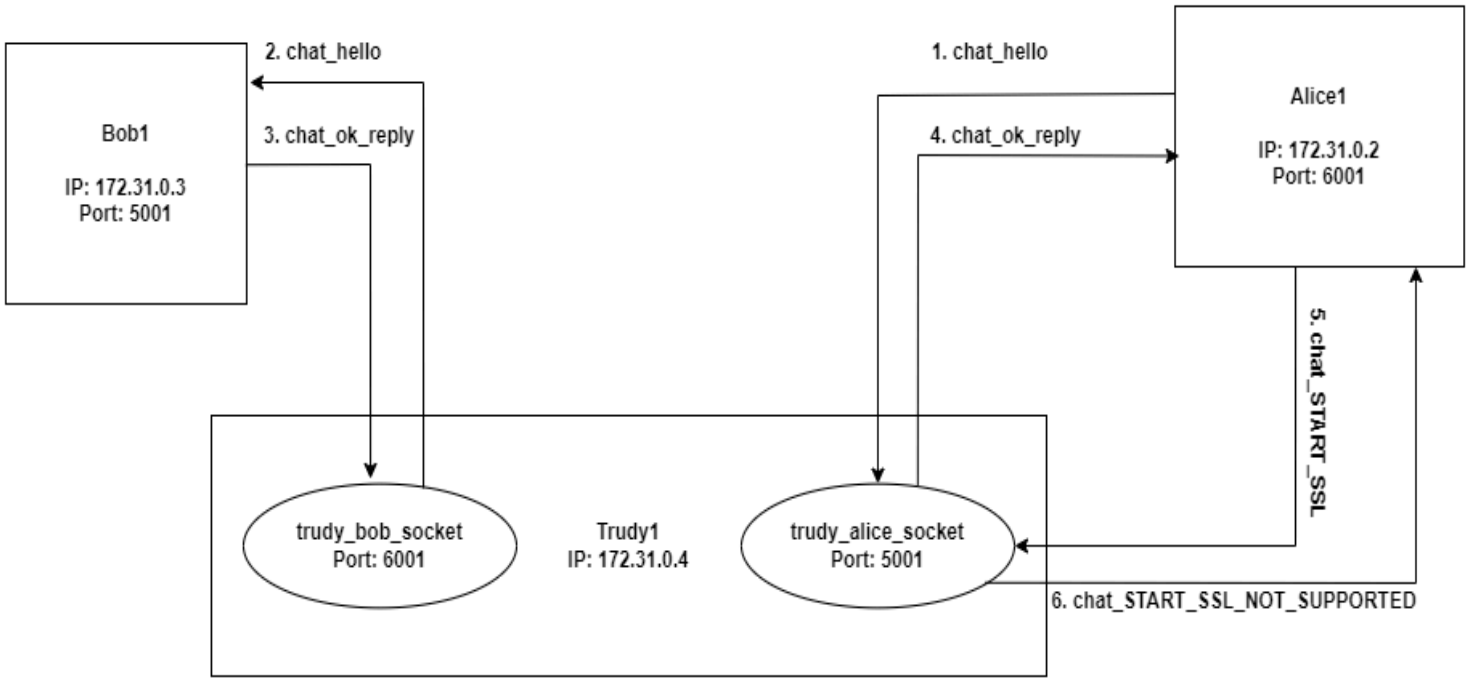


Fig: Control Message flow diagram of downgrade attack

- When the downgrade attack is performed by trudy1, the entire chatting happens in plain text messages through **UDP sockets (SOCK_DGRAM)**.

- Here, trudy1 acts as a passive attacker, i.e., trudy1 only eavesdrops on the chat messages sent between alice1 and bob1, but does not modify their content.

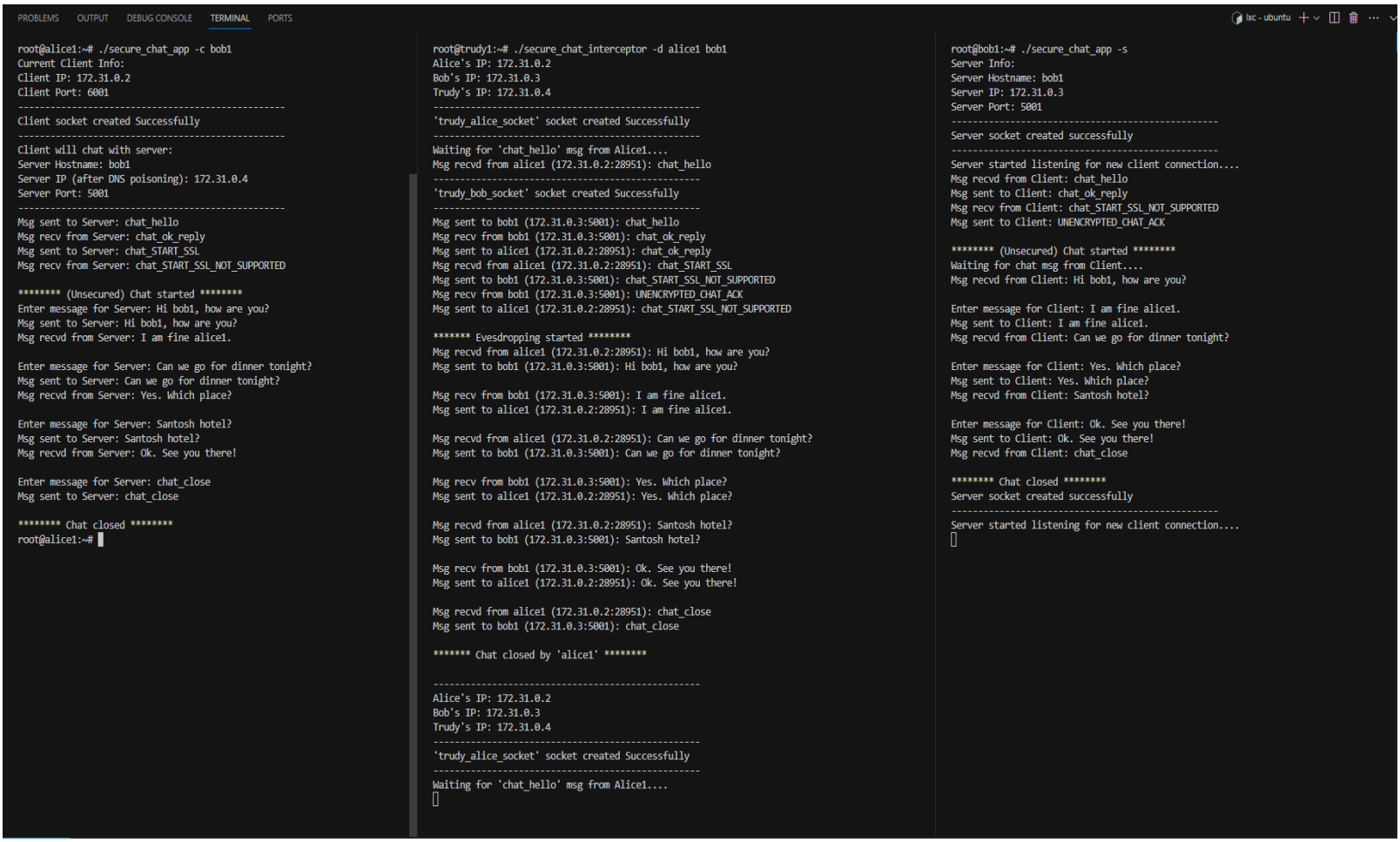
- To act as a passive attacker, it poisons the /etc/hosts file by replacing the IP addresses of alice1 with bob1 and vice versa.

- When alice1 sends a ‘chat_START_SSL’ message to bob1, it gets intercepted by trudy1.

- Now trudy1, sends ‘chat_START_SSL_NOT_SUPPORTED’ message to alice1, hence performing the ‘Downgrade attack’.

- From now the entire communication between alice1 and bob1 happens in plain text. Hence, trudy1 can easily eavesdrop on chat messages.

Downgrade attack on secure chat output:



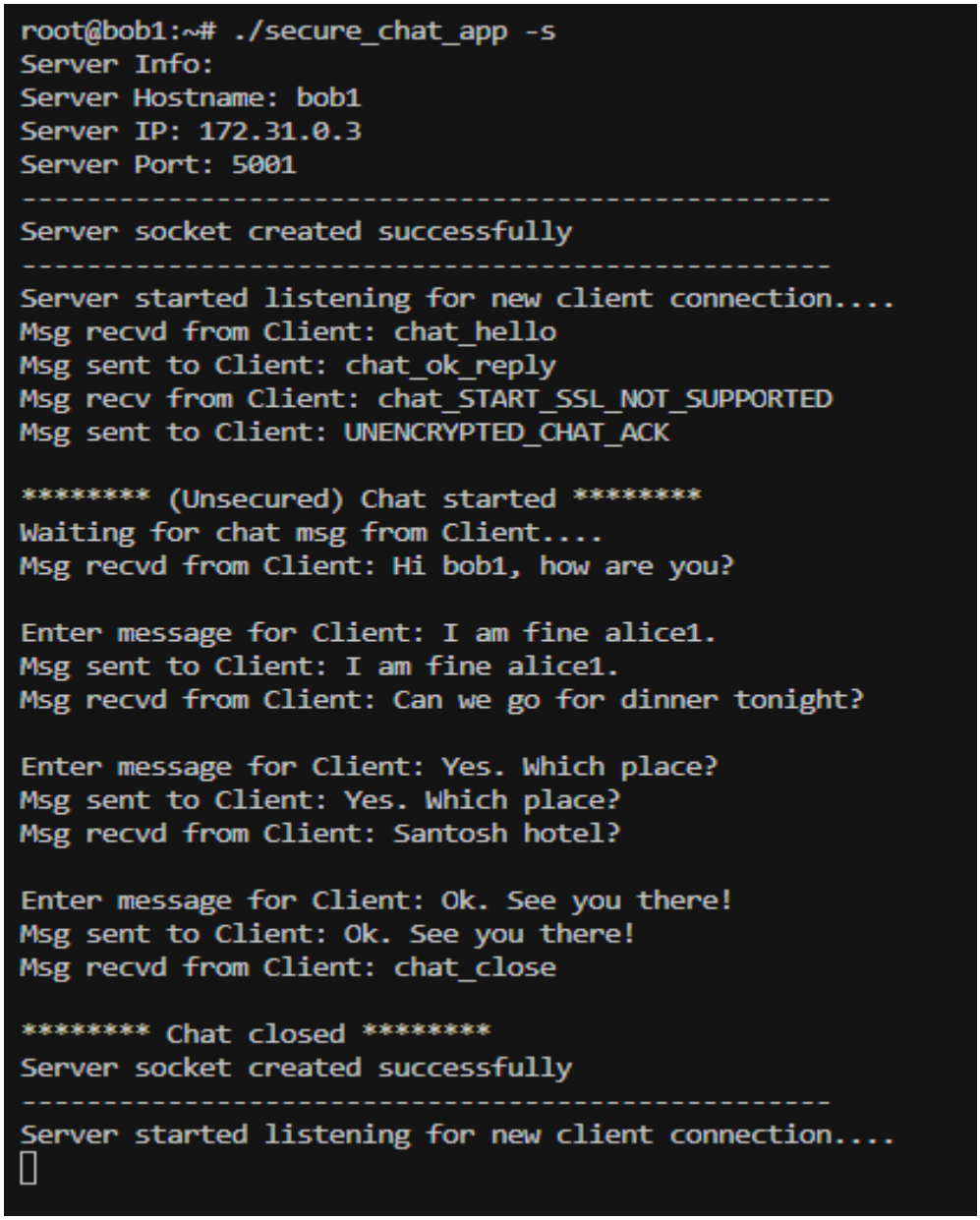
- When the chat is closed by anyone (client or server), then the server starts listening for new client connection requests.

- When the chat is closed, trudy1 keeps on running to eavesdrop any upcoming client server communication that will occur in future.

Downgrade attack on secure chat ‘alice1’ output:

```
root@alice1:~# ./secure_chat_app -c bob1
Current Client Info:
Client IP: 172.31.0.2
Client Port: 6001
------------------------------------------------
Client socket created Successfully
------------------------------------------------
Client will chat with server:
Server Hostname: bob1
Server IP (after DNS poisoning): 172.31.0.4
Server Port: 5001
------------------------------------------------
Msg sent to Server: chat_hello
Msg recv from Server: chat_ok_reply
Msg sent to Server: chat_START_SSL
Msg recv from Server: chat_START_SSL_NOT_SUPPORTED

******** (Unsecured) Chat started ********
Enter message for Server: Hi bob1, how are you?
Msg sent to Server: Hi bob1, how are you?
Msg recvd from Server: I am fine alice1.

Enter message for Server: Can we go for dinner tonight?
Msg sent to Server: Can we go for dinner tonight?
Msg recvd from Server: Yes. Which place?

Enter message for Server: Santosh hotel?
Msg sent to Server: Santosh hotel?
Msg recvd from Server: Ok. See you there!

Enter message for Server: chat_close
Msg sent to Server: chat_close

******** Chat closed ********
root@alice1:~#
```

Downgrade attack on secure chat ‘bob1’ output:

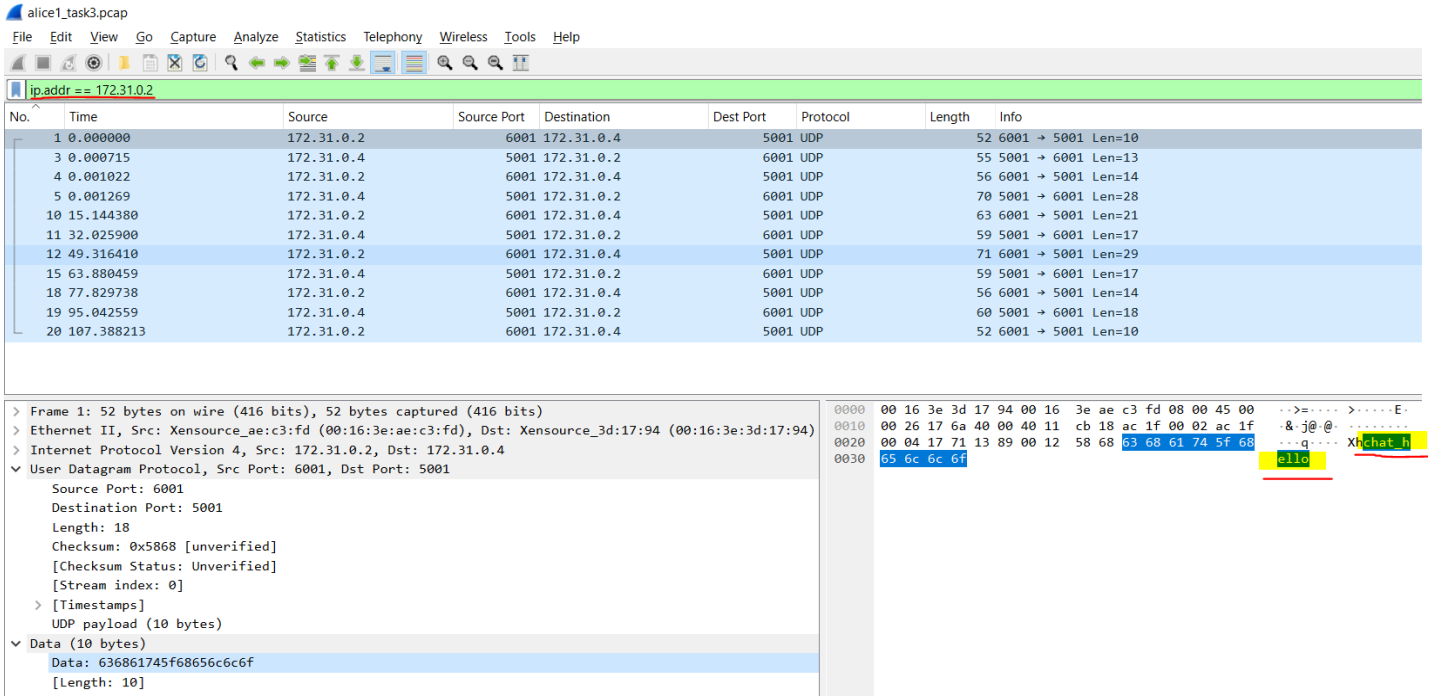


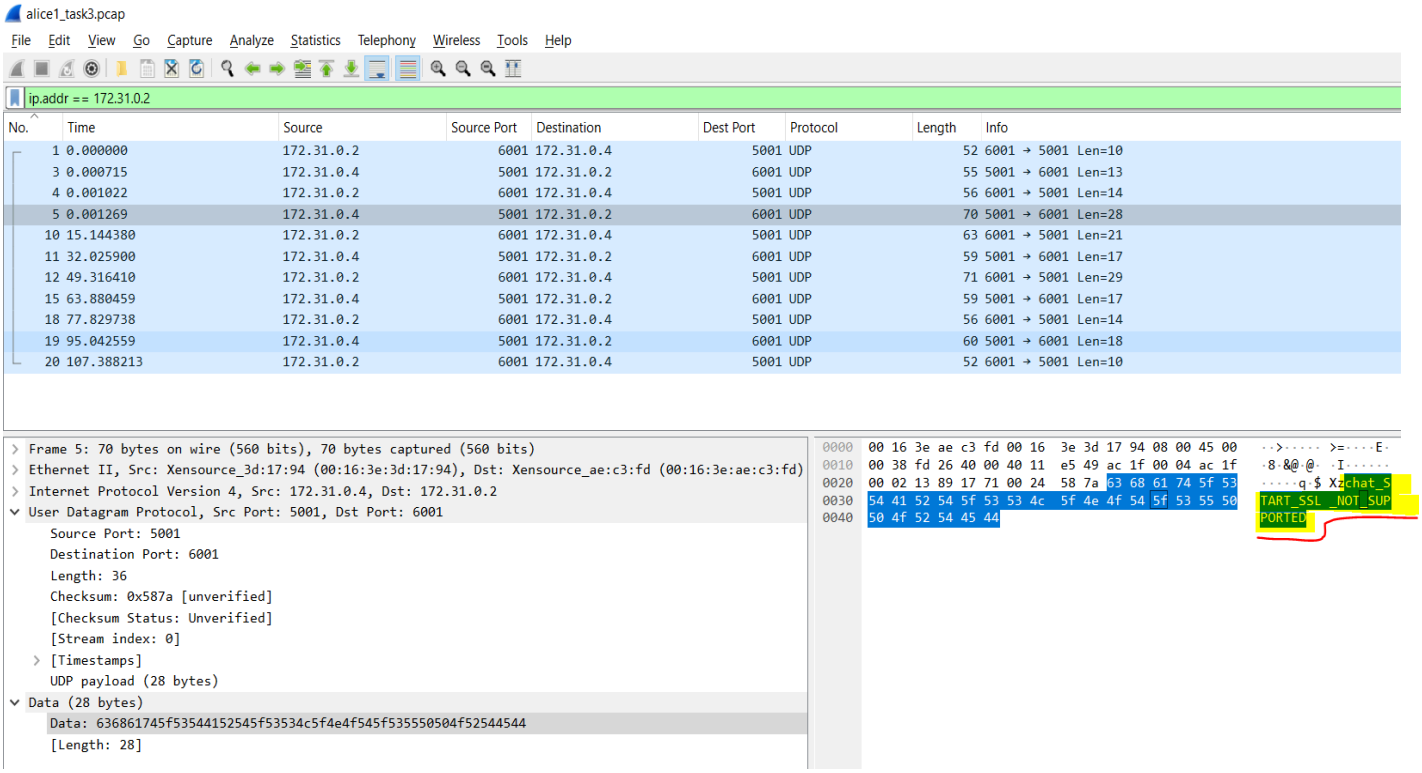
```
root@bob1:~# ./secure_chat_app -s
Server Info:
Server Hostname: bob1
Server IP: 172.31.0.3
Server Port: 5001
------------------------------------------------
Server socket created successfully
------------------------------------------------
Server started listening for new client connection....
Msg recvd from Client: chat_hello
Msg sent to Client: chat_ok_reply
Msg recv from Client: chat_START_SSL_NOT_SUPPORTED
Msg sent to Client: UNENCRYPTED_CHAT_ACK

******** (Unsecured) Chat started ********
Waiting for chat msg from Client....
Msg recvd from Client: Hi bob1, how are you?

Enter message for Client: I am fine alice1.
Msg sent to Client: I am fine alice1.
Msg recvd from Client: Can we go for dinner tonight?

Enter message for Client: Yes. Which place?
Msg sent to Client: Yes. Which place?
Msg recvd from Client: Santosh hotel?

Enter message for Client: Ok. See you there!
Msg sent to Client: Ok. See you there!
Msg recvd from Client: chat_close

******** Chat closed ********
Server socket created successfully
------------------------------------------------
Server started listening for new client connection....
```

Downgrade attack on secure chat ‘trudy1’ output:

```
root@trudy1:~# ./secure_chat_interceptor -d alice1 bob1
Alice's IP: 172.31.0.2
Bob's IP: 172.31.0.3
Trudy's IP: 172.31.0.4
------------------------------------------------
'trudy_alice_socket' socket created Successfully
------------------------------------------------
Waiting for 'chat_hello' msg from Alice1....
Msg recvd from alice1 (172.31.0.2:28951): chat_hello
------------------------------------------------
'trudy_bob_socket' socket created Successfully
------------------------------------------------
Msg sent to bob1 (172.31.0.3:5001): chat_hello
Msg recv from bob1 (172.31.0.3:5001): chat_ok_reply
Msg sent to alice1 (172.31.0.2:28951): chat_ok_reply
Msg recvd from alice1 (172.31.0.2:28951): chat_START_SSL
Msg sent to bob1 (172.31.0.3:5001): chat_START_SSL_NOT_SUPPORTED
Msg recv from bob1 (172.31.0.3:5001): UNENCRYPTED_CHAT_ACK
Msg sent to alice1 (172.31.0.2:28951): chat_START_SSL_NOT_SUPPORTED

******* Evesdropping started ********
Msg recvd from alice1 (172.31.0.2:28951): Hi bob1, how are you?
Msg sent to bob1 (172.31.0.3:5001): Hi bob1, how are you?

Msg recv from bob1 (172.31.0.3:5001): I am fine alice1.
Msg sent to alice1 (172.31.0.2:28951): I am fine alice1.

Msg recvd from alice1 (172.31.0.2:28951): Can we go for dinner tonight?
Msg sent to bob1 (172.31.0.3:5001): Can we go for dinner tonight?

Msg recv from bob1 (172.31.0.3:5001): Yes. Which place?
Msg sent to alice1 (172.31.0.2:28951): Yes. Which place?

Msg recvd from alice1 (172.31.0.2:28951): Santosh hotel?
Msg sent to bob1 (172.31.0.3:5001): Santosh hotel?

Msg recv from bob1 (172.31.0.3:5001): Ok. See you there!
Msg sent to alice1 (172.31.0.2:28951): Ok. See you there!

Msg recvd from alice1 (172.31.0.2:28951): chat_close
Msg sent to bob1 (172.31.0.3:5001): chat_close

******* Chat closed by 'alice1' ********

------------------------------------------------
Alice's IP: 172.31.0.2
Bob's IP: 172.31.0.3
Trudy's IP: 172.31.0.4
------------------------------------------------
'trudy_alice_socket' socket created Successfully
------------------------------------------------
Waiting for 'chat_hello' msg from Alice1....
```

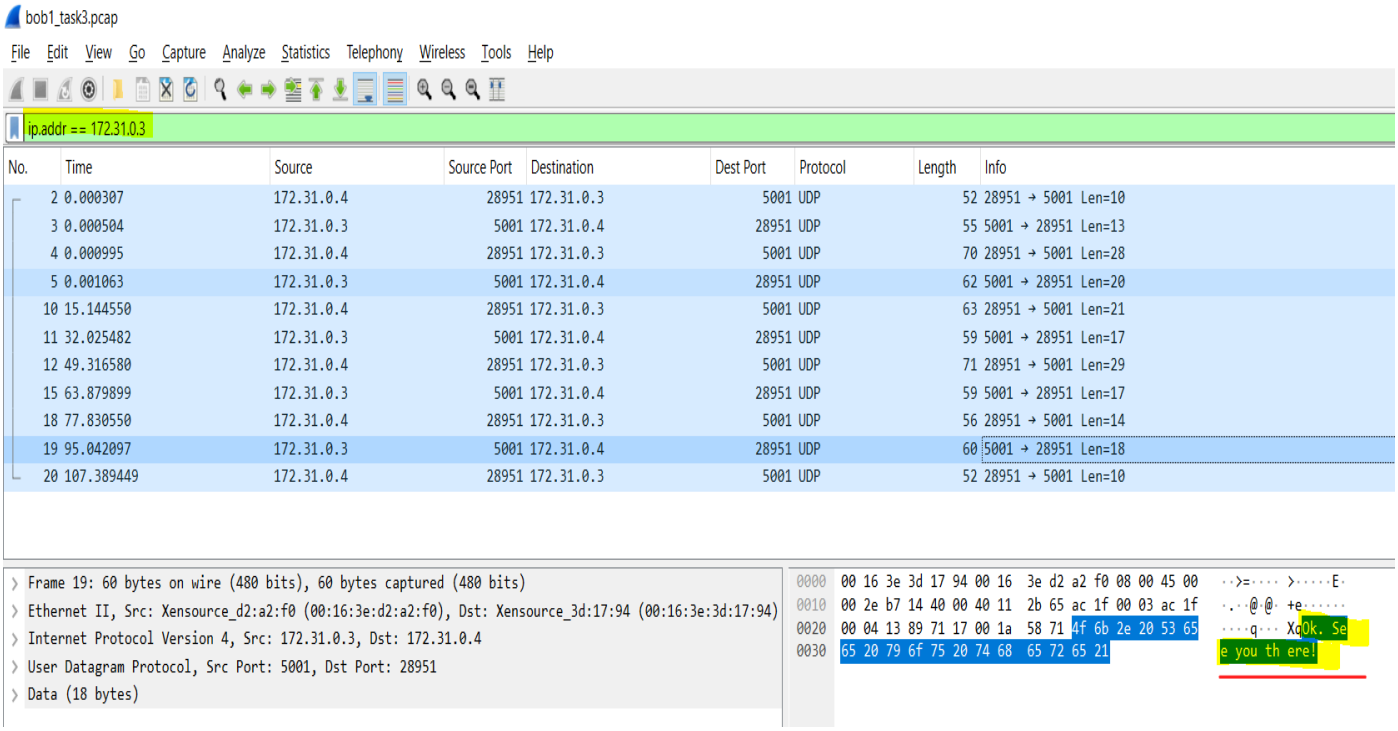
1. Tcpdump trace captured at ‘alice1’:





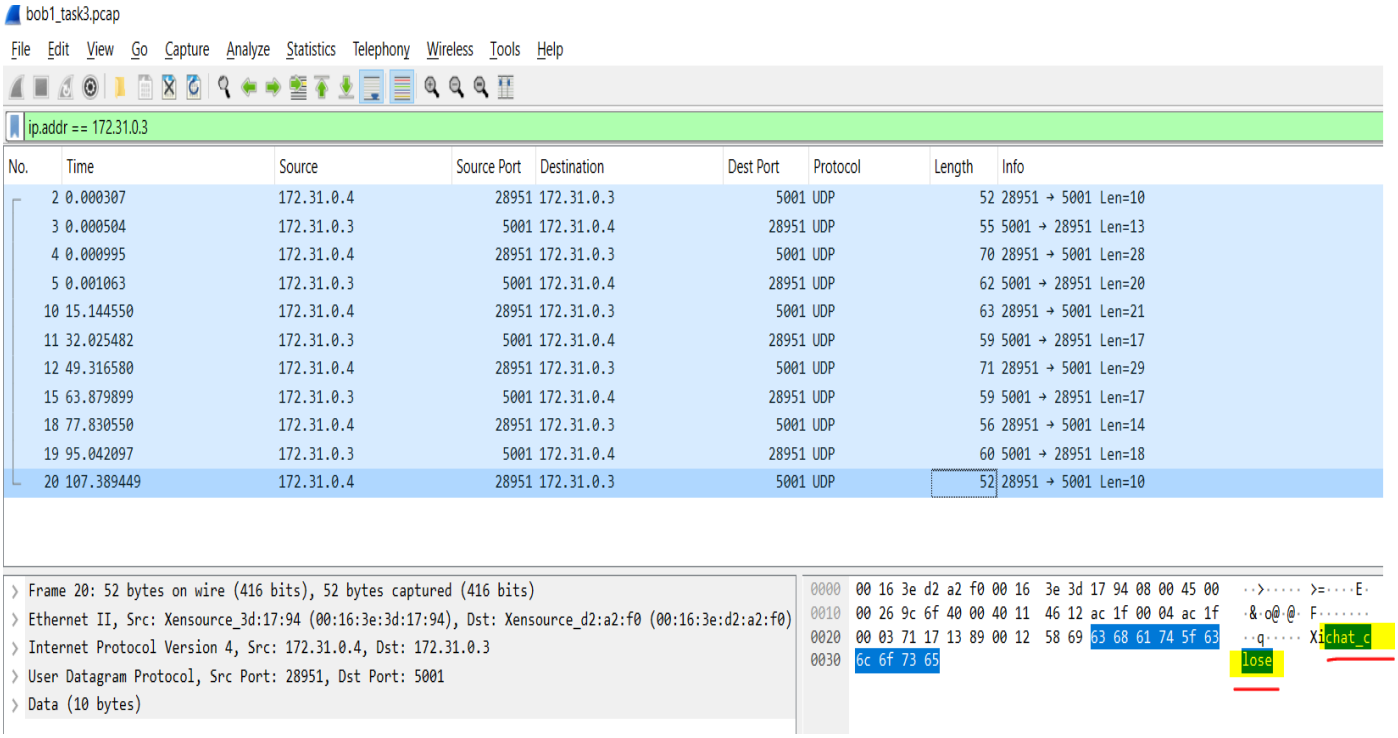
In this pcap, we can observe the ‘chat_START_SSL_NOT_SUPPORTED’ msg sent by trudy1 to alice1.

2. Tcpdump trace captured at ‘bob1’:

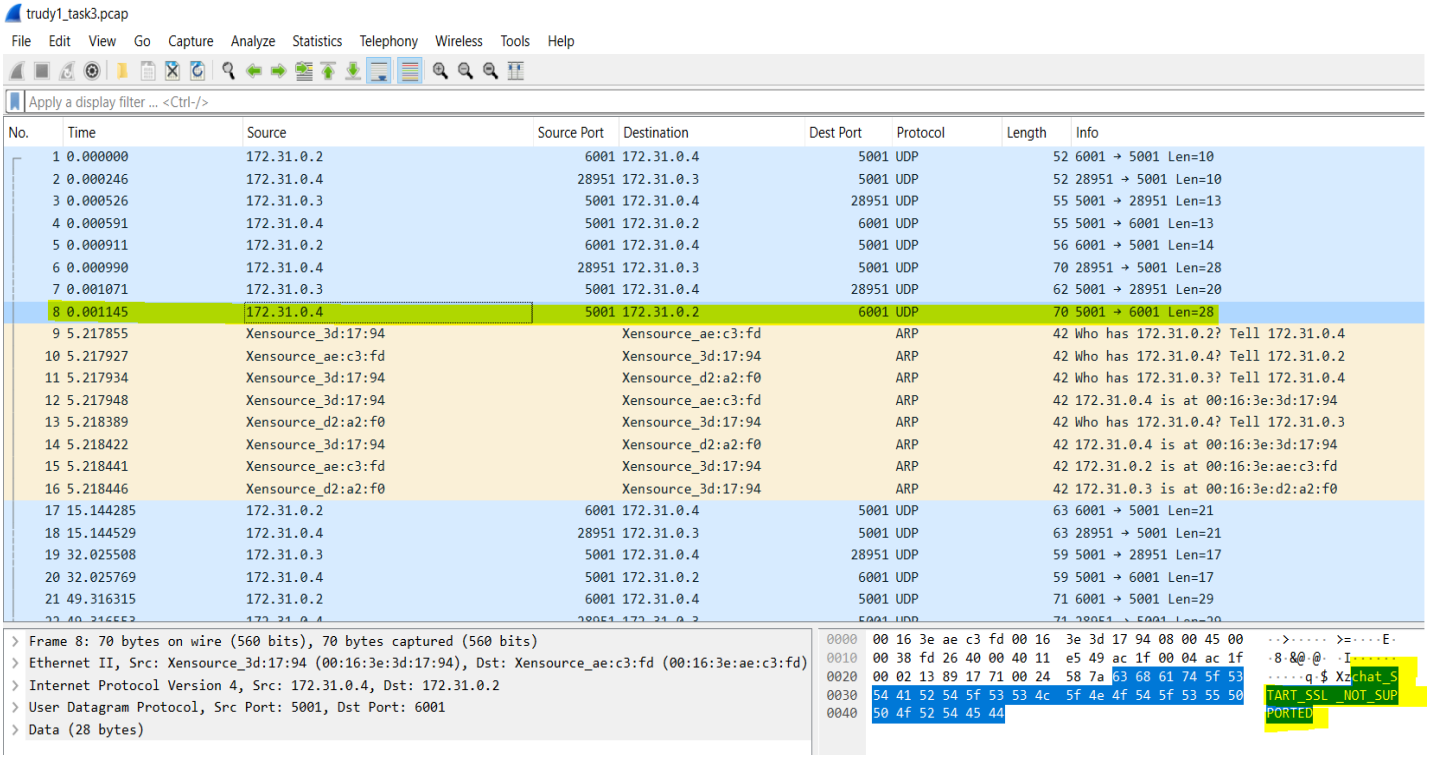


- We can observe that the chat messages are transferred in the plain text from this pcap trace.

- After receiving the ‘chat_close’ message from the client, the bob1 (server) closed the current client connection and started listening for new client connection requests.

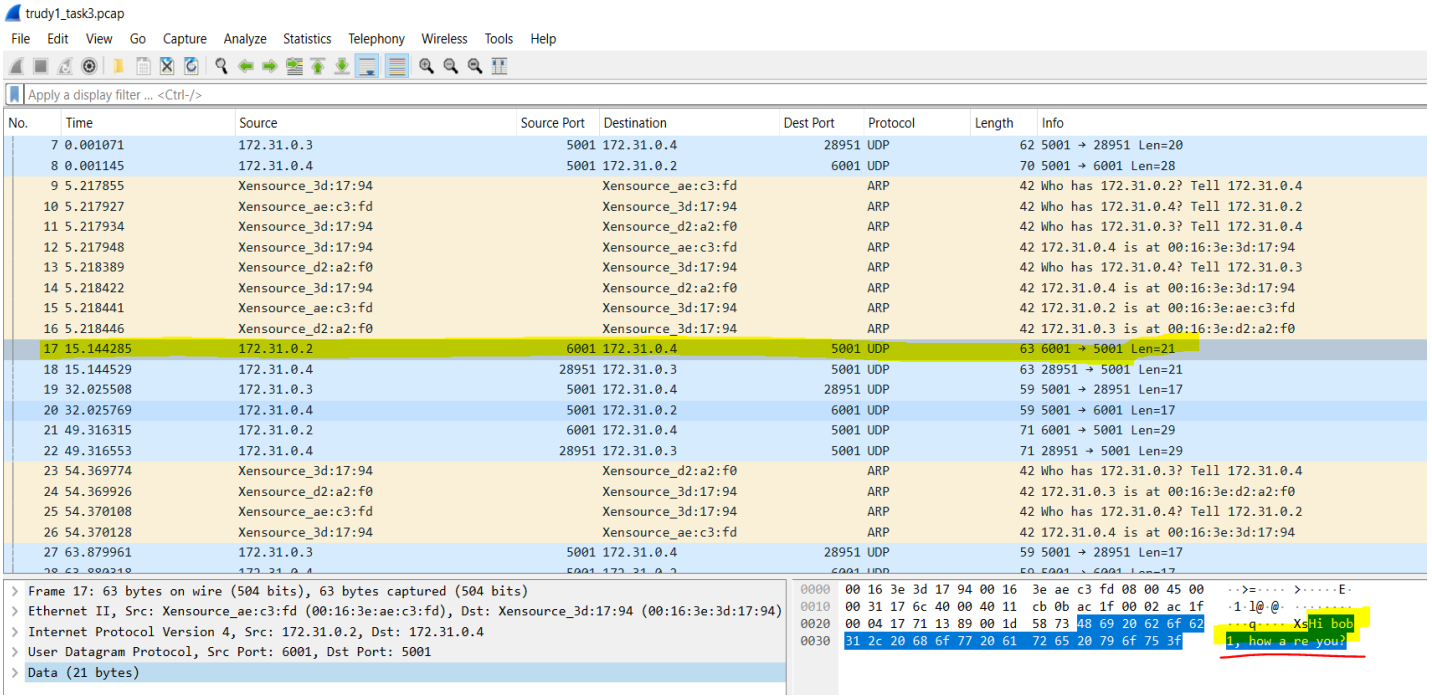


3. Tcpdump trace captured at ‘trudy1’:

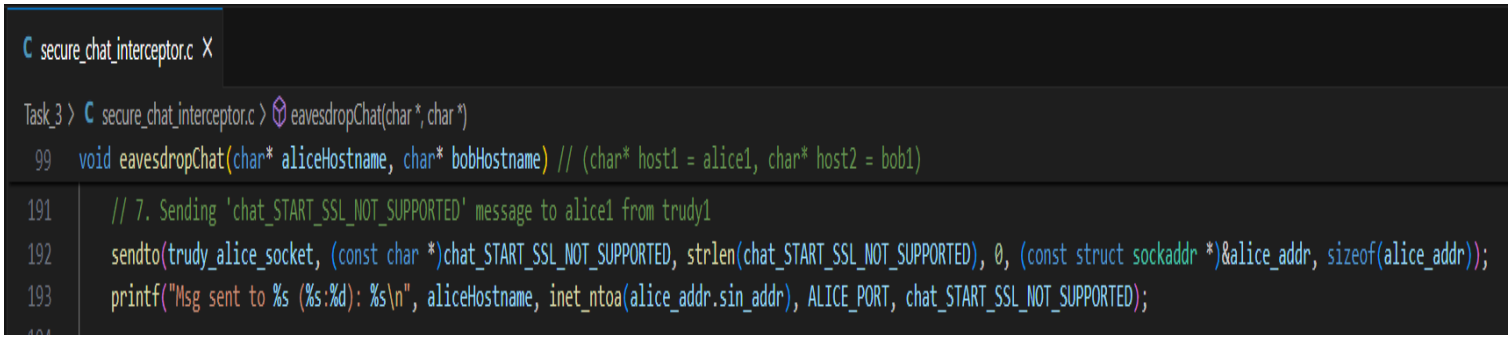


- We can observe in the pcap that trudy1 sends ‘chat_START_SSL_NOT_SUPPORTED’ message to alice1, hence performing the ‘Downgrade attack’.

- Trudy can observe all the chat messages communicated between alice1 and bob1, since they are sent in plain text.



Code snippet used by trudy1 to perform downgrade attack:



```
C secure_chat_interceptor.c X

Task_3 > C secure_chat_interceptor.c > eavesdropChat(char *, char *)
99  void eavesdropChat(char* aliceHostname, char* bobHostname) // (char* host1 = alice1, char* host2 = bob1)

191     // 7. Sending 'chat_START_SSL_NOT_SUPPORTED' message to alice1 from trudy1
192     sendto(trudy_alice_socket, (const char *)chat_START_SSL_NOT_SUPPORTED, strlen(chat_START_SSL_NOT_SUPPORTED), 0, (const struct sockaddr *)&alice_addr, sizeof(alice_addr));
193     printf("Msg sent to %s (%s:%d): %s\n", aliceHostname, inet_ntoa(alice_addr.sin_addr), ALICE_PORT, chat_START_SSL_NOT_SUPPORTED);
```

Code snippet used by client (alice1) to start unsecure chat:

```
C secure_chat_app.c X

Task_4 > C secure_chat_app.c > runClient(char *)
654  void runClient(char *serverHostName)
695      // exchange_client_messages(): returns 1 if unsecure chatting is to be performed, 0 for secure chatting
696      if (exchange_client_messages(client_socket, server_addr) == 1){
697          perform_clientside_unsecure_chatting(client_socket, server_addr.s4);
698          // return 0;
699          exit(1);
700      }
```

TASK 4: Active MITM attack

a) Answer:

i) Trudy generating fake certificate for alice1:

Generating alice1's private key and CSR:

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$ openssl genpkey -out fake_alice_private_key.pem -algorithm RSA -pkeyopt rsa_keygen_bits:2048
[illegible]
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$ openssl req -new -config fake_alice.cnf -key fake_alice_private_key.pem -out fake_alice.csr
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$ openssl req -in fake_alice.csr -noout -text | openssl dgst -md5 -out fake_alice_csr_digest
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$ openssl pkeyutl -sign -in fake_alice_csr_digest -out fake_alice_csr_sign -inkey fake_alice_private_key.pem
```

Int CA verifying the authenticity of alice1's CSR:

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/int$ openssl req -in fake_alice.csr -noout -text | openssl dgst -md5 -out fake_alice_csr_digest_generated
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/int$ openssl req -in fake_alice.csr -pubkey -noout > fake_alice_public_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/int$ openssl pkeyutl -verify -sigfile fake_alice_csr_sign -in fake_alice_csr_digest_generated -inkey fake_alice_public_key.pem -p
ubin
Signature Verified Successfully
```

Int CA generating 'fake crt' for alice1:

```
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/int$ openssl x509 -req -in fake_alice.csr -CA int.crt -extfile fake_alice.txt -CAkey int_private_key.pem -CAcreateserial -out fak
ealice.crt -days 730 -sha256
Certificate request self-signature ok
subject=CN = FakeAlicel.com, emailAddress = FakeAlice1@gmail.com, O = FakeAlice1, L = Nagpur, C = IN
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/int$ openssl x509 -in fakealice.crt -outform DER | openssl dgst -md5 > fake_alice_crt_digest
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/int$ openssl pkeyutl -sign -in fake_alice_crt_digest -out fake_alice_crt_sign -inkey int_private_key.pem
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/int$ openssl x509 -in fakealice.crt -outform DER | openssl dgst -md5 > fake_alice_crt_digest_generated
```

Alice1 verifying the authenticity of 'fake crt':

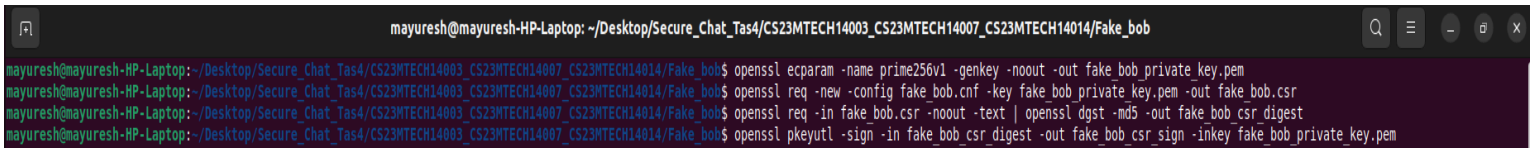
```
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$ openssl x509 -in fakealice.crt -outform DER | openssl dgst -md5 > fakealice_crt_digest_generated
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$ openssl pkeyutl -verify -sigfile fake_alice_crt_sign -in fakealice_crt_digest_generated -inkey int_public_key.pem -pu
bin
Signature Verified Successfully
```

Alice1 verifying the fake certificate:

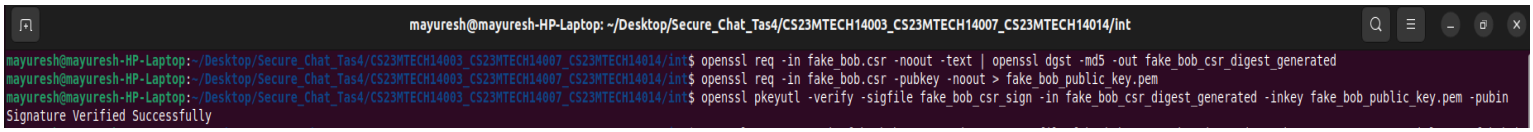
```
mayuresh@mayuresh-HP-Laptop: ~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice

mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$ openssl verify -CAfile root.crt -untrusted int.crt fakealice.crt
fakealice.crt: OK
mayuresh@mayuresh-HP-Laptop:~/Desktop/Secure_Chat_Tas4/CS23MTECH14003_CS23MTECH14007_CS23MTECH14014/Fake_alice$
```

ii) Trudy generating fake certificate for bob1:

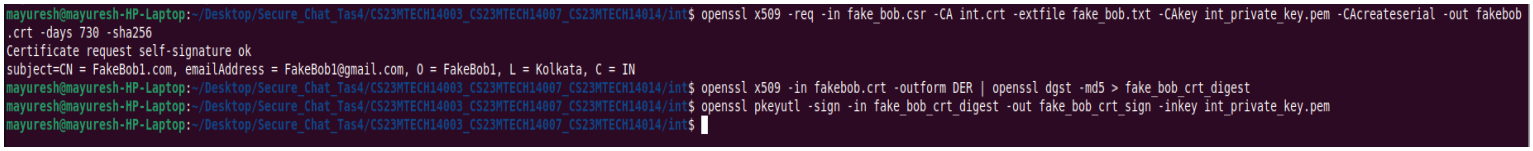
Generating bob1's private key and CSR:



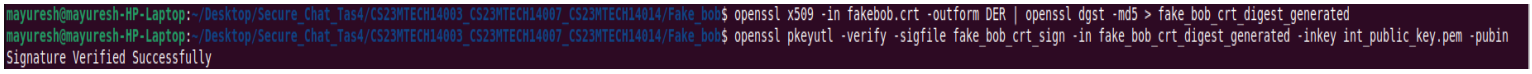
Int CA verifying the authenticity of bob1's CSR:



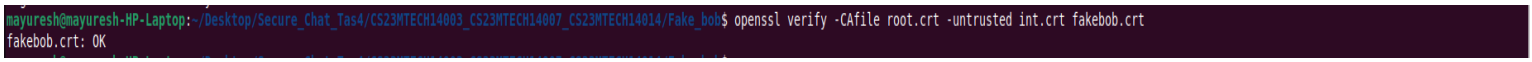
Int CA generating 'fake crt' for bob1:



Bob1 verifying the authenticity of 'fake crt'



Bob1 verifying the fake certificate:



Active MITM attack overview:

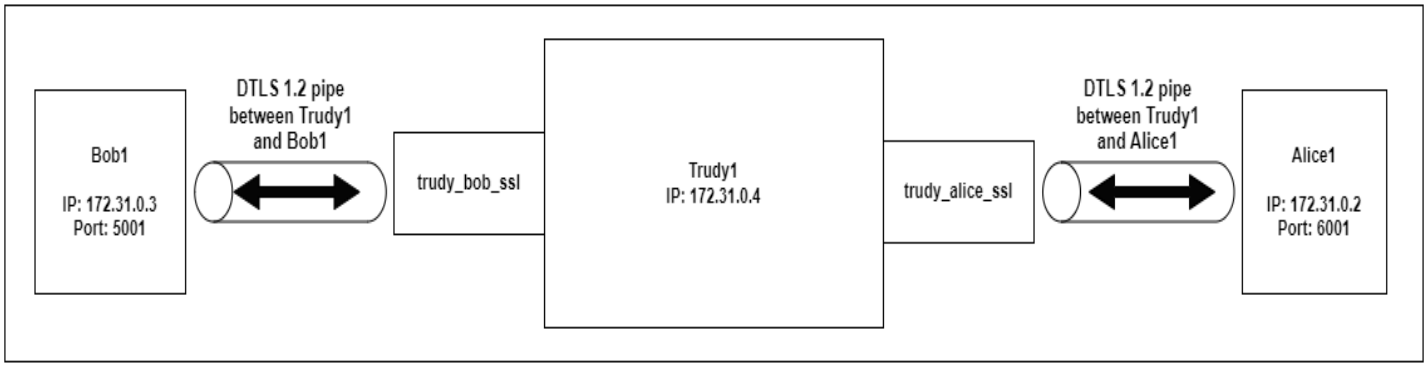


Fig: DTLS v1.2 pipes in active MITM attack

- Here, trudy1 performs an Active MITM attack by intercepting and modifying the chat messages communicated between alice1 and bob1.

- Trudy performs DNS poisoning by replacing the IP addresses of alice1 with bob1 and vice versa in the /etc/hosts file.

- Trudy1 forward the control messages exchanged between alice1 and bob1 without any tampering.

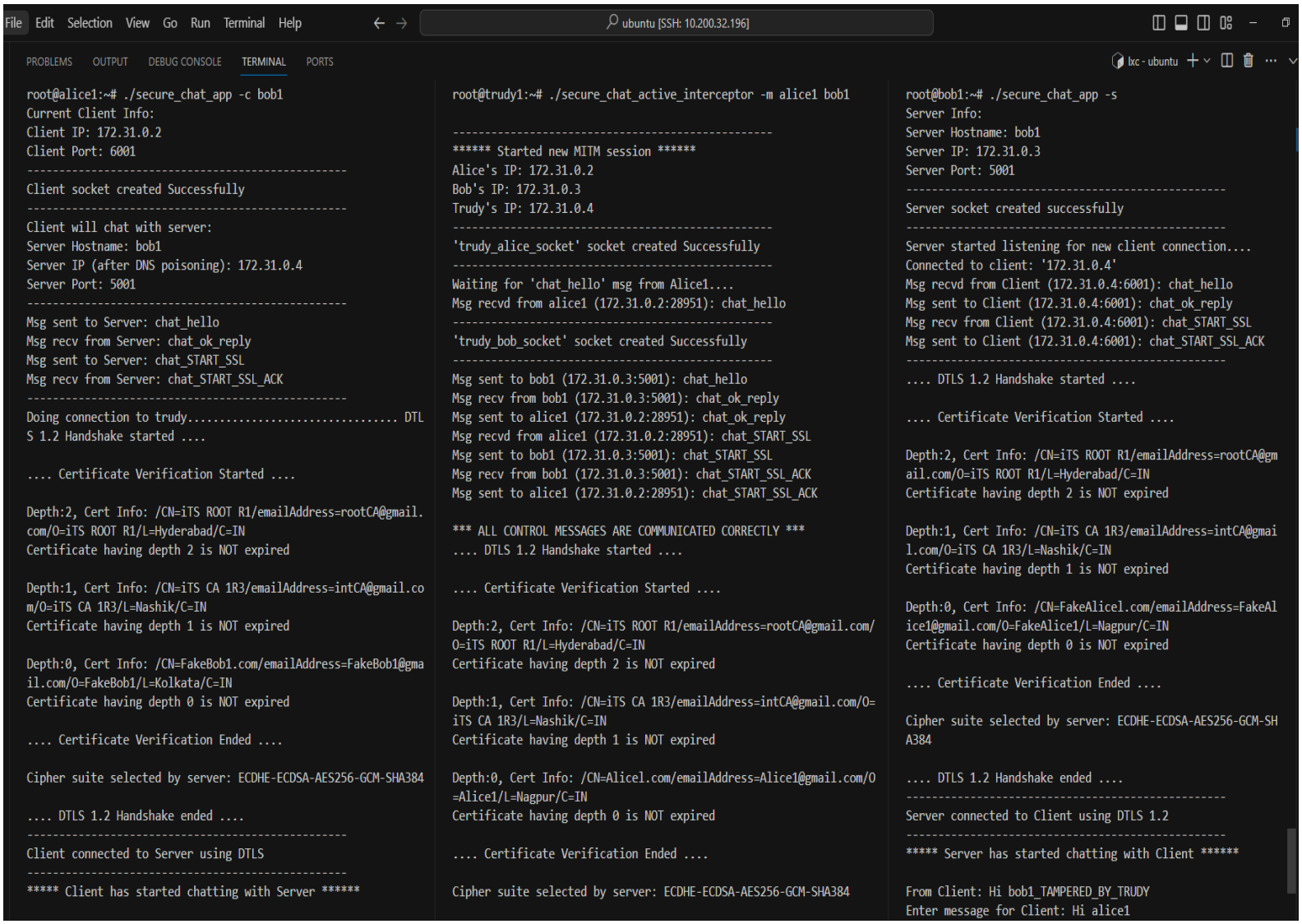
- Trudy1 establishes two DTLS v1.2 pipes:
    1. Between trudy1 and alice1
    2. Between trudy1 and bob1

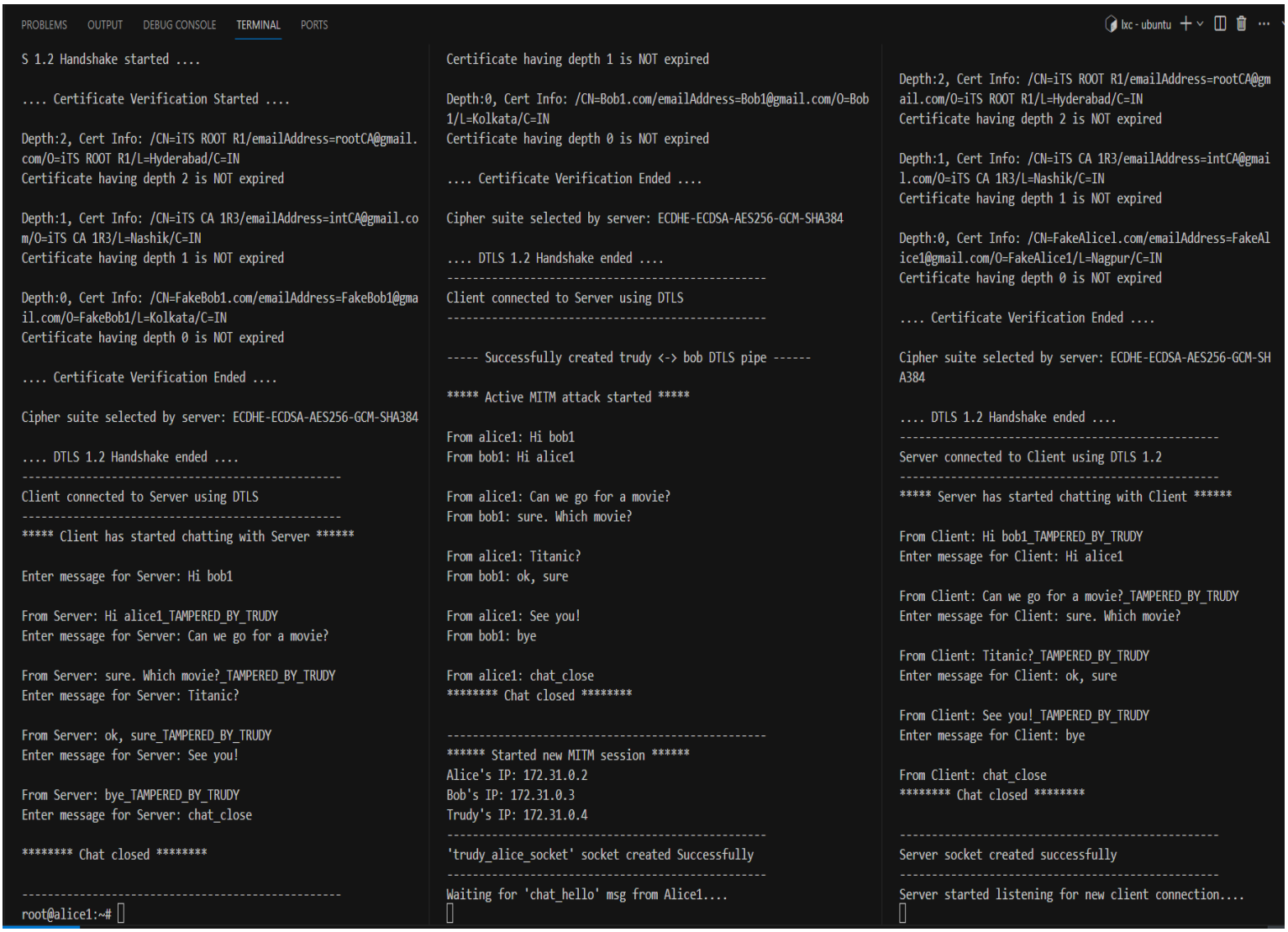
- During DTLS v1.2 handshake with alice1, trudy1 sends the bob1's fake certificate to alice1. And, during DTLS v1.2 handshake with bob1, trudy1 sends the alice1's fake certificate to bob1.

- Therefore, trudy1 can easily see as well as tamper the encrypted chat messages exchanged between alice1 and bob1.

- In this case, trudy tampers the messages by appending '***_TAMPERED_BY_TRUDY***' text to each chat message exchanged between alice1 and bob1.

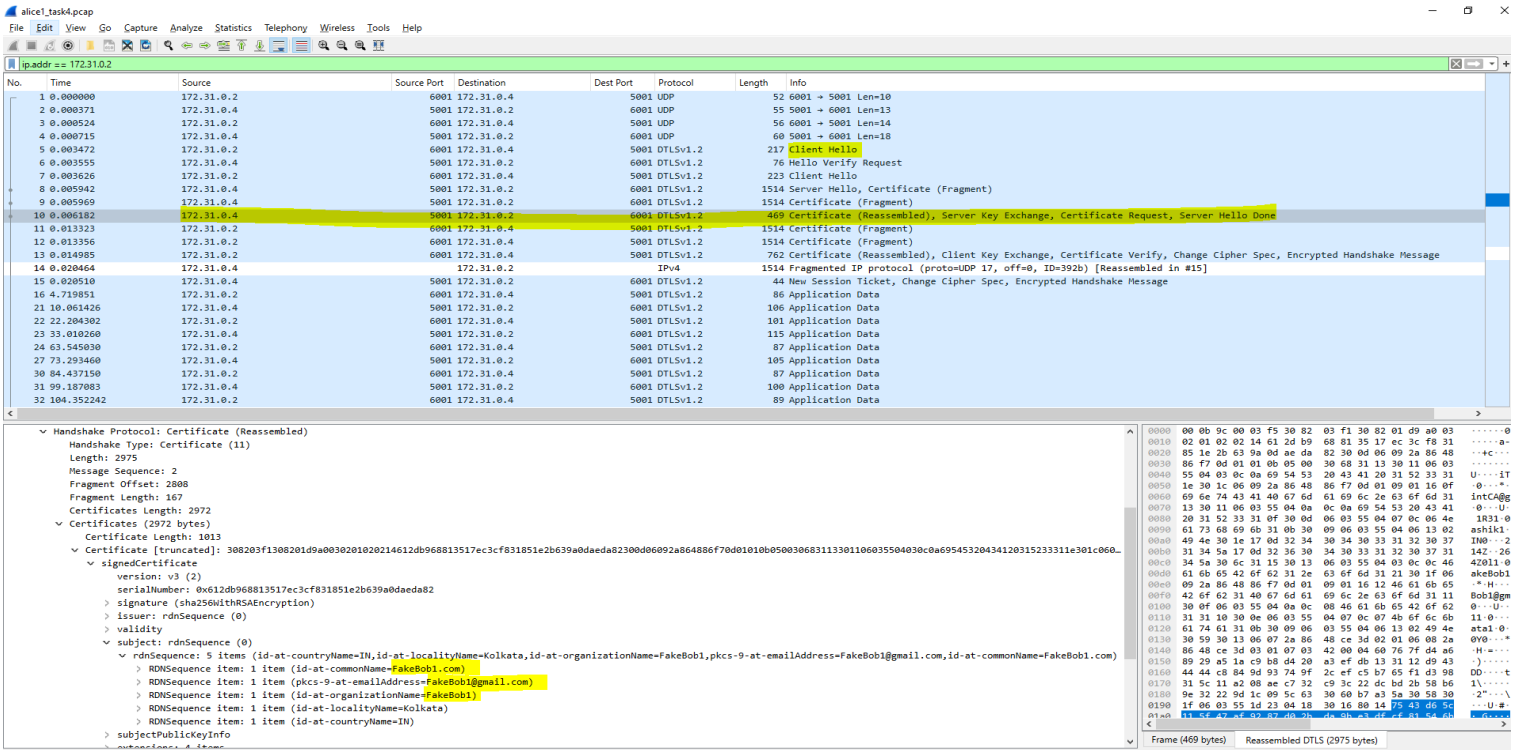
Active MITM attack on secure chat output:



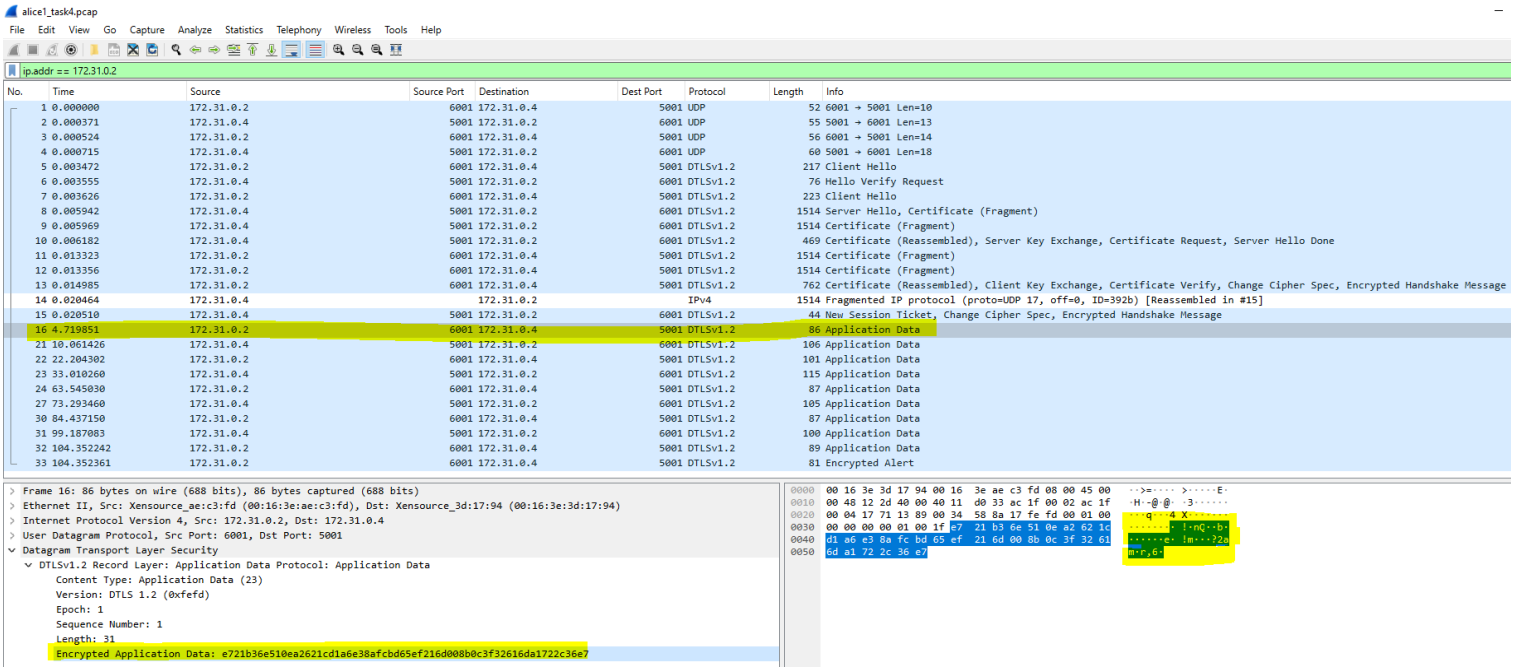

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
lxc - ubuntu
S 1.2 Handshake started ....
.... Certificate Verification Started ....
Depth:2, Cert Info: /CN=iTS ROOT R1/emailAddress=rootCA@gmail.com/O=iTS ROOT R1/L=Hyderabad/C=IN
Certificate having depth 2 is NOT expired
Depth:1, Cert Info: /CN=iTS CA 1R3/emailAddress=intCA@gmail.com/O=iTS CA 1R3/L=Nashik/C=IN
Certificate having depth 1 is NOT expired
Depth:0, Cert Info: /CN=FakeBob1.com/emailAddress=FakeBob1@gmail.com/O=FakeBob1/L=Kolkata/C=IN
Certificate having depth 0 is NOT expired
.... Certificate Verification Ended ....
Cipher suite selected by server: ECDHE-ECDSA-AES256-GCM-SHA384
.... DTLS 1.2 Handshake ended ....
Client connected to Server using DTLS
***** Client has started chatting with Server ******
Enter message for Server: Hi bob1
From Server: Hi alice1_TAMPERED_BY_TRUDY
Enter message for Server: Can we go for a movie?
From Server: sure. Which movie?_TAMPERED_BY_TRUDY
Enter message for Server: Titanic?
From Server: ok, sure_TAMPERED_BY_TRUDY
Enter message for Server: See you!
From Server: bye_TAMPERED_BY_TRUDY
Enter message for Server: chat_close
******** Chat closed ********
root@alice1:~#
Certificate having depth 1 is NOT expired
Depth:0, Cert Info: /CN=Bob1.com/emailAddress=Bob1@gmail.com/O=Bob1/L=Kolkata/C=IN
Certificate having depth 0 is NOT expired
.... Certificate Verification Ended ....
Cipher suite selected by server: ECDHE-ECDSA-AES256-GCM-SHA384
.... DTLS 1.2 Handshake ended ....
Client connected to Server using DTLS
----- Successfully created trudy <-> bob DTLS pipe ------
***** Active MITM attack started *****
From alice1: Hi bob1
From bob1: Hi alice1
From alice1: Can we go for a movie?
From bob1: sure. Which movie?
From alice1: Titanic?
From bob1: ok, sure
From alice1: See you!
From bob1: bye
From alice1: chat_close
******** Chat closed ********
****** Started new MITM session ******
Alice's IP: 172.31.0.2
Bob's IP: 172.31.0.3
Trudy's IP: 172.31.0.4
'trudy_alice_socket' socket created Successfully
Waiting for 'chat_hello' msg from Alice1....
Depth:2, Cert Info: /CN=iTS ROOT R1/emailAddress=rootCA@gmail.com/O=iTS ROOT R1/L=Hyderabad/C=IN
Certificate having depth 2 is NOT expired
Depth:1, Cert Info: /CN=iTS CA 1R3/emailAddress=intCA@gmail.com/O=iTS CA 1R3/L=Nashik/C=IN
Certificate having depth 1 is NOT expired
Depth:0, Cert Info: /CN=FakeAlice1.com/emailAddress=FakeAlice1@gmail.com/O=FakeAlice1/L=Nagpur/C=IN
Certificate having depth 0 is NOT expired
.... Certificate Verification Ended ....
Cipher suite selected by server: ECDHE-ECDSA-AES256-GCM-SHA384
.... DTLS 1.2 Handshake ended ....
Server connected to Client using DTLS 1.2
***** Server has started chatting with Client ******
From Client: Hi bob1_TAMPERED_BY_TRUDY
Enter message for Client: Hi alice1
From Client: Can we go for a movie?_TAMPERED_BY_TRUDY
Enter message for Client: sure. Which movie?
From Client: Titanic?_TAMPERED_BY_TRUDY
Enter message for Client: ok, sure
From Client: See you!_TAMPERED_BY_TRUDY
Enter message for Client: bye
From Client: chat_close
******** Chat closed ********
Server socket created successfully
Server started listening for new client connection....

1. Tcpdump trace captured at ‘alice1’:

- In the below pcap trace, we can observe that the ‘Client Hello’ message is sent from alice1 (172.31.0.2) to trudy1 (172.31.0.4) during DTLS v1.2 handshake.
- We can see that trudy1 has sent a ***‘fake certificate’*** of bob1 to alice1.

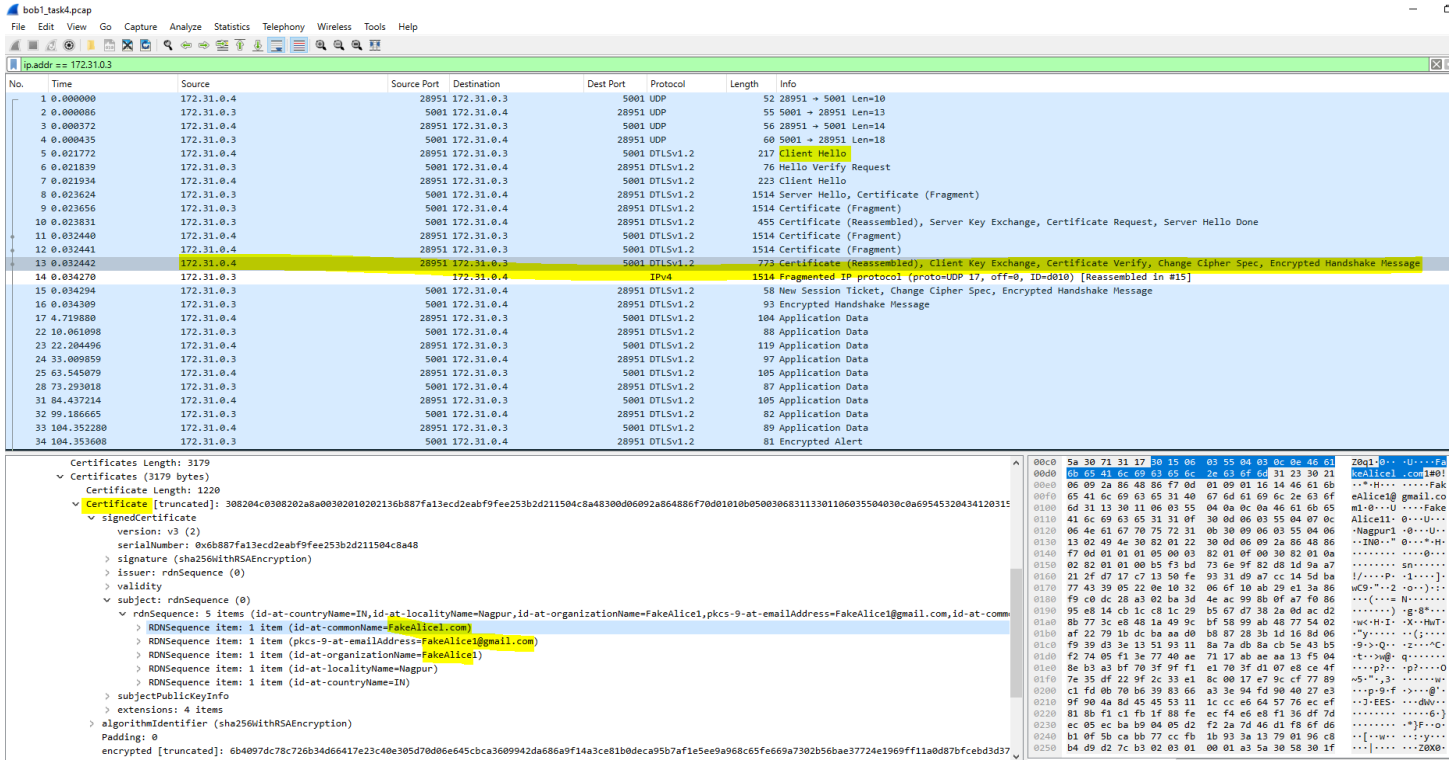


- From below pcap trace, we can observe that the chat messages between trudy1 and alice1 are encrypted using DTLS v1.2.

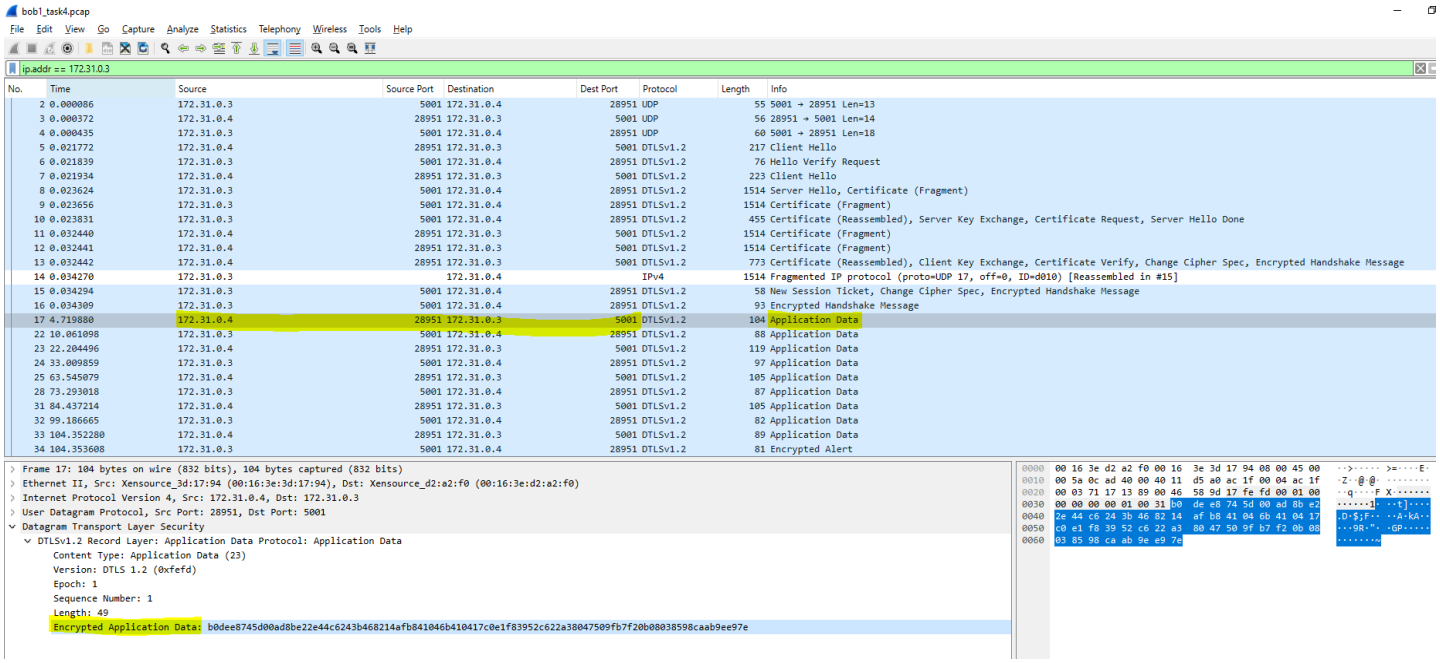


2. Tcpdump trace captured at ‘bob1’:

- In the pcap below, we can observe that the ‘Client Hello’ message is sent from trudy1 (172.31.0.4) to bob1 (172.31.0.3) during DTLS v1.2 handshake.
- We can see that trudy1 has sent a ***‘fake certificate’*** of alice1 to bob1.

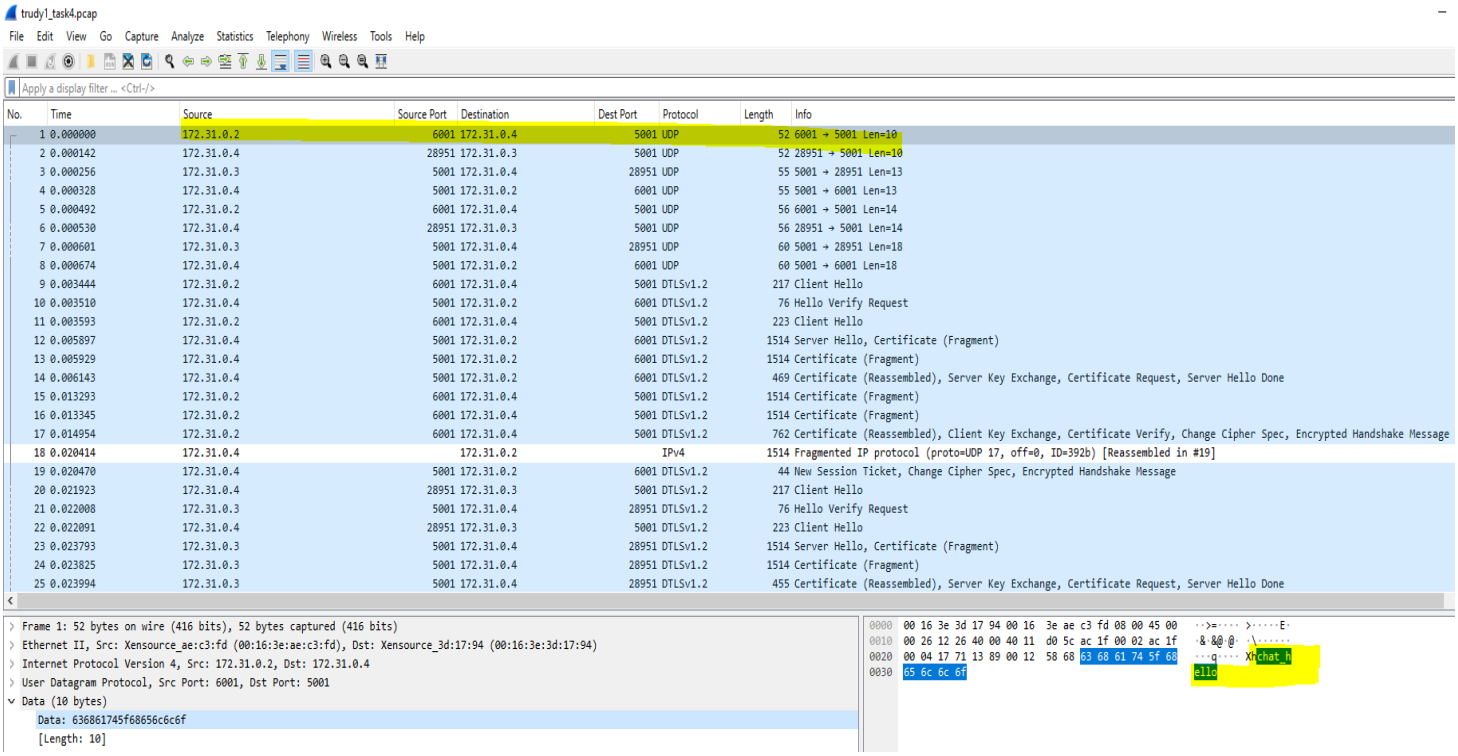


- From below pcap trace, we can observe that the chat messages between trudy1 and bob1 are encrypted using DTLS v1.2.

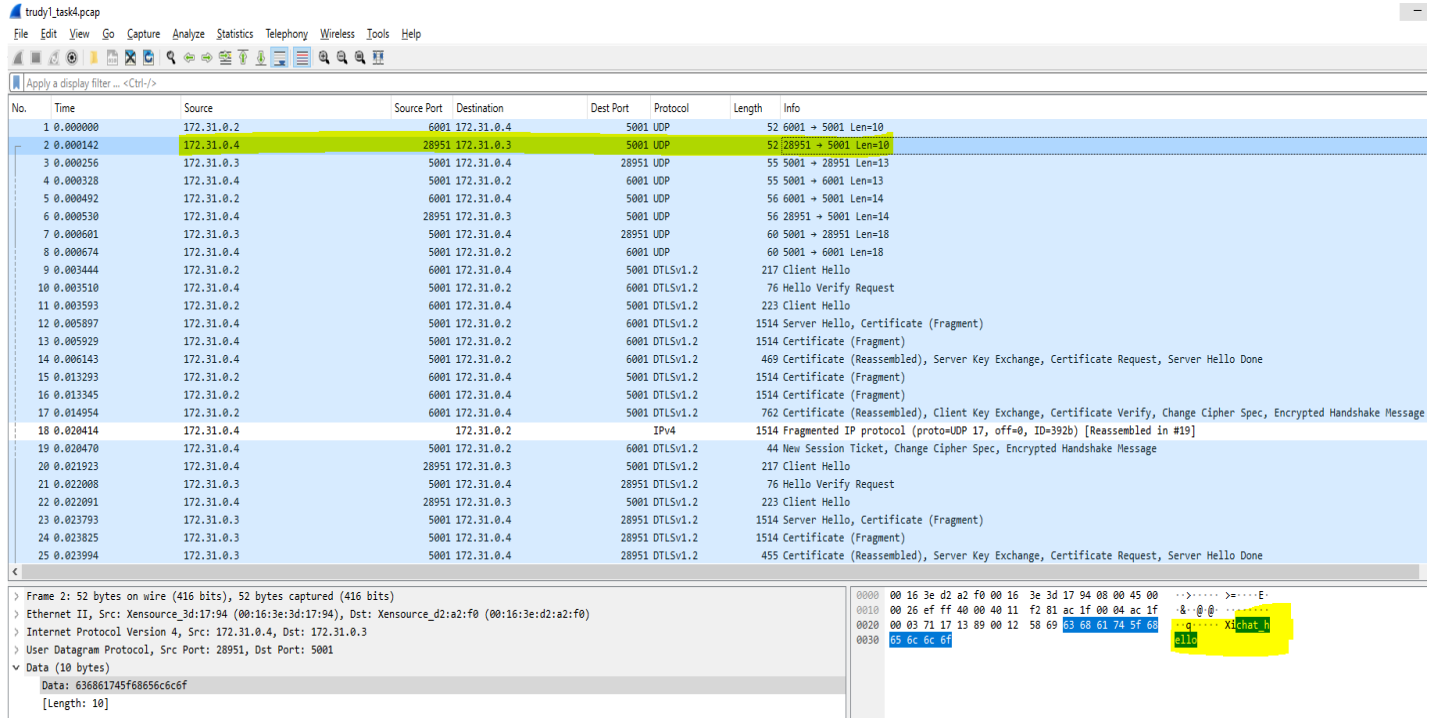


3. Tcpdump trace captured at ‘trudy1’:

- In the pcap below, we can observe that trudy1 (172.31.0.4) receives the ‘chat_hello’ plain text control message from alice1 (172.31.0.2).

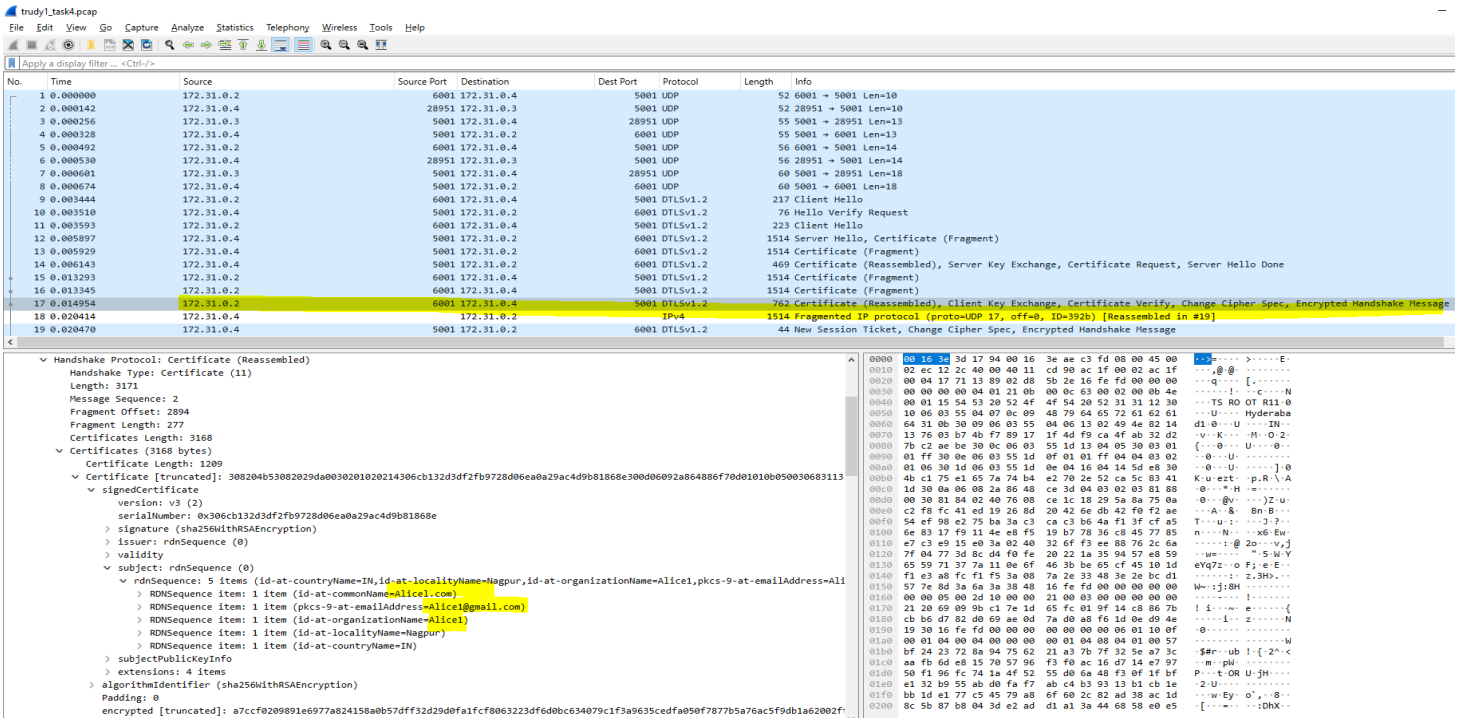


- Trudy1 (172.31.0.4) forwards this ‘chat_hello’ plain text control message to bob1 (172.31.0.3).
- Similarly, trudy1 forwards all the control messages sent from alice1 to bob1 and vice versa.



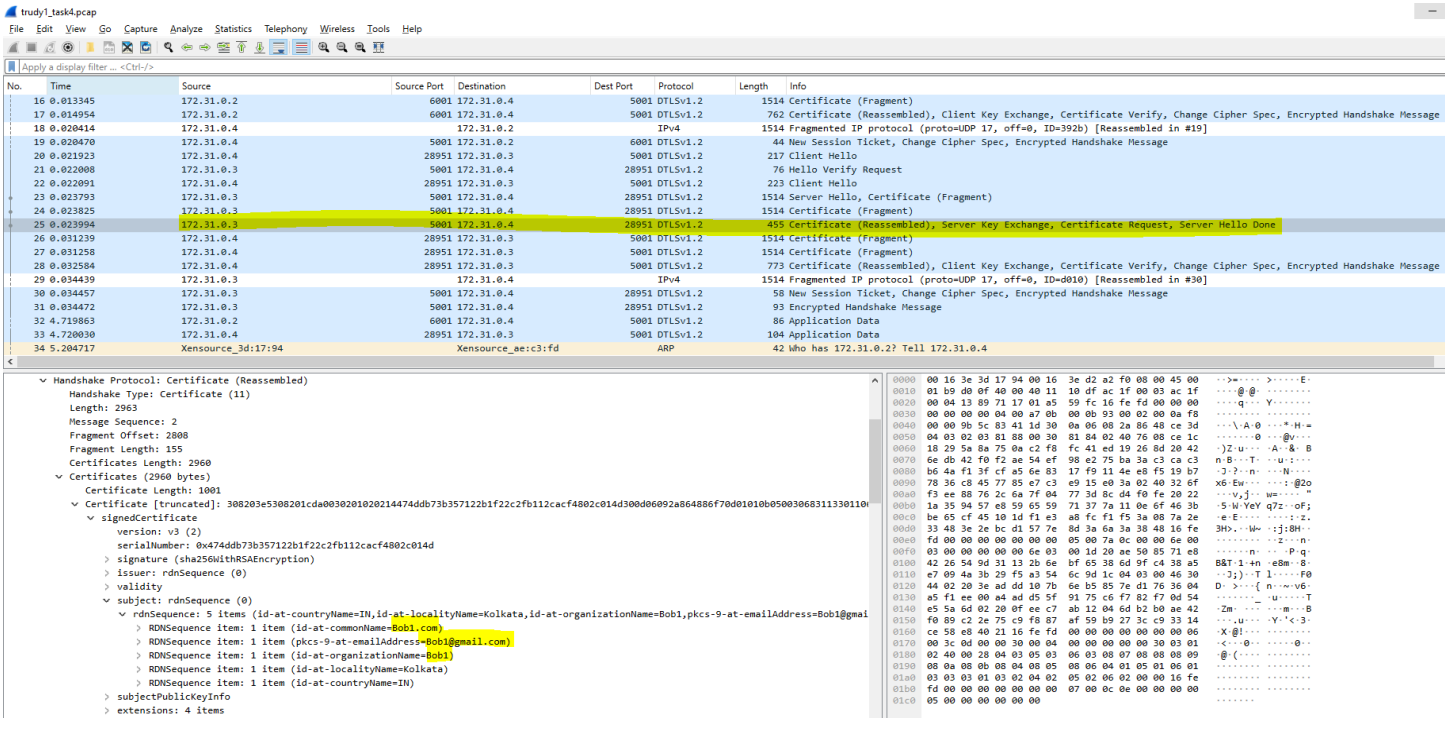
Certificate received by trudy1 from alice1:

- Alice1 (172.31.0.2) shares her genuine certificate to trudy1 (172.31.0.4) during DTLS v1.2 handshake.

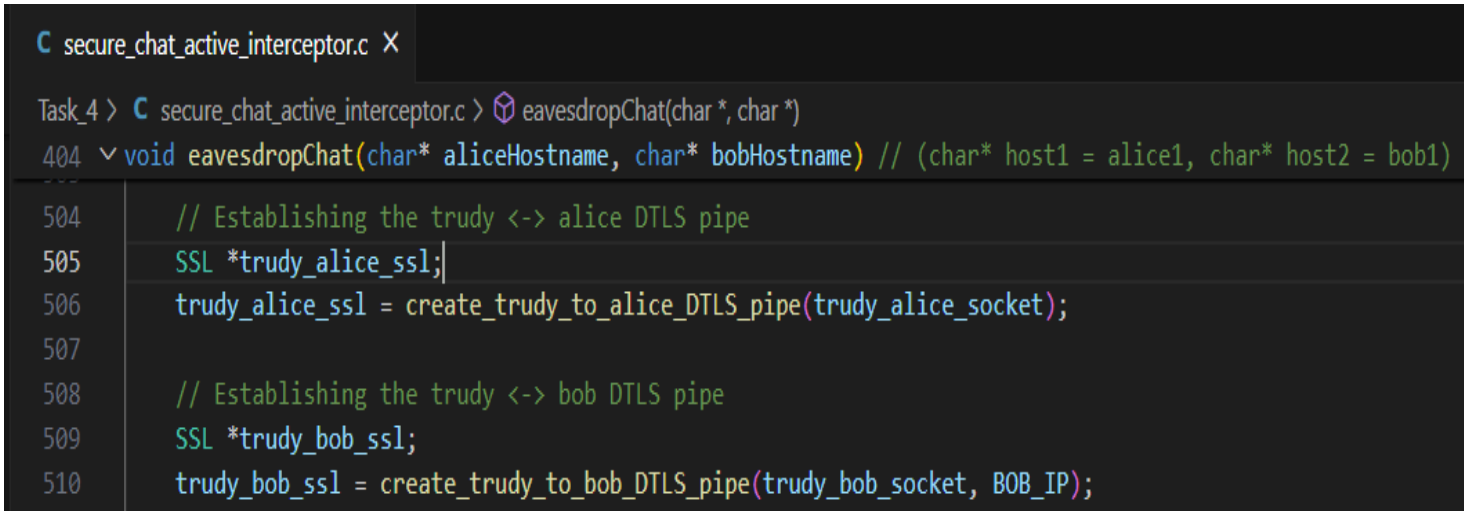


Certificate received by trudy1 from bob1:

- Bob1 (172.31.0.3) shares her genuine certificate to trudy1 (172.31.0.4) during DTLS v1.2 handshake.



Code used to establish two DTLS pipes:



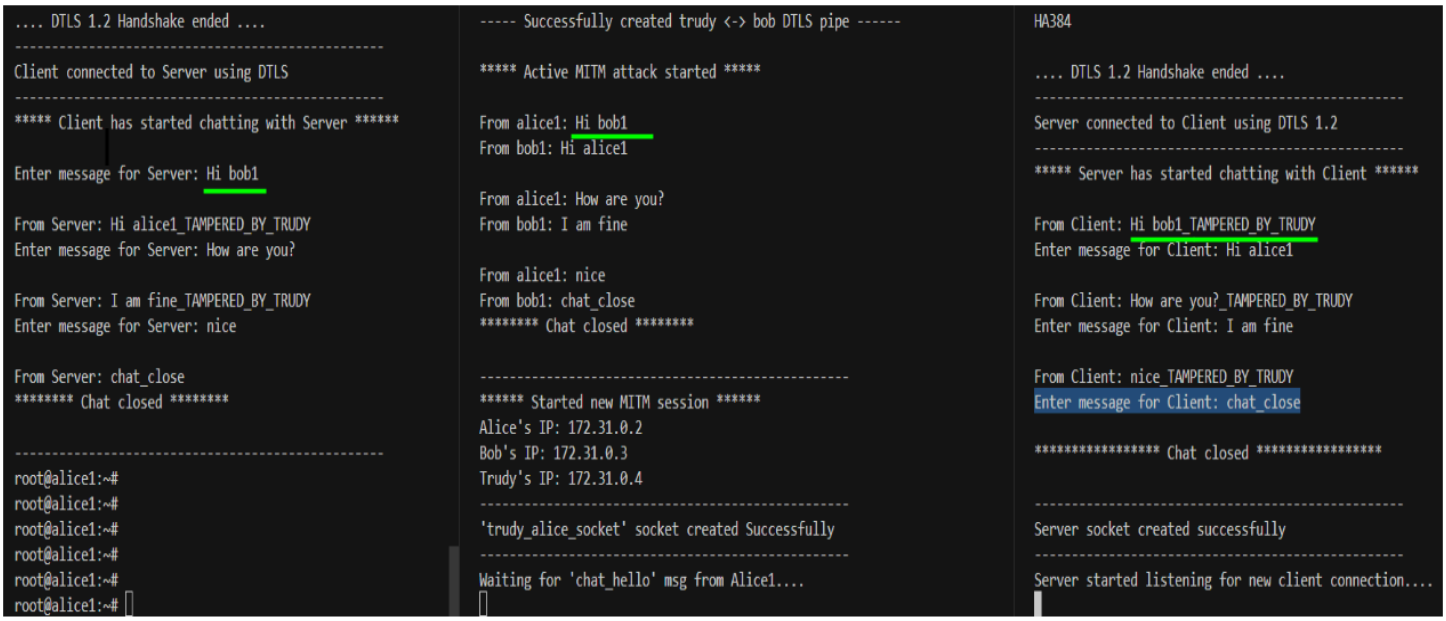
Two DTLS v1.2 pipes are established:

1. Between trudy1 and alice1 indicated by ‘trudy_alice_ssl’ in the code snippet
2. Between trudy1 and bob1 indicated by ‘trudy_bob_ssl’ in the code snippet

Active MITM attack performed by trudy1:

- In the below output screenshot, we can observe that, the trudy1 is performing the Active MITM attack by tampering the chat messages communicated between alice1 and bob1.

- Here, trudy1 is tampering the chat messages by appending "***_TAMPERED_BY_TRUDY***" text to them.



Code snippet used for msg tampering:

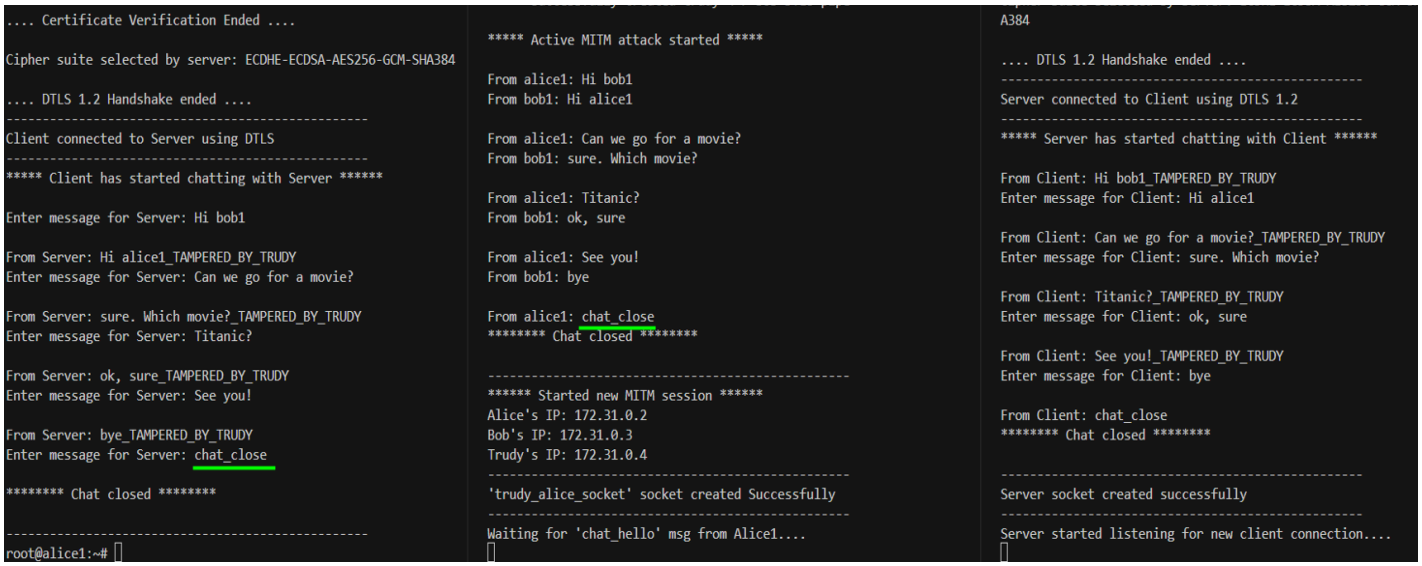
secure_chat_active_interceptor.c

Task_4 > secure_chat_active_interceptor.c > eavesdropChat(char *, char *)

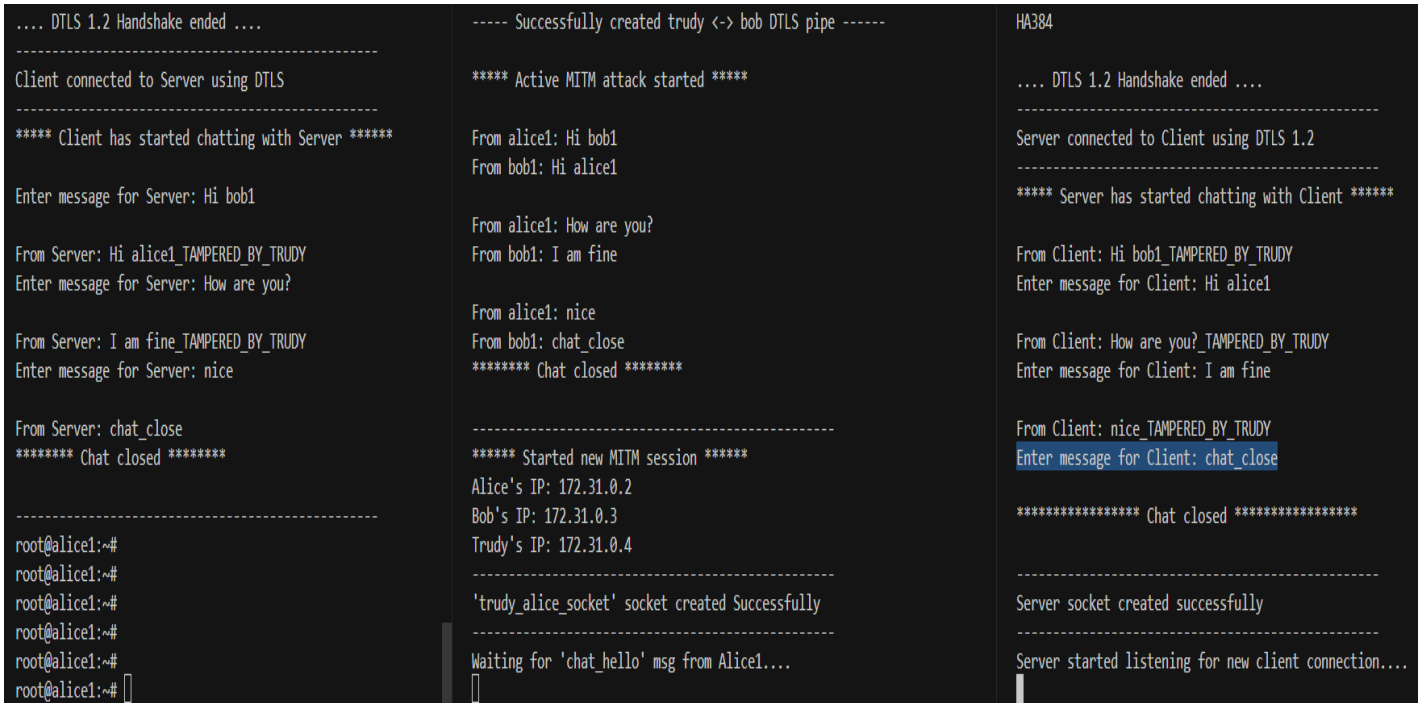
```
404 void eavesdropChat(char* aliceHostname, char* bobHostname) // (char* host1 = alice1, char* host2 = bob1)
513     printf("\n***** Active MITM attack started *****\n");
514     int len;
515     while(1)
516     {
517         // ------------------------------------------------------------ //...
519         memset(buffer, 0, sizeof(buffer));
520         do
521         {
522             len = SSL_read(trudy_alice_ssl, buffer, sizeof(buffer));
523         } while (len == -1);
524         if (len > 0)
525         {
526             printf("\nFrom alice1: %s", buffer);
527             if (strcmp(buffer, "chat_close") != 0) {
528                 strcat(buffer, "_TAMPERED_BY_TRUDY");
529             }
530         }
531         // write the modified msg to trudy_bob_ssl
532         len = SSL_write(trudy_bob_ssl, buffer, strlen(buffer));
533         if (len > 0)
534         {
535             if (strcmp(buffer, "chat_close") == 0)
536             {
537                 printf("\n******** Chat closed ********\n");
538                 break;
539             }
540         }
```

Chat close scenarios:

1. When client (alice1) closes the chat:



2. When server (bob1) closes the chat:



- **In both scenarios, when the chat is closed, both the trudy1 and bob1 keeps on listening for new client connection requests.**

Injecting packet loss

Injecting packet loss at ‘alice1’:

```
root@alice1:~# sudo tc qdisc add dev eth0 root netem loss 18%
root@alice1:~# sudo tc qdisc show dev eth0
qdisc netem 800d: root refcnt 3 limit 1000 loss 18%
root@alice1:~#
```

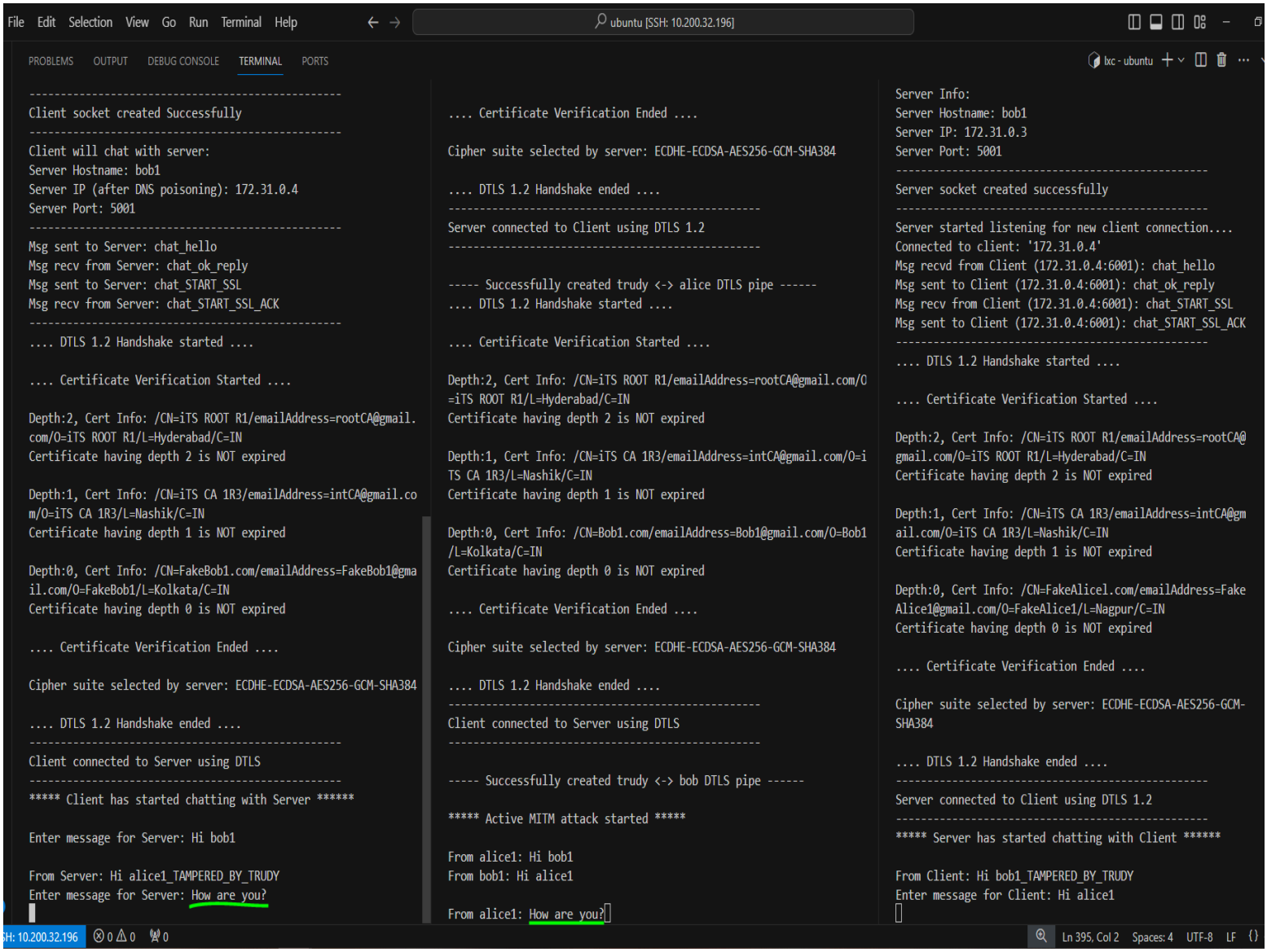
Injecting packet loss at ‘bob1’:

```
root@bob1:~# sudo tc qdisc add dev eth0 root netem loss 18%
root@bob1:~# sudo tc qdisc show dev  eth0
qdisc netem 800e: root refcnt 3 limit 1000 loss 18%
root@bob1:~#
```

Injecting packet loss at ‘trudy1’:

```
root@trudy1:~# sudo tc qdisc add dev eth0 root netem loss 18%
root@trudy1:~# sudo tc qdisc show dev eth0
qdisc netem 800f: root refcnt 3 limit 1000 loss 18%
root@trudy1:~#
```

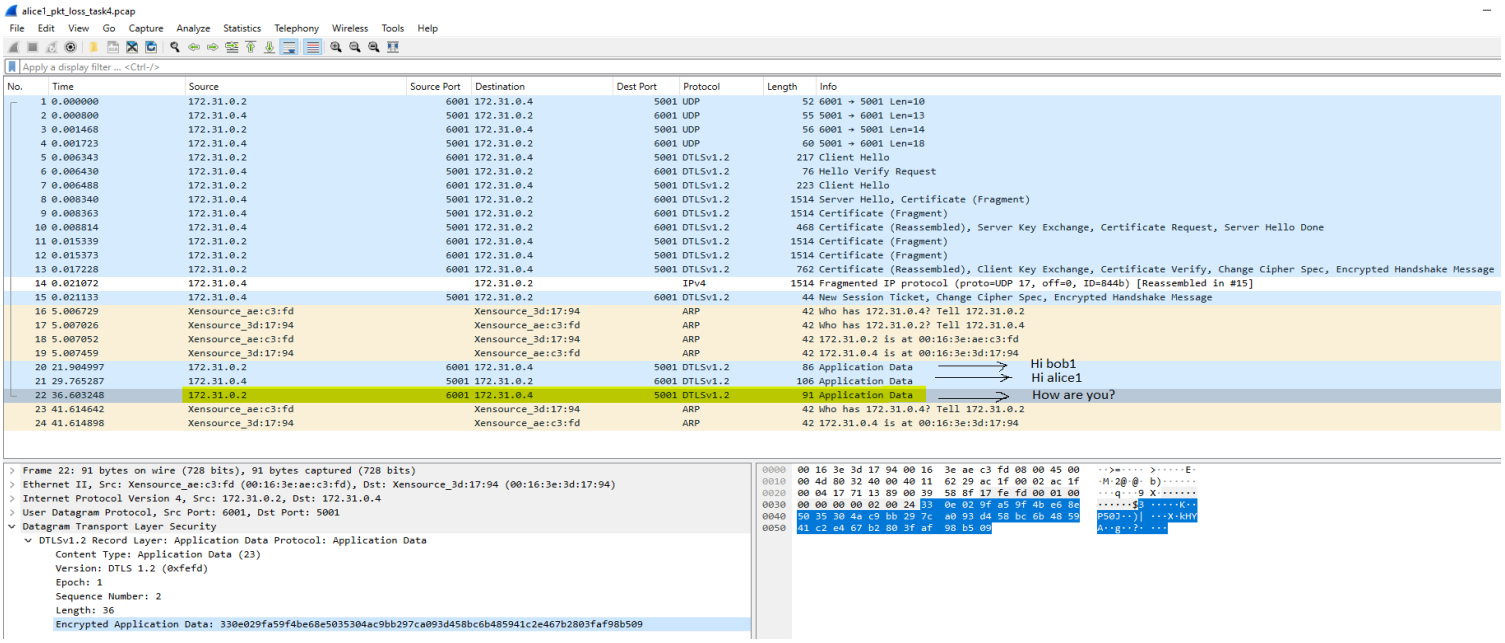
Secure chat along with active MITM output after injecting packet loss of 18%:



- In the above screenshot, we can observe that the chat message ‘How are you?’ sent from trudy1 to bob1 gets lost due to injected packet loss of 18%.

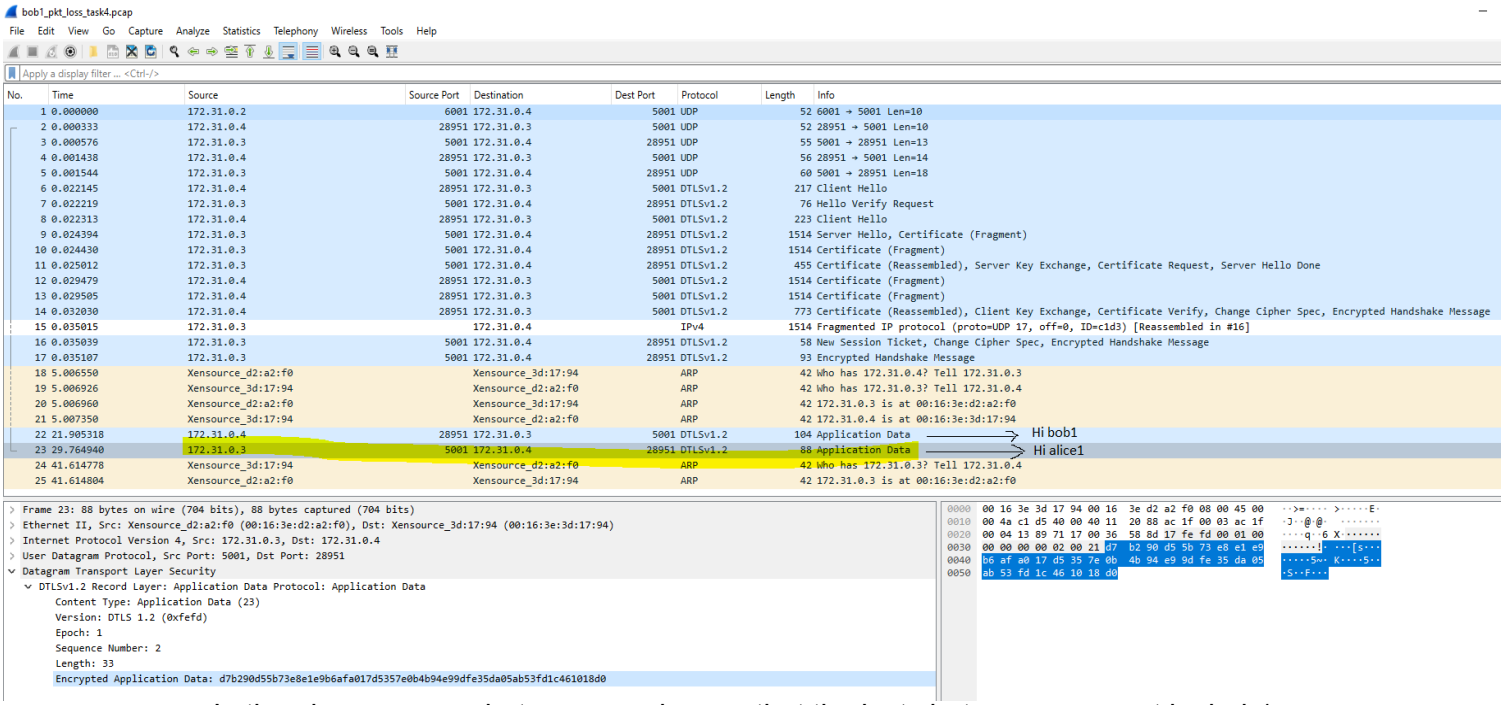
<u>Same thing can be confirmed from below tcpdump pcap traces:</u>

Tcpdump trace captured at alice1:



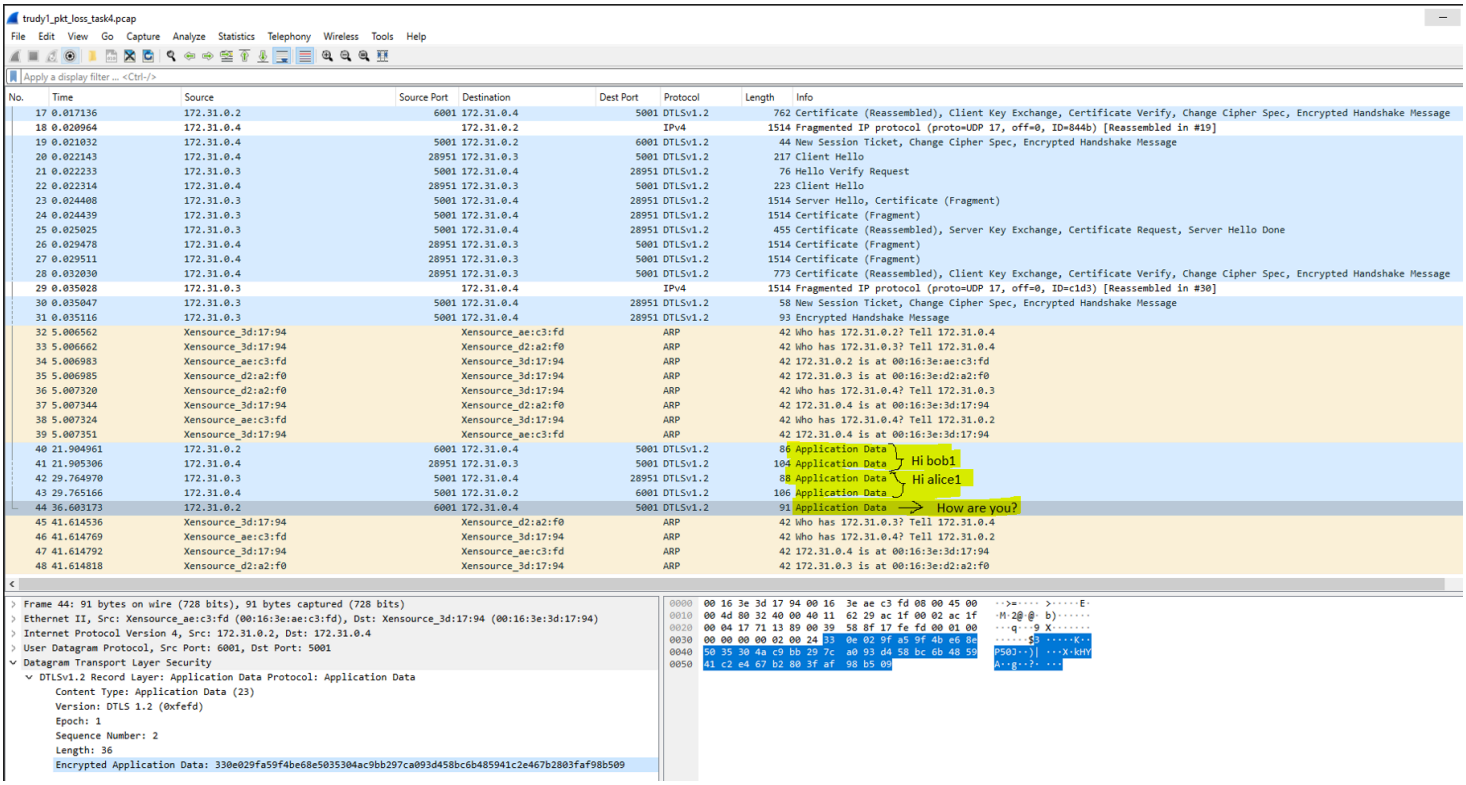
In the above screenshot, we can observe that the last chat message sent by alice1 (172.31.0.2) to trudy1 (172.31.0.4) is ‘How are you?’.

Tcpdump trace captured at bob1:



- In the above screenshot, we can observe that the last chat message sent by bob1 (172.31.0.3) to trudy1 (172.31.0.4) is ‘Hi alice1’.
- We can see that bob1 has not received the last message ‘How are you?’ sent by alice1.

Tcpdump trace captured at trudy1:



- From the above screenshot, we can observe that the first 2 chat messages ‘Hi bob1’ and ‘Hi alice1’ have been successfully exchanged between alice1 and bob1.

- However, the last chat message ‘How are you?’ sent from trudy1 to bob1 got lost due to injected packet loss.

Answer:

1. DTLS v1.2 protocol doesn’t provide reliability. Hence we can see that few chat messages got lost during communication.

2. To avoid this packet loss, we must implement some reliable data transfer mechanism to exchange the DTLS v1.2 messages.

Credit Statement

Task 1 contributions:

	Hrishikesh Hemke (CS23MTECH14003)	**Mayuresh Dindorkar (CS23MTECH14007)**	**Shrenik Ganguli (CS23MTECH14014)**
Software Installation	OpenSSL version 3.0.2		
Coding	Generated keys, CSR, digests and Certificates (CRT) for Root Certification Authority (Root CA) and verified them.	Generated keys, CSR, digests and Certificates (CRT) for Intermediate Certification Authority (Int CA) and verified them from Root CA.	Generated keys, CSR, digests and Certificates (CRT) for Alice1 and Bob1 and verified them from Intermediate CA.
Report writing	Written report for Root CA.	Written report for Intermediate CA.	Written report for Alice1 and Bob1.
Bug fixes	*Bug*: The trust chain was not getting formed in the alice1 and bob1's certificate. *Resolution*: Use the configuration and extension file.		

Task 2 contributions:

	Hrishikesh Hemke (CS23MTECH14003)	**Mayuresh Dindorkar (CS23MTECH14007)**	**Shrenik Ganguli (CS23MTECH14014)**
Coding	Run the commands to capture the tcpdump pcap traces at both alice1 and bob1's end. Also, injected the packet loss in the secure chat application using 'tc-netem' command and captured the corresponding pcap files.	Implemented the DTLS v1.2 connection between alice1 and bob1 using openssl in C programming language. Implemented the certificate exchange and verification logic during DTLS v1.2 handshake. Also implemented the cipher suites negotiation logic.	Implemented the exchange of control messages between alice1 and bob1 using UDP sockets. Studied the difference between DTLS 1.2 and TLS 1.2 handshakes.
Testing	Tested the cipher suites negotiation functionality.	Tested the correctness of control message exchange between alice1 and bob1.	Tested the certificate exchange and verification logic during DTLS v1.2 handshake.
Report writing	Written the report for the above work.	Written the report for the above work.	Written the report for the above work.

Bug fixes	*Bug*: ‘key too small’ error for 1024-bit RSA key of alice1. *Resolution*: Used a 2048-bit RSA key for alice1.
Additional features	1. The server (bob1) keeps on listening for new client connection requests when the chat with the current client is closed. 2. Trimmed the preceding as well as trailing white spaces present in the chat messages.

Task 3 contributions:

	Hrishikesh Hemke (CS23MTECH14003)	**Mayuresh Dindorkar (CS23MTECH14007)**	**Shrenik Ganguli (CS23MTECH14014)**
Coding	Modified the control message flow to in corporate ‘START_SSL_NOT_SUPPORTED’ message. Implemented the logic in the code to divert the execution from secure to unsecure chat.	Implemented the unsecure chat using two UDP sockets: 1. ‘trudy_alice_socket’ between truy1 and alice1. 2. ‘Trudy_bob_socket’ between trudy1 and bob1.	Designed the architecture of all modules involved in code including trudy1. Run the commands to capture the tcpdump pcap traces at alice1, bob1 and trudy1 containers.
Report writing	Written the report for the above work.	Written the report for the above work.	Written the report for the above work.
Additional features	1. The server (bob1) keeps on listening for new client connection requests when the chat with the current client is closed. 2. Trimmed the preceding as well as trailing white spaces present in the chat messages.		

Task 4 contributions:

	Hrishikesh Hemke (CS23MTECH14003)	**Mayuresh Dindorkar (CS23MTECH14007)**	**Shrenik Ganguli (CS23MTECH14014)**
Coding	Run the commands to capture the tcpdump pcap traces at alice1, bob1 and trudy1 containers. Injected the packet loss in the secure chat application using 'tc-netem' command and captured the corresponding pcap files.	Implemented the two DTLS v1.2 pipes: 1. Between trudy1 and alice1 2. Between trudy1 and bob1 Also implemented the logic for transfer of fake certificates from trudy1 to alice1 as well as bob1.	Generated the fake certificates (CRT), CSR, digests and keys for alice1 and bob1. Verified these fake certificates using intermediate CA.
Report writing	Written the report for the above work.	Written the report for the above work.	Written the report for the above work.

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: Hrishikesh Hemke , Dindorkar Mayuresh Rajesh , Shrenik Ganguli
Date: 07/04/2024
Signature: HH, MRD, SG