Network Security

Assignment-6

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: Akash Tadwai, Sai Varshittha, Amit Kumar
Date: 08/04/2022
Signature: AT, SVP, AK

Task-1

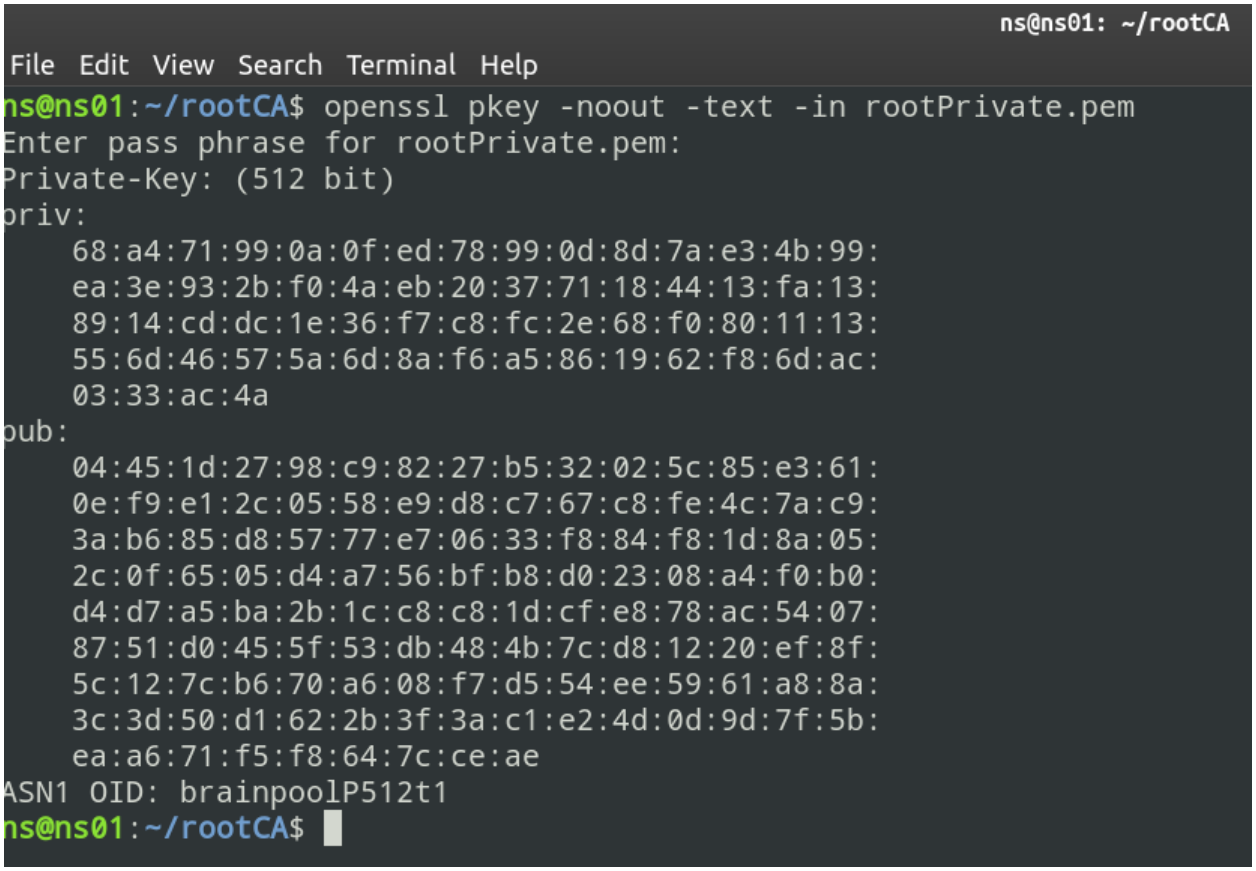
Root CA Self Signed Certificate

Root CA generates a self signed certificate which would then be the Root certificate.

EC512 key generation

```
$ openssl genpkey -algorithm EC -out rootPrivate.pem -pkeyopt
ec_paramgen_curve:brainpoolP512t1 -aes256
```

Displaying rootPrivate.pem



Generating self signed Certificate

```
$ openssl req -new -x509 -days 365 -key rootPrivate.pem -out root.crt
```

Displaying root.crt

```
ns@ns01:~/rootCA$
ns@ns01:~/rootCA$ openssl x509 -text -noout -in root.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            54:28:10:41:cc:bb:35:b0:80:15:0c:f6:47:22:3e:50:f4:62:99:fc
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = rootCA, emailAddress = es18btech11019@iith.ac.in
        Validity
            Not Before: Apr  7 05:49:15 2022 GMT
            Not After : Apr  7 05:49:15 2023 GMT
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = rootCA, emailAddress = es18btech11019@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:45:1d:27:98:c9:82:27:b5:32:02:5c:85:e3:61:
                    0e:f9:e1:2c:05:58:e9:d8:c7:67:c8:fe:4c:7a:c9:
                    3a:b6:85:d8:57:77:e7:06:33:f8:84:f8:1d:8a:05:
                    2c:0f:65:05:d4:a7:56:bf:b8:d0:23:08:a4:f0:b0:
                    d4:d7:a5:ba:2b:1c:c8:c8:1d:cf:e8:78:ac:54:07:
                    87:51:d0:45:5f:53:db:48:4b:7c:d8:12:20:ef:8f:
                    5c:12:7c:b6:70:a6:08:f7:d5:54:ee:59:61:a8:8a:
                    3c:3d:50:d1:62:2b:3f:3a:c1:e2:4d:0d:9d:7f:5b:
                    ea:a6:71:f5:f8:64:7c:ce:ae
                ASN1 OID: brainpoolP512t1
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                B9:F5:92:6B:79:E1:B7:38:6F:7E:EC:CD:A4:2C:E3:96:EC:05:30:79
            X509v3 Authority Key Identifier:
                keyid:B9:F5:92:6B:79:E1:B7:38:6F:7E:EC:CD:A4:2C:E3:96:EC:05:30:79

            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: ecdsa-with-SHA256
         30:81:85:02:40:46:b9:92:ba:73:ea:92:02:f5:85:7d:72:a0:
         fd:b6:18:d3:2e:ee:e6:46:4e:97:d3:c5:36:82:d1:dd:34:0f:
         8d:0e:69:ea:15:61:a9:10:42:32:36:b6:5e:8d:b6:76:e2:6b:
         38:70:dd:f1:fa:0f:95:53:e4:88:52:45:1f:3f:d5:02:41:00:
         a9:7e:4b:ee:1a:0b:d0:3c:56:78:01:66:22:da:f9:9d:d9:78:
         e9:f4:52:a8:ba:e0:81:0e:cf:a7:2a:b4:b0:96:8f:34:43:a9:
         0f:b0:66:08:4c:e2:73:77:1c:97:b0:2c:5e:df:cb:c7:f2:70:
         9f:96:ee:30:0b:37:34:66:42:23
ns@ns01:~/rootCA$
```

Alice Key & CSR Generation

Alice generates key-pair and csr files in alice1 container.

For generating key-pair files:

```
$ openssl genrsa -out alicePrivate.pem 2048
```

For separating public Key

```
$ openssl pkey -in alicePrivate.pem -pubout -out alicePublic.pem
```

For creating new CSR request:

```
$ openssl req -new -key alicePrivate.pem -out aliceCSR.csr
```

Displaying ailceCSR.csr :

```
root@alice1:~# openssl req -noout -verify -text -in aliceCSR.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = alice1, emailAddress = cs18btech11035@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c3:54:38:04:65:7e:88:34:30:58:81:ab:a6:6b:
                    4a:61:cb:84:cb:75:56:c7:b3:8d:2a:42:67:d3:2f:
                    15:cf:2d:d1:6f:ae:25:2c:0b:df:54:df:43:6f:71:
                    6b:df:b2:e2:0d:66:80:6a:9d:fb:80:4d:98:7e:73:
                    53:ba:be:90:60:40:9c:ee:67:89:82:77:3d:3c:f9:
                    10:71:63:58:f6:ac:7b:20:6d:c3:f6:41:37:c1:28:
                    06:fb:cd:9f:6e:d1:22:83:18:14:0f:42:b7:9d:3d:
                    1a:62:c1:8f:63:f5:90:0f:bb:1b:c0:62:6f:eb:81:
                    af:b8:90:46:fe:af:71:5d:dc:79:84:0b:a9:74:23:
                    a9:6a:96:0f:0b:42:5c:82:7c:92:ed:58:f8:53:43:
                    3f:31:74:31:51:be:c3:26:ce:8b:df:28:90:35:76:
                    d6:85:e5:5e:2b:9a:ce:75:b1:3a:61:12:2d:b5:4a:
                    db:87:bb:d3:c1:73:cb:63:83:a1:07:fc:35:7c:80:
                    1b:0d:08:69:d7:05:2f:b2:e6:4e:d4:8f:ff:59:43:
                    20:25:38:7e:e0:d0:a0:01:6b:c3:bf:c3:8b:b5:60:
                    9d:be:41:2a:f4:8e:ca:58:e4:21:91:59:aa:3d:15:
                    43:78:e7:8b:8c:44:48:24:71:01:4e:00:ab:09:63:
                    2b:37
                Exponent: 65537 (0x10001)
        Attributes:
            unstructuredName         :IITH
            challengePassword        :ns01
    Signature Algorithm: sha256WithRSAEncryption
         1b:da:57:e6:17:b7:46:33:08:f4:bb:9c:07:09:18:c8:c5:dd:
         52:14:eb:b0:f1:0b:e9:a9:90:83:32:87:57:39:65:49:cc:fc:
         6b:87:d6:eb:25:a7:26:b3:44:b2:ae:4f:2e:5b:4d:11:75:5b:
         1c:38:94:01:93:4b:3a:e3:ad:86:f7:18:3d:e3:8d:9c:e3:f9:
         03:44:82:03:b4:28:3a:c6:da:6e:94:2c:c8:33:26:ae:ea:1a:
         55:2e:41:c6:35:2d:5c:03:89:11:5d:4a:82:cc:e0:63:0a:75:
         8d:bf:51:2f:ca:e9:71:43:fd:90:4d:be:08:c8:8f:f2:f7:a0:
         7f:77:87:74:17:75:19:61:2d:47:88:19:3b:da:03:38:cf:fa:
         76:c7:30:d7:40:aa:56:16:7e:e8:66:b1:a3:ed:0e:a9:ef:75:
         17:de:be:1f:44:d8:c9:8a:57:1b:56:e5:c2:16:c1:ec:04:21:
         cf:07:de:3c:0b:cb:bb:7a:f9:5a:51:70:59:68:f9:fb:41:bc:
         94:0b:e4:4d:04:e2:7e:7f:bc:29:20:82:e1:bb:ce:3f:31:01:
         e2:8d:03:43:6b:64:67:f5:27:96:25:d4:ed:41:69:63:13:f4:
         75:45:ff:f6:bb:40:84:2a:c6:0d:d8:2c:9a:3a:2b:6b:5d:47:
         27:3a:cf:69
```

Bob Key and CSR generation

Bob generates key-pair and csr files in bob1 container.

For generating key-pair files:

```
$ openssl genrsa -out bobPrivate.pem 2048
```

For separating public Key

```
$  openssl pkey -in bobPrivate.pem -pubout -out bobPublic.pem
```

For creating new CSR request:

```
$ openssl req -new -key bobPrivate.pem -out bobCSR.csr
```

Displaying Bob's csr request

```
$ openssl req -noout -verify -text -in bobCSR.csr
```

```
root@bob1:~# openssl req -noout -verify -text -in bobCSR.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = bob1, emailAddress = es18btech11019@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:de:90:f7:99:35:13:65:3d:49:46:1d:23:c0:15:
                    f7:35:08:07:32:68:97:01:76:e2:34:f0:f1:0f:d5:
                    7d:b7:6e:9b:be:2b:2b:9e:ce:c5:db:51:6c:f0:b0:
                    d1:ca:c3:19:9b:18:2d:9a:c6:22:80:7b:37:59:91:
                    46:db:71:10:c7:16:36:48:22:07:66:b4:63:33:cf:
                    6a:fb:26:fa:e6:27:00:3c:94:8a:e3:25:4e:82:d2:
                    6c:2c:dd:34:80:8b:14:27:5d:92:8e:a5:1b:06:38:
                    6d:f5:db:d3:22:95:3f:14:4c:93:8e:2e:68:c4:d0:
                    e5:9e:18:3a:59:a4:51:4f:99:20:c1:a6:19:55:06:
                    a7:02:a6:c9:31:44:38:c2:24:1e:13:c2:3c:bb:56:
                    de:8a:33:06:36:7f:5e:b3:8c:82:7e:e6:e7:d0:91:
                    19:29:31:67:08:07:e0:28:32:34:d3:29:e8:22:e0:
                    db:b2:1d:9e:5d:ea:78:9c:dd:16:e0:70:aa:bf:df:
                    bd:12:56:e8:f6:1f:3b:31:77:03:ee:bc:e3:41:28:
                    1b:6f:1b:f5:e8:5b:fd:db:71:fd:cf:06:0d:76:af:
                    98:37:1c:21:76:74:14:ab:5c:78:fa:2c:59:ed:c0:
                    72:4a:3b:54:15:b6:26:14:46:91:ba:12:6e:3d:dc:
                    78:0d
                Exponent: 65537 (0x10001)
        Attributes:
            unstructuredName         :IITH
            challengePassword        :ns01
    Signature Algorithm: sha256WithRSAEncryption
         bc:d7:d4:31:ea:99:46:c0:88:be:1b:4c:a5:a0:af:2b:16:1d:
         e5:7a:40:38:af:a9:82:5f:9b:a2:89:8e:a2:56:1a:a1:0e:8d:
         86:d4:c3:65:25:e4:d0:55:27:f7:07:e0:1f:dc:c0:3d:70:c6:
         b2:7f:57:a4:da:9b:79:23:0f:7d:d0:97:39:ea:5e:0f:23:f4:
         ba:43:2c:27:8f:f7:73:f6:f9:42:f6:87:bb:4f:0b:bc:74:c1:
         52:b6:6f:6a:53:83:a0:08:b3:75:45:05:8d:81:dc:33:43:a8:
         27:c8:19:3a:bf:8c:79:65:00:b2:f8:fd:a2:cc:e3:58:b6:ed:
         f7:f5:16:a2:a5:fa:4d:7a:4f:9a:42:a2:16:2c:34:bf:35:43:
         99:36:b0:11:15:64:91:21:00:5d:21:8e:71:10:f4:85:e6:cf:
         d5:f2:d8:b5:67:bb:09:9e:22:1f:03:42:35:4f:1c:2a:14:e6:
         82:06:38:3d:1a:83:d7:3a:2a:f3:d9:cf:bb:c5:21:b1:76:ad:
         7c:b3:f1:87:13:16:26:36:a9:27:bd:bd:c2:55:f8:4d:25:36:
         a0:3b:07:d9:e6:7d:7d:af:59:19:45:3e:3d:e4:55:8a:05:f0:
         d3:0f:25:53:5b:8e:9c:3a:3f:78:17:3d:96:a4:ed:97:45:ba:
         37:22:3a:76
```

Verification and Issuing Certificates

Now Alice sends the csr file to rootCA to get its certificate issued. For rootCA to verify that alice is indeed the person who is the owner of the domain given in its csr, rootCA asks Alice to sign some message with its private key and send it back. After successfully verifying that Alice is indeed the owner of the domain in CSR, rootCA signs the csr, generates its certificate and sends it back to Alice.
For Alice to verify that the csr is indeed signed by rootCA, she uses openssl verify command.

```
ns@ns01:~/rootCA$ cat verify.txt
This message is to Verify the Authenticity of domain name in csr request.
ns@ns01:~/rootCA$
```

In VM:

```
$ lxc file push verify.txt alice1/root/ #sending verify.txt to alice
```

In alice1 container:

$ openssl dgst -sha256 -sign alicePrivate.pem -out sha256alice.sign verify.txt

```
root@alice1:~# openssl dgst -sha256 -sign alicePrivate.pem -out sha256alice.sign verify.txt
root@alice1:~# ls
aliceCSR.csr  alicePrivate.pem  alicePublic.pem  root.crt  secure_chat_app.py  sha256alice.sign  snap  test  utils.py  verify.txt
root@alice1:~#
```

In VM:

```
$ lxc file pull alice1/root/{aliceCSR.csr,sha256alice.sign} ./
```

```
ns@ns01:~/alice$ lxc file pull alice1/root/{aliceCSR.csr,sha256alice.sign} ./
ns@ns01:~/alice$ ls
aliceCSR.csr  sha256alice.sign
ns@ns01:~/alice$
```

To verify that the csr request indeed came from alice and is not tampered in the middle, we should extract the public key of alice from csr and then verify the signed digest.

To extract the public key of Alice from the csr file we use the following command:

```
$ openssl req -in aliceCSR.csr -noout -pubkey -out alicePublic.pem
```

Verifying the -sha256 digest

```
$ openssl dgst -sha256 -verify alicePublic.pem -signature
sha256alice.sign ../rootCA/verify.txt
```

```
ns@ns01:~/alice$ ls
aliceCSR.csr  alicePublic.pem  sha256alice.sign
ns@ns01:~/alice$ openssl dgst -sha256 -verify alicePublic.pem -signature sha256alice.sign ../rootCA/verify.txt
Verified OK
ns@ns01:~/alice$
```

Now rootCA issues the certificate to alice

```
$ openssl x509 -req -days 365 -in aliceCSR.csr -CA ../rootCA/root.crt
-CAkey ../rootCA/rootPrivate.pem -CAcreateserial -out alice.crt
```

```
ns@ns01:~/alice$ openssl x509 -req -days 365 -in aliceCSR.csr -CA ../rootCA/root.crt -CAkey ../rootCA/rootPrivate.pem -CAcreateserial -out alice.crt
Signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = alice1, emailAddress = cs18btech11035@iith.ac.in
Getting CA Private Key
Enter pass phrase for ../rootCA/rootPrivate.pem:
ns@ns01:~/alice$
```

rootCA sends alice.crt file to alice and again alice verifies that it is indeed signed by rootCA by using openssl verify command. (Assuming that alice1 container already has root.crt in its rootstore).

ns@ns01:~/alice$ `lxc file push alice.crt alice1/root/`

Verifying the certificate using openssl

```
$ openssl verify -verbose -CAfile root.crt alice.crt
```

```
ns@ns01: ~/rootCA                    root@alice1: ~
root@alice1:~# ls
alice.crt  aliceCSR.csr  alicePrivate.pem  alicePublic.pem  root.crt  sha256.sign  snap  test  verify.txt
root@alice1:~# openssl verify -verbose -CAfile root.crt alice.crt
alice.crt: OK
root@alice1:~#
```

We do the same things for Bob to generate his certificate and verify that any information isn't tampered in all phases. The screenshots of bob are shown below

```
root@bob1:~# ls
bobCSR.csr  bobPrivate.pem  bobPublic.pem  snap  verify.txt
root@bob1:~# openssl dgst -sha256 -sign bobPrivate.pem -out sha256bob.sign verify.txt
Enter pass phrase for bobPrivate.pem:
root@bob1:~#
```

```
ns@ns01:~/rootCA$ cd ../bob && lxc file pull bob1/root/{bobCSR.csr,sha256bob.sign} ./
ns@ns01:~/bob$ ls
bobCSR.csr  sha256bob.sign
ns@ns01:~/bob$
```

```
ns@ns01:~/bob$ openssl dgst -sha256 -verify bobPublic.pem -signature sha256bob.sign ../rootCA/verify.txt
Verified OK
ns@ns01:~/bob$
```

```
ns@ns01:~/bob$ openssl x509 -req -days 365 -in bobCSR.csr -CA ../rootCA/root.crt -CAkey ../rootCA/rootPrivate.pem -CAcreateserial -out bob.crt
Signature ok
subject=C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = bob, emailAddress = es18btech11019@iith.ac.in
Getting CA Private Key
Enter pass phrase for ../rootCA/rootPrivate.pem:
ns@ns01:~/bob$ ls
bob.crt  bobCSR.csr  bobPublic.pem  sha256bob.sign
ns@ns01:~/bob$
```

```
root@bob1:~# openssl verify -verbose -CAfile root.crt bob.crt
bob.crt: OK
root@bob1:~#
```

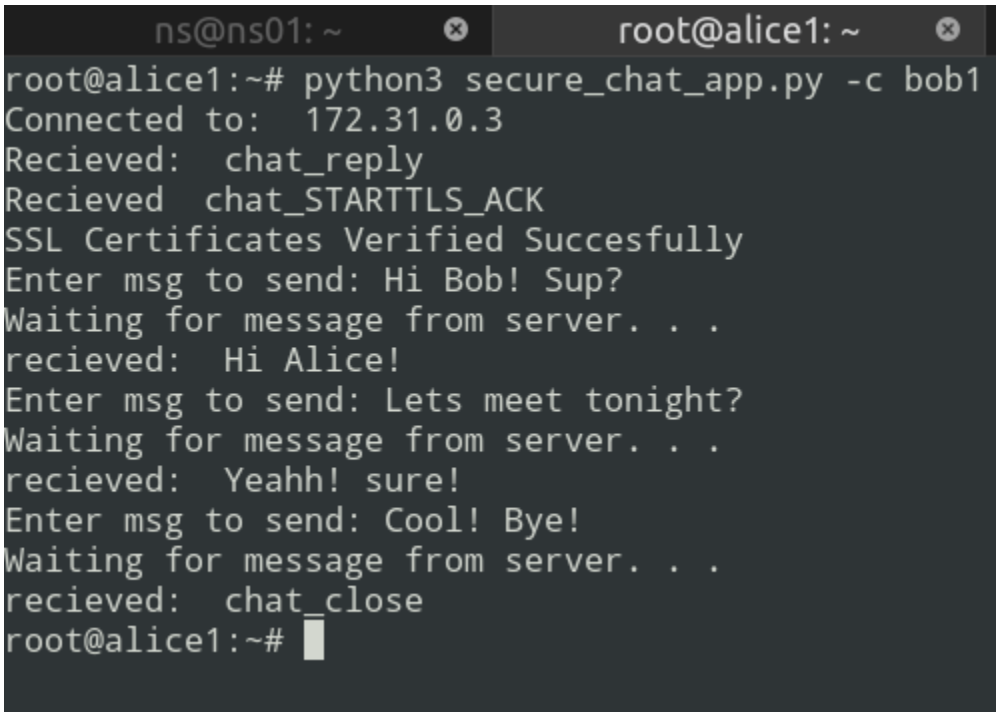
Task - 2 Secure Chat App

Code explanation:

- Code is organized into two classes ***Client*** & ***Server***.
- **Client**:
    - ***create_connection***: Create a TCP/IP socket and accept connections from clients and returns clientSocket
    - ***tls_client***: Function to perform TLS handshake with server verifies certificates, sends and receives messages from server
- **Server**:
    - ***create_and_accept_connections***: Creates a TCP/IP socket and accepts connections from clients.
    - ***tls_server***: Performs TLS handshake with client
- More info of the functions/modules used are documented in the code itself

Sample Chat

Client side:



Server Side:

```
ns@ns01: ~                    root@alice1: ~
root@bob1:~# python3 secure_chat_app.py -s
Waiting for clients to join
Client connected from:  ('172.31.0.2', 46688)
Waiting for client message . . .
received >> chat_hello
Waiting for client message . . .
received >> chat_STARTTLS
Secure TLS 1.3 pipe Established
Waiting for client message . . .
received >> Hi Bob! Sup?
Enter message to send: Hi Alice!
Waiting for client message . . .
received >> Lets meet tonight?
Enter message to send: Yeahh! sure!
Waiting for client message . . .
received >> Cool! Bye!
Enter message to send: chat_close
root@bob1:~#
```

PCAP Verification:

1) As we can see the order of packets, initially **TCP messages** were sent, later **TLS 1.3** is established for sending application data

9	0.000681	172.31.0.3	172.31.0.2	TCP	66	6174 → 46466 [ACK] Seq=11 Ack=24 Win=65152 L
10	0.000704	172.31.0.3	172.31.0.2	TCP	83	6174 → 46466 [PSH, ACK] Seq=11 Ack=24 Win=65
11	0.001786	172.31.0.2	172.31.0.3	TCP	66	46466 → 6174 [ACK] Seq=24 Ack=28 Win=64256 L
12	0.007516	172.31.0.2	172.31.0.3	TLSv1.3	583	Client Hello
13	0.007540	172.31.0.3	172.31.0.2	TCP	66	6174 → 46466 [ACK] Seq=28 Ack=541 Win=64640
14	0.010091	172.31.0.3	172.31.0.2	TLSv1.3	2262	Server Hello, Change Cipher Spec, Applicatio
15	0.010114	172.31.0.2	172.31.0.3	TCP	66	46466 → 6174 [ACK] Seq=541 Ack=2224 Win=6387
16	0.017134	172.31.0.2	172.31.0.3	TLSv1.3	2041	Change Cipher Spec, Application Data, Applic
17	0.017160	172.31.0.3	172.31.0.2	TCP	66	6174 → 46466 [ACK] Seq=2224 Ack=2516 Win=640

2) In frame/packet 4, we can see **chat_hello** message and similarly in frame 6, we can see **chat_reply** message(both un-encrypted)

```
▾ Data (10 bytes)
    Data: 636861745f68656c6c6f
    [Length: 10]
```

```
0000  00 16 3e 89 0d 45 00 16  3e d0 af c8 08 00 45 00   ··>··E·· >·····E·
0010  00 3e 40 ec 40 00 40 06  a1 8a ac 1f 00 02 ac 1f   ·>@·@·@· ········
0020  00 03 b5 82 18 1e a7 d0  8f 02 3b 09 c7 0b 80 18   ········ ··;·····
0030  01 f6 58 74 00 00 01 01  08 0a ff 9f a7 d1 30 3e   ··Xt···· ······0>
0040  38 c8 63 68 61 74 5f 68  65 6c 6c 6f              8·chat_h ello
```

3) In frame 8 we can see **chat_STARTTLS** and similarly in frame 10 we can see **chat_STARTTLS_ACK**(both un-encrypted)

```
▾ Data (13 bytes)
    Data: 636861745f5354415254544c53
    [Length: 13]
```

```
0000  00 16 3e 89 0d 45 00 16  3e d0 af c8 08 00 45 00   ··>··E·· >·····E·
0010  00 41 40 ee 40 00 40 06  a1 85 ac 1f 00 02 ac 1f   ·A@·@·@· ········
0020  00 03 b5 82 18 1e a7 d0  8f 0c 3b 09 c7 15 80 18   ········ ··;·····
0030  01 f6 58 77 00 00 01 01  08 0a ff 9f a7 d1 30 3e   ··Xw···· ······0>
0040  38 c8 63 68 61 74 5f 53  54 41 52 54 54 4c 53      8·chat_S TARTTLS
```

4) TLS1.3 messages can be observed after chat_STARTTLS message

a) Client hello, Server hello, change cipher spec(from frame 12)

```
TLSv1.3    583 Client Hello
TCP         66 6174 → 46466 [ACK] Seq=28 Ack=541 Win=64640 Len=0 TS
TLSv1.3   2262 Server Hello, Change Cipher Spec, Application Data,
TCP         66 46466 → 6174 [ACK] Seq=541 Ack=2224 Win=63872 Len=0
TLSv1.3   2041 Change Cipher Spec, Application Data, Application Da
TCP         66 6174 → 46466 [ACK] Seq=2224 Ack=2516 Win=64000 Len=0
TLSv1.3   1137 Application Data
TCP         66 46466 → 6174 [ACK] Seq=2516 Ack=3295 Win=64128 Len=0
TLSv1.3   1137 Application Data
TCP         66 46466 → 6174 [ACK] Seq=2516 Ack=4366 Win=63872 Len=0
TLSv1.3     90 Application Data
```

b) All application data is also encrypted

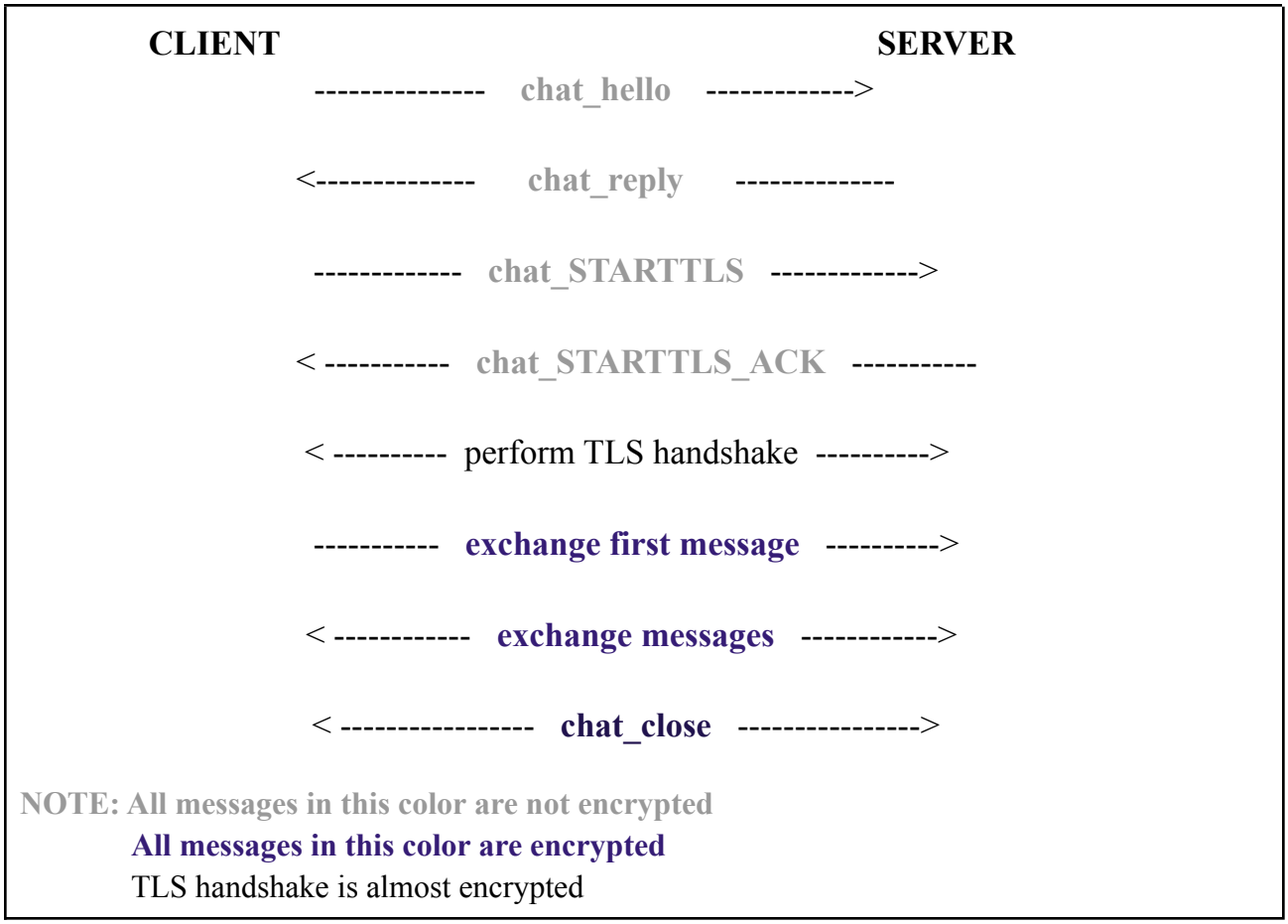
```
▸ Frame 18: 1137 bytes on wire (9096 bits), 1137 bytes captured (9096 bits)
▸ Ethernet II, Src: Xensourc_89:0d:45 (00:16:3e:89:0d:45), Dst: Xensourc_d0:af:c8 (00:16:3e:d0:af:c8)
▸ Internet Protocol Version 4, Src: 172.31.0.3 (172.31.0.3), Dst: 172.31.0.2 (172.31.0.2)
▸ Transmission Control Protocol, Src Port: 6174, Dst Port: 46466, Seq: 2224, Ack: 2516, Len: 1071
▾ Transport Layer Security
  ▾ TLSv1.3 Record Layer: Application Data Protocol: Application Data
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 1066
      Encrypted Application Data: f6d8560e635c9fbc4ac6545c9506d2b8b639d0bebaa4d7417ef751f860cb39f5bc1d631a…
```

5) Most of the handshake is also encrypted (including certificate info)

Wireshark · Packet 14 · task2.pcap

```
▸ Frame 14: 2262 bytes on wire (18096 bits), 2262 bytes captured (18096 bits)
▸ Ethernet II, Src: Xensourc_89:0d:45 (00:16:3e:89:0d:45), Dst: Xensourc_d0:af:c8 (00:16
▸ Internet Protocol Version 4, Src: 172.31.0.3 (172.31.0.3), Dst: 172.31.0.2 (172.31.0.2
▸ Transmission Control Protocol, Src Port: 6174, Dst Port: 46466, Seq: 28, Ack: 541, Len
▾ Transport Layer Security
  ▸ TLSv1.3 Record Layer: Handshake Protocol: Server Hello
  ▸ TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  ▾ TLSv1.3 Record Layer: Application Data Protocol: Application Data
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 23
      Encrypted Application Data: ad35b4871eb466edade2716cc2e7ca7e6f34d5e3164d6a
  ▸ TLSv1.3 Record Layer: Application Data Protocol: Application Data
  ▸ TLSv1.3 Record Layer: Application Data Protocol: Application Data
  ▸ TLSv1.3 Record Layer: Application Data Protocol: Application Data
```

CHAT PROTOCOL - Secure Chat Application



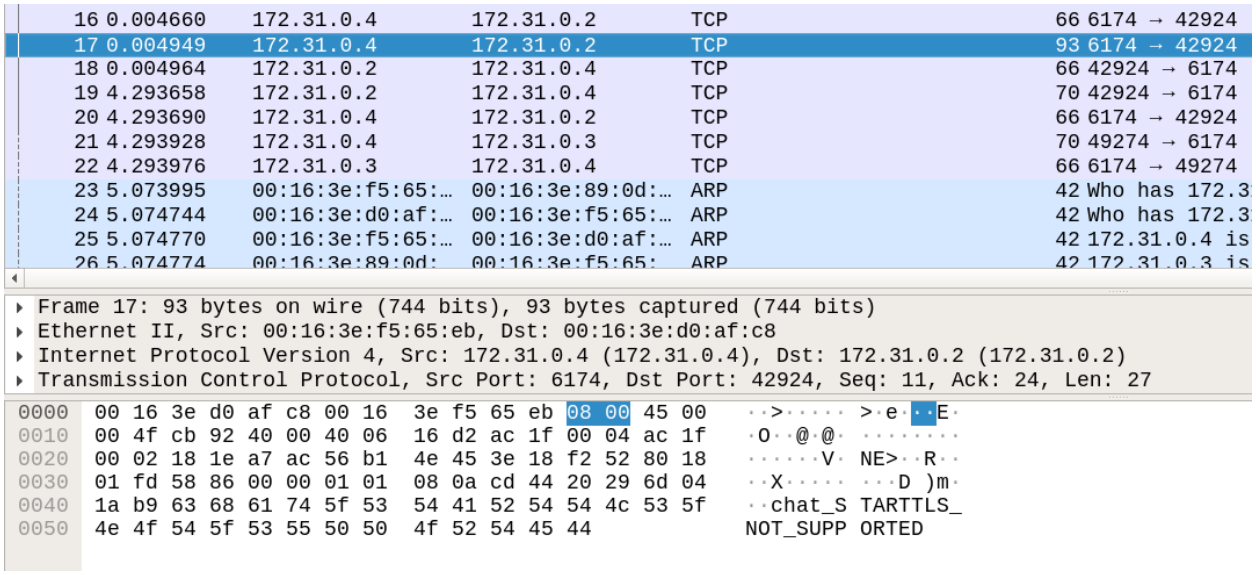
TASK - 3 - Downgrade Attack

Code Explanation:

- *Fake connections:*
  We create fake sockets for Trudy so that it can intercept messages between Alice and Bob. Trudy acts as a fake server while interacting with Alice and as fake client while interacting with Bob. This is implemented in the fake_connection**()** function. This function returns a fake client and fakeserver socket along with client socket.
- *Forwarding messages:* Once the fake connections are made, Trudy will be able to forward the data from Alice to Bob.This is implemented in **forwarding()** function. This function exits when it gets a 'chat_close' message.
- *Downgrade attack :* When client Alice sends a "chat_STARTTLS" message, Trudy sends a TLS not supported message and thus leads to insecure communication.
- More info of the functions/modules used are documented in the code itself

PCAP verification

1. From frame 17, we can see that Trudy (172.31.0.4) sends Alice (172.31.0.2) ***chat_STARTTLS_NOT_SUPPORTED*** instead of forwarding chat_STARTTLS to Bob.



2. From frame 27 and 29, we can see that application data (here 'hi bob') is not encrypted, and we can also see that in frame 27, the message goes from Alice to Trudy (172.31.0.2 ->172.31.0.4) and then Trudy forwards that message to Bob in frame 29 (172.31.0.4 ->172.31.0.3)

```
▸ Frame 27: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
▸ Ethernet II, Src: 00:16:3e:89:0d:45, Dst: 00:16:3e:f5:65:eb
▸ Internet Protocol Version 4, Src: 172.31.0.3 (172.31.0.3), Dst: 172.31.0.4 (172.31.0.4)
▸ Transmission Control Protocol, Src Port: 6174, Dst Port: 49274, Seq: 11, Ack: 15, Len: 12

0000  00 16 3e f5 65 eb 00 16  3e 89 0d 45 08 00 45 00   ··>·e··· >··E··E·
0010  00 40 54 8b 40 00 40 06  8d e7 ac 1f 00 03 ac 1f   ·@T·@·@· ········
0020  00 04 18 1e c0 7a c2 f8  dc 82 f8 1a 84 7a 80 18   ·····z·· ·····z··
0030  01 fd 58 78 00 00 01 01  08 0a c1 bd 6c 2c 96 21   ··Xx···· ····l,·!
0040  9c 85 48 6f 77 20 61 72  65 20 79 6f 75 3f         ··How ar e you?
```

CHAT PROTOCOL- Downgrade Attack

CLIENT	***TRUDY***	***SERVER***
---------- chat_hello ---------->		---------- chat_hello ---------->
<---------- chat_reply ----------		< ---------- chat_reply ----------
---------- chat_hello ---------->		---------- chat_hello ---------->
------- chat_STARTTLS ------>		------ chat_STARTTLS ------>

chat_STARTTLS_NOT_SUPPORTED↵↵

<---------- Insecure communication ---------->

-------- first_client_msg --------> -------- first_client_msg -------->

receives

<---------- > captures <---------->

and forwards

Messages

if received msg

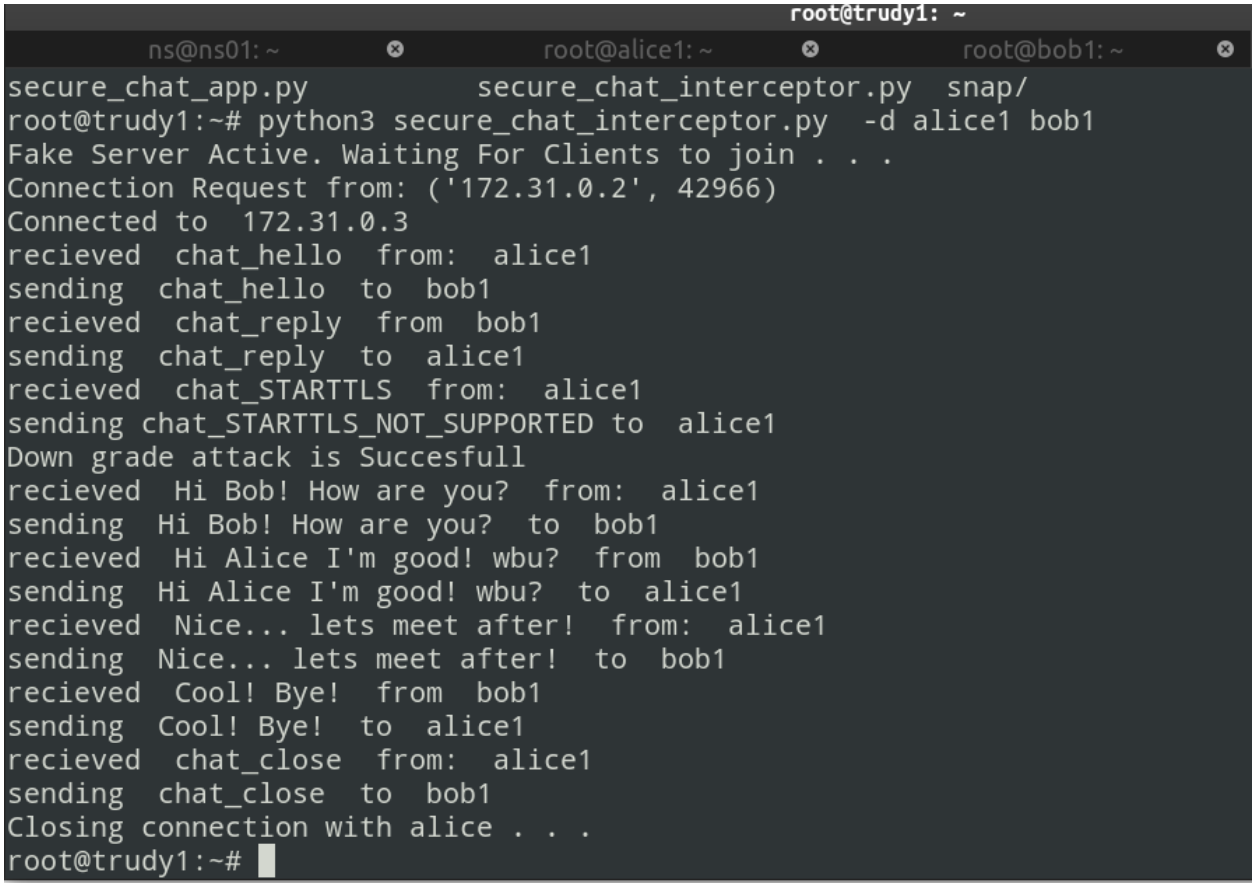
< ---------- > 'chat_close' <----------

connection

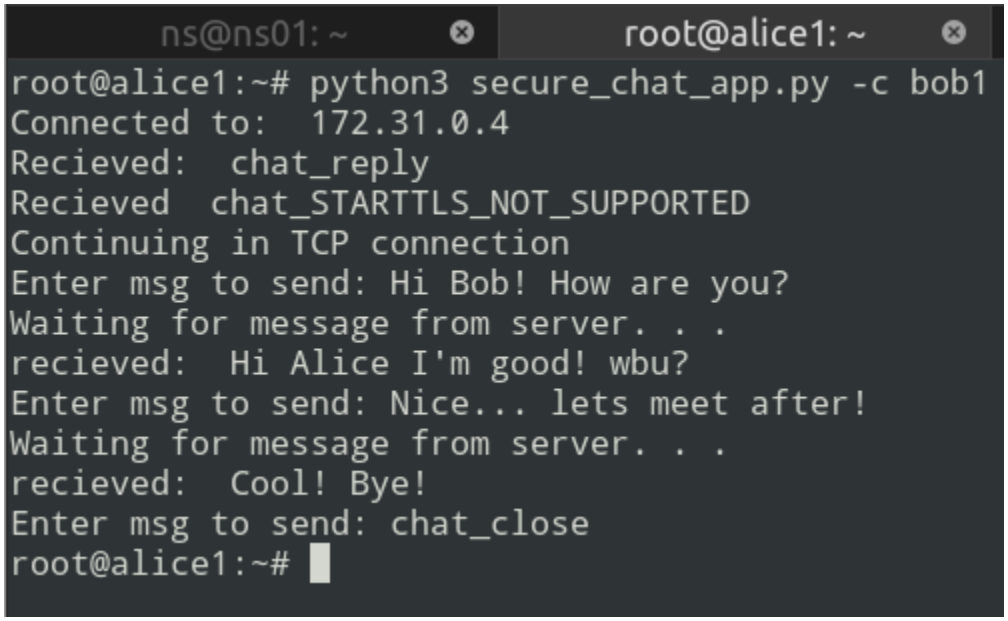
terminates

Sample chat:

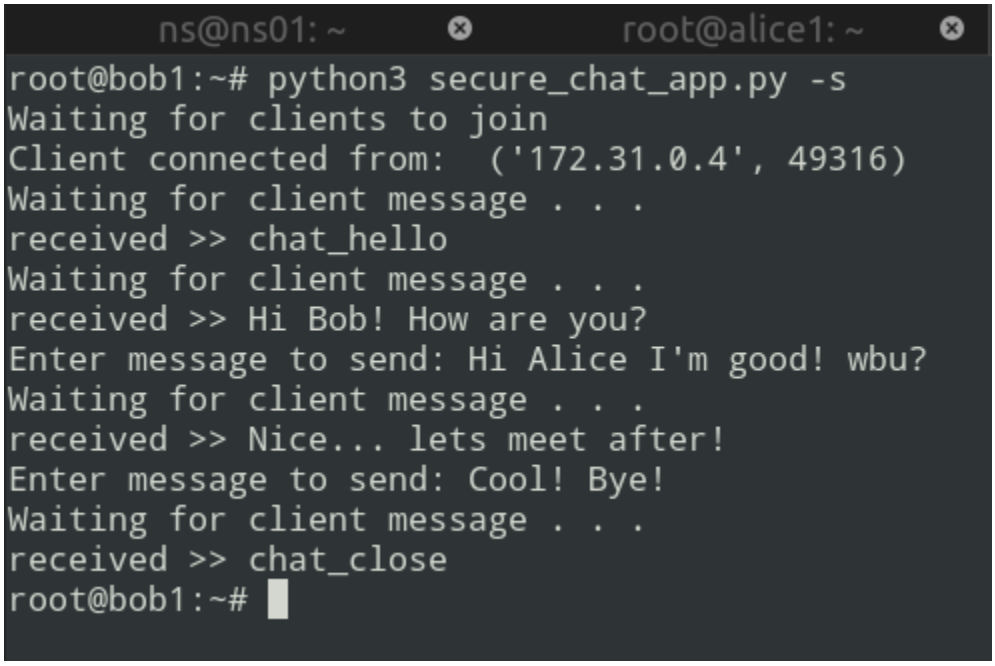
Trudy (Downgrade)



Alice (Client)



Bob (server)

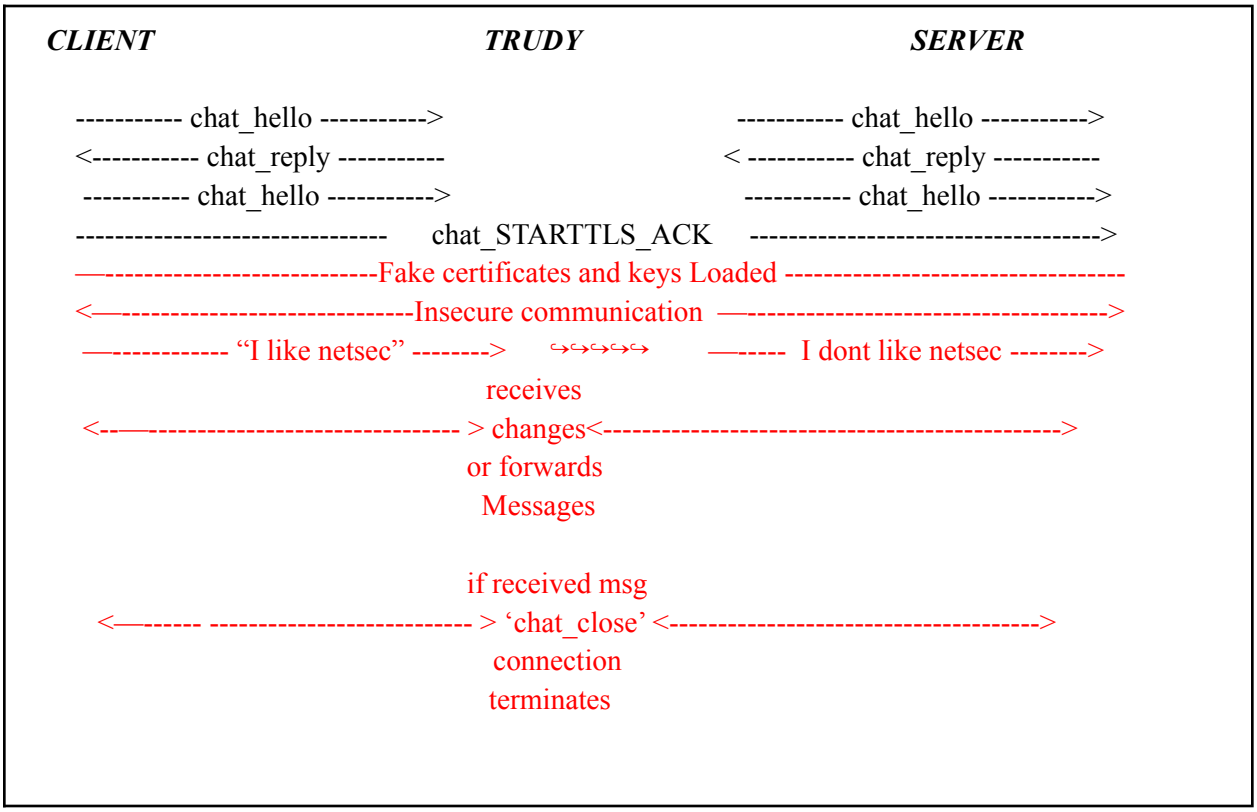


TASK - 4 Man In the Middle Attacks

Code Explanation:

- *Fake Connections* :
  We create fake sockets for Trudy so that it can intercept messages between Alice and Bob. Trudy acts as a fake server while interacting with Alice and as fake client while interacting with Bob. This is implemented in the ***fake_connection*()** function. This function returns a fake client and fakeserver socket along with client socket.
- *Fake Certificates:*
  We created fake CSRs of Bob and Alice. Trudy enters the Root Certificate Authority illegally and signs the fake CSRs. Since root CAs certificate is in trust stores of Alice and Bob, the illegally signed certificates are still valid. Hence, even if Alice and Bob are trying to establish TLS secure connection, Trudy will be able to intercept and tamper the messages.
- *Changing messages:*
  Trudy can choose to change the messages between Alice and Bob or just eavesdrop the chat. This is implemented in the ***change_message*()** function.
- More info of the functions/modules used are documented in the code itself
- *Loading keys and certificates:*
  load_keys_and_cert() function loads required keys and certificates.

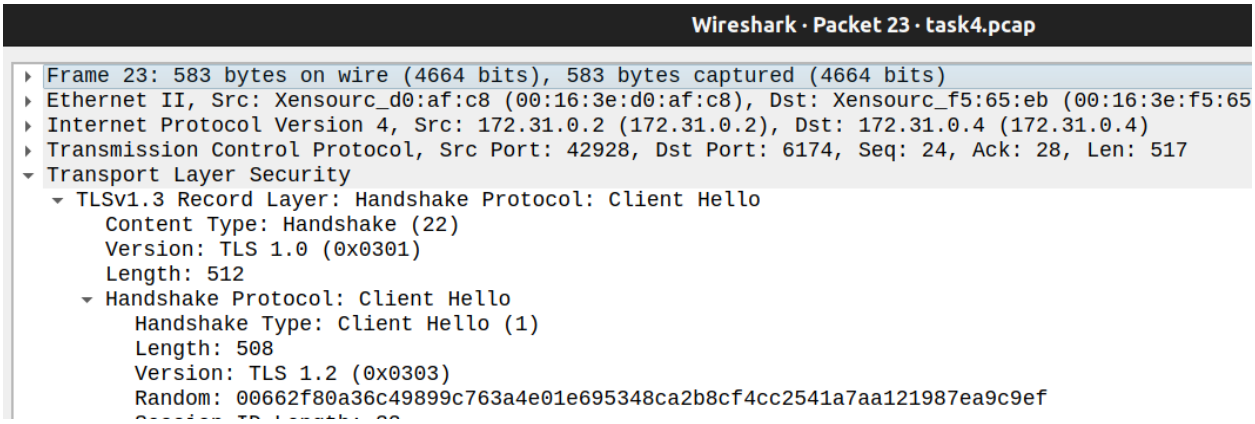
CHAT PROTOCOL - Man In the Middle Attacks



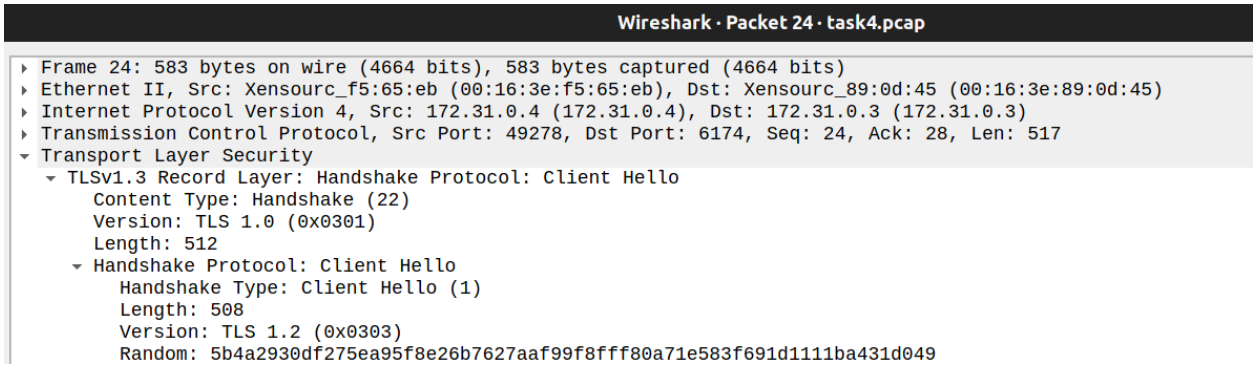
PCAP Verification:

We can see the client hello packet between

1) alice(client) and trudy(server) (172.31.0.2 ->172.31.0.4)

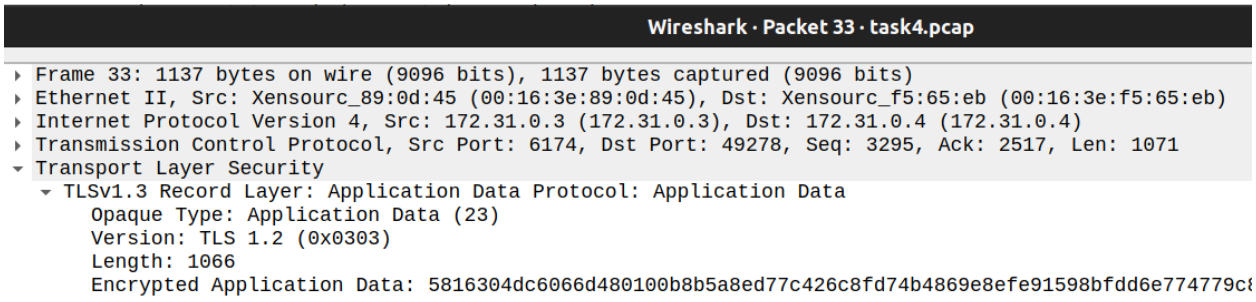


2) trudy(client) and bob(server) (172.31.0.4 -> 172.31.0.3)



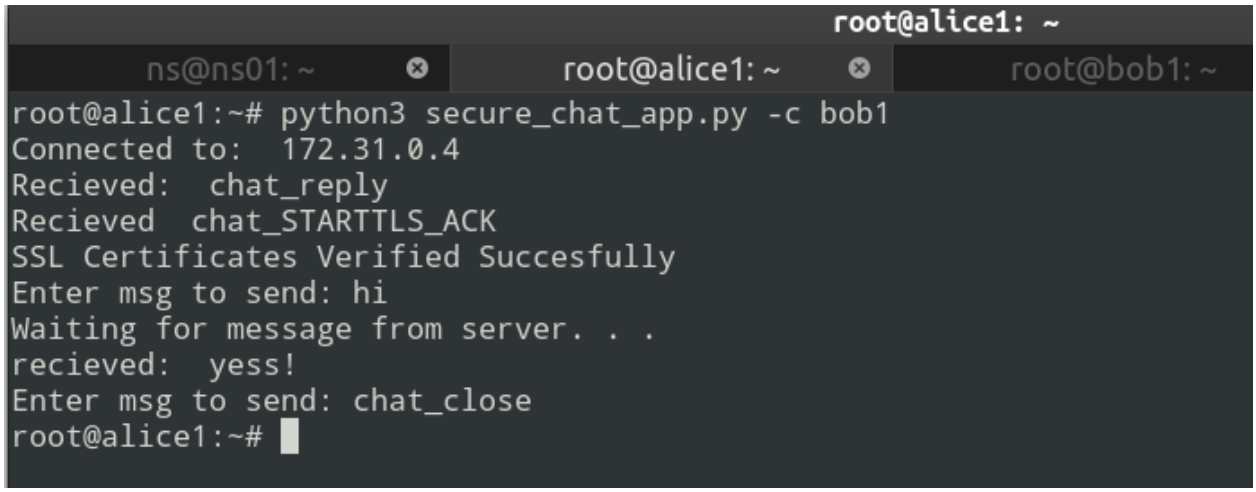
This proves that there are **two TLS pipes** established.

We can see that application data is encrypted totally

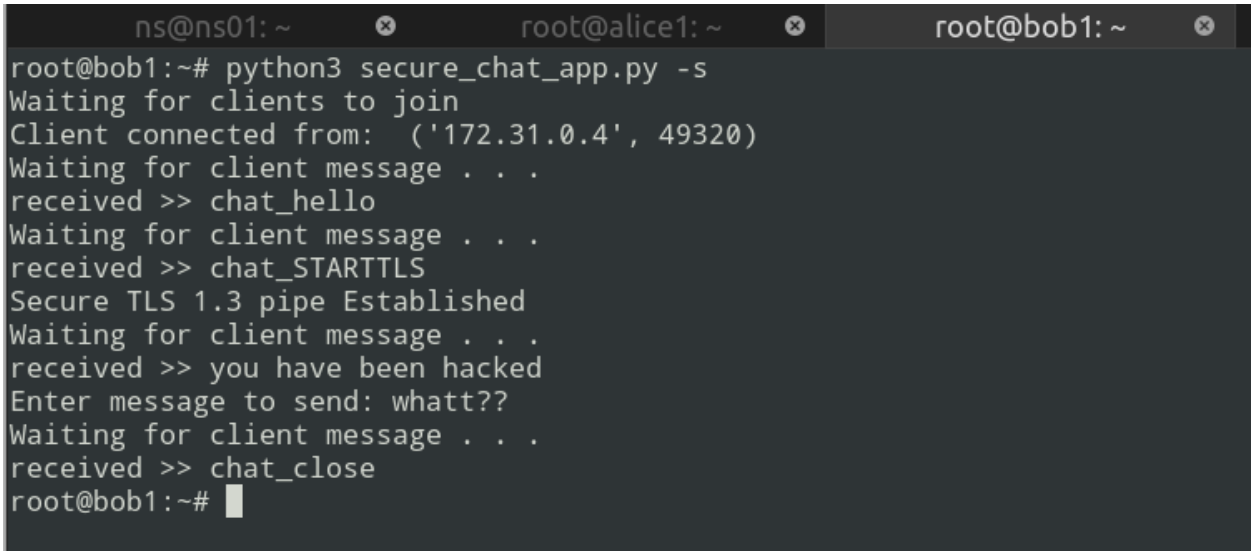


Sample Chat:

Alice (Client)

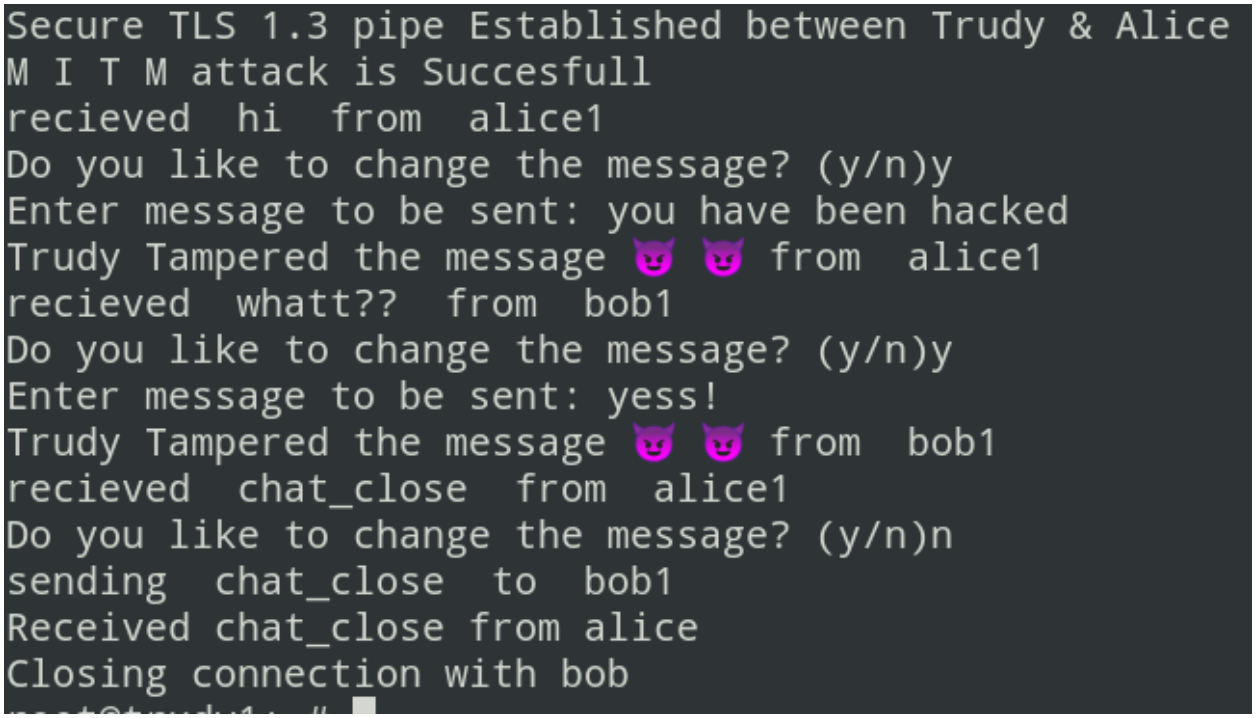


Bob (Server)


ns@ns01: ~
root@alice1: ~
root@bob1: ~
root@bob1:~# python3 secure_chat_app.py -s
Waiting for clients to join
Client connected from: ('172.31.0.4', 49320)
Waiting for client message . . .
received >> chat_hello
Waiting for client message . . .
received >> chat_STARTTLS
Secure TLS 1.3 pipe Established
Waiting for client message . . .
received >> you have been hacked
Enter message to send: whatt??
Waiting for client message . . .
received >> chat_close
root@bob1:~#

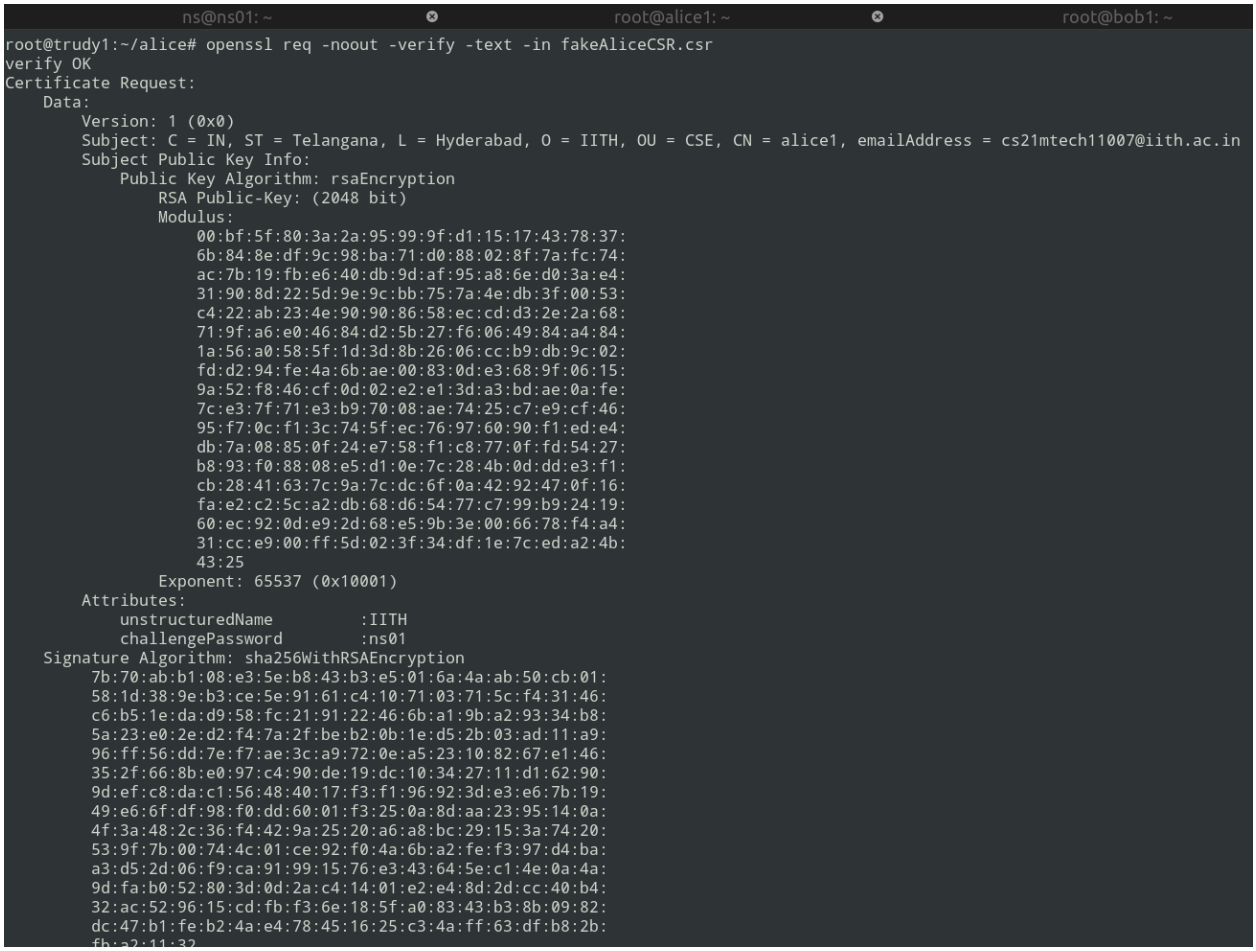
Trudy (MITM)

```
root@trudy1:~# python3 secure_chat_interceptor.py  -d alice1 bob1
Fake Server Active. Waiting For Clients to join . . .
Connection Request from: ('172.31.0.2', 42966)
Connected to  172.31.0.3
recieved  chat_hello  from:  alice1
sending  chat_hello  to  bob1
recieved  chat_reply  from  bob1
sending  chat_reply  to  alice1
recieved  chat_STARTTLS  from:  alice1
sending chat_STARTTLS_NOT_SUPPORTED to  alice1
Down grade attack is Succesfull
recieved  Hi Bob! How are you?  from:  alice1
sending  Hi Bob! How are you?  to  bob1
recieved  Hi Alice I'm good! wbu?  from  bob1
sending  Hi Alice I'm good! wbu?  to  alice1
recieved  Nice... lets meet after!  from:  alice1
sending  Nice... lets meet after!  to  bob1
recieved  Cool! Bye!  from  bob1
sending  Cool! Bye!  to  alice1
recieved  chat_close  from:  alice1
sending  chat_close  to  bob1
Closing connection with alice . . .
root@trudy1:~#
root@trudy1:~# python3 secure_chat_interceptor.py  -m alice1 bob1
Fake Server Active. Waiting For Clients to join . . .
Connection Request from: ('172.31.0.2', 42970)
Connected to  172.31.0.3
recieved  chat_hello  from  alice1
Do you like to change the message? (y/n)n
sending  chat_hello  to  bob1
recieved  chat_reply  from  bob1
Do you like to change the message? (y/n)n
sending  chat_reply  to  alice1
recieved  chat_STARTTLS  from  alice1
Do you like to change the message? (y/n)n
sending  chat_STARTTLS  to  bob1
recieved  chat_STARTTLS_ACK  from  bob1
Do you like to change the message? (y/n)n
sending  chat_STARTTLS_ACK  to  alice1
SSL Certificates Verified Succesfully
Secure TLS 1.3 pipe is established between Bob & Trudy
```



Fake certificates for Task-4

Displaying csr of fakeAlice

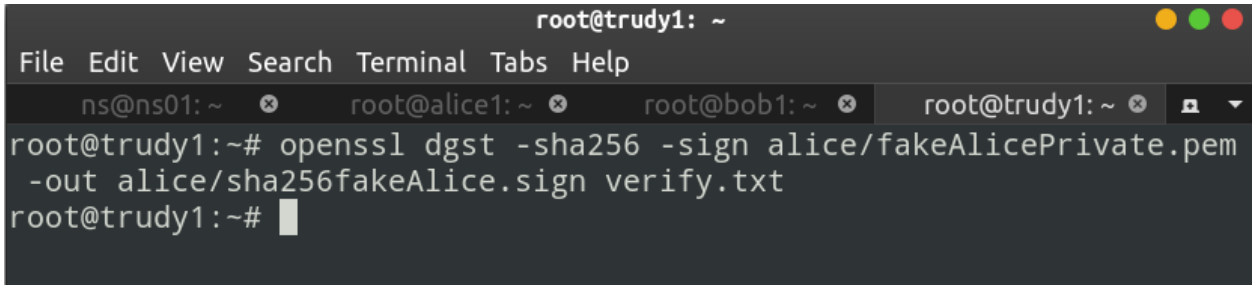


Displaying fakeBobCSR.csr

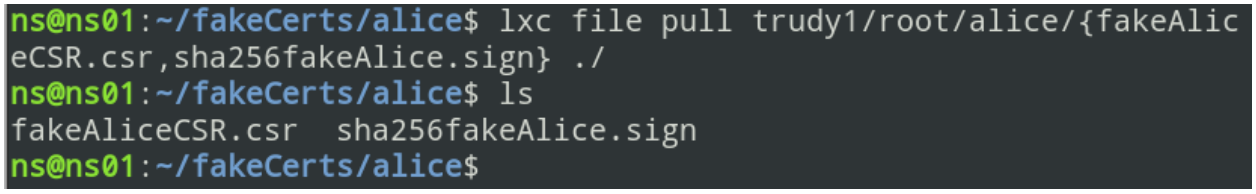
```
root@trudy1:~/bob# openssl req -noout -verify -text -in fakeBobCSR.csr
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Hyderabad, O = IITH, OU = CSE, CN = bob1, emailAddress = cs21mtech11007@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c2:5c:1d:1a:af:75:53:5a:0d:1c:44:3c:7f:c2:
                    6b:50:ec:f2:3f:20:d4:91:85:fb:d5:3b:90:30:ce:
                    d7:08:b9:c2:85:7c:2c:55:33:87:4e:a7:e8:7d:5e:
                    d7:33:fa:95:b6:85:5c:a6:a1:57:7a:36:fb:19:6f:
                    e4:01:89:41:e1:1f:88:67:9b:b6:da:f4:f3:06:1b:
                    23:b9:cb:ab:ab:fa:45:fd:76:05:23:b7:44:29:43:
                    62:bd:6f:9b:14:01:3d:22:34:2d:90:10:37:3a:d9:
                    2c:94:f4:ab:ec:2d:57:b8:87:1d:08:62:7f:39:0e:
                    66:97:95:6a:dc:7d:08:5d:7a:a4:54:60:28:aa:90:
                    40:c0:93:ec:1d:0f:a3:d9:dd:49:5d:5d:13:15:59:
                    2a:9c:93:42:21:c0:56:b8:c6:ae:33:27:53:81:33:
                    c3:82:7f:f7:4b:af:d3:4b:bf:ad:c9:10:bb:6e:f5:
                    f1:3f:2e:c7:5d:cb:45:a1:21:29:89:6d:cc:f4:68:
                    86:7d:a3:9f:4f:6d:c2:3a:64:6f:25:62:8f:a3:84:
                    4e:9a:ac:69:31:aa:bb:f3:b1:6b:de:ff:f3:48:e8:
                    79:a4:7e:32:03:25:0c:05:12:4f:2e:17:6a:83:d2:
                    c7:44:3e:e6:f5:09:94:f8:b7:36:f9:87:f4:d6:3a:
                    5f:7d
                Exponent: 65537 (0x10001)
        Attributes:
            unstructuredName         :IITH
            challengePassword        :ns01
    Signature Algorithm: sha256WithRSAEncryption
         ad:44:94:3f:c0:22:7f:c0:d8:c9:86:9a:09:b6:3c:92:84:31:
         7c:93:d5:03:16:c8:47:ce:53:9f:8d:dc:28:d5:8f:53:d0:ed:
         a4:24:6e:ba:ef:55:57:e4:90:d2:f7:3c:4d:ce:48:c2:1e:e6:
         23:aa:55:5a:72:60:9e:7e:9a:f6:e2:3e:ec:9d:d7:c0:5f:a9:
         24:1b:76:5f:13:08:5d:f1:4b:29:9c:9d:9d:3c:e5:b1:5d:e7:
         f8:e6:5c:67:84:2b:e7:8a:6a:07:30:3a:ad:a3:b7:5c:b4:dd:
         17:e2:77:a0:f2:b5:e7:06:44:20:e1:5b:be:f0:d2:50:e5:74:
         cd:15:5e:5d:d5:86:a7:12:fc:b3:e9:9f:c7:68:e2:cb:7d:cc:
         8d:d7:45:0e:02:cf:93:20:42:cb:e9:2b:81:4c:b5:f2:c4:b3:
         4b:84:81:00:3e:c9:32:ac:37:0e:8e:14:6c:a2:b1:be:19:48:
         b5:e1:2a:00:7a:20:b2:e0:d5:6a:ab:57:f9:ab:c7:e7:26:96:
         89:88:f9:ef:f7:45:3b:70:de:ed:db:42:c4:0f:00:98:35:ab:
         2f:d3:ac:ed:fb:c9:d0:57:4c:bf:13:69:45:ce:4f:fe:5b:de:
         d0:76:b4:2c:7f:6c:e2:7d:ae:64:41:74:63:cf:46:bf:86:f0:
         a9:62:de:1f
```

fakeAlice:

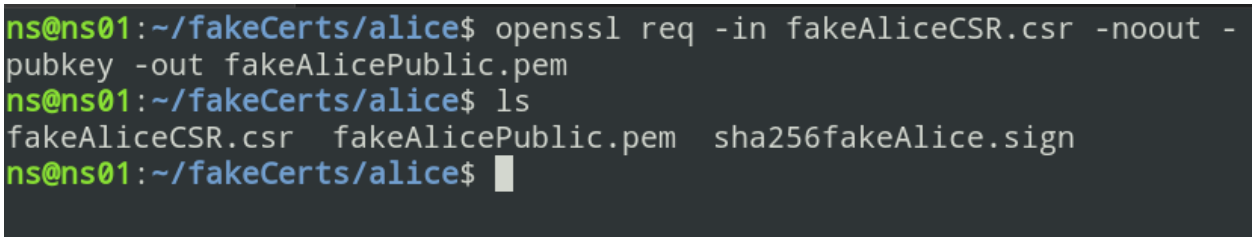
Fake alice creating digest and signing msg with its Private key:



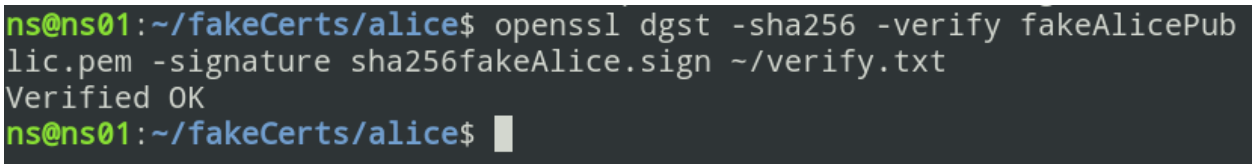
Sending Alice's Fake CSR and signed digest to VM :



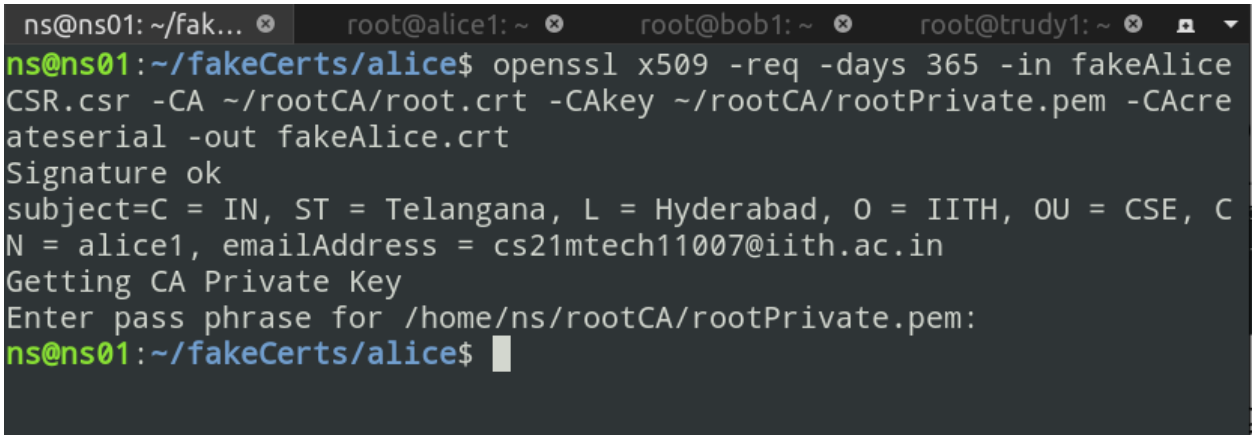
Root CA Extracting Alice's fake public key from CSR:



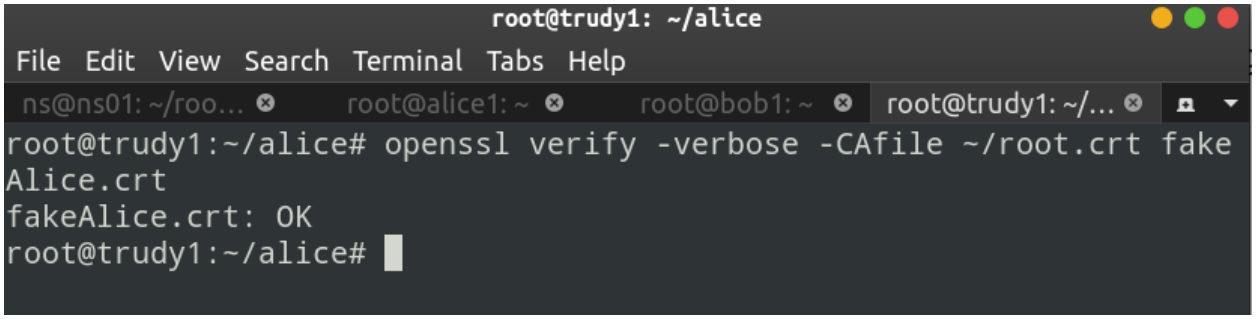
Verifying Fake Alice's Public key :



Root CA signs Fake Alice's certificate:

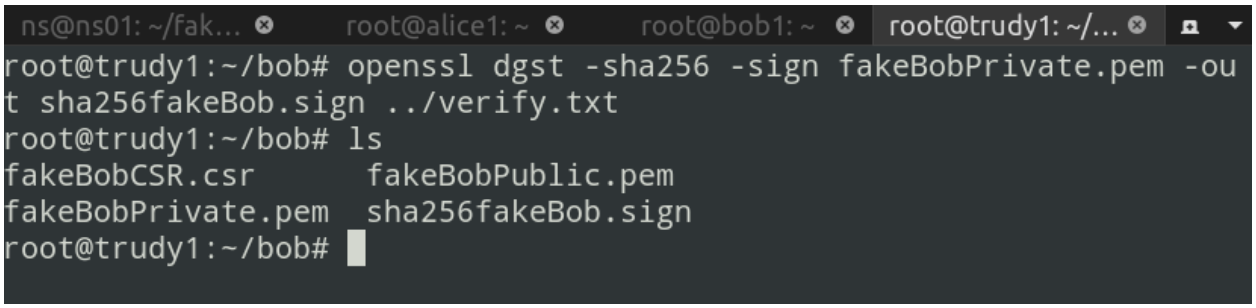


Fake Alice verifying that the certificate is indeed signed by root CA:

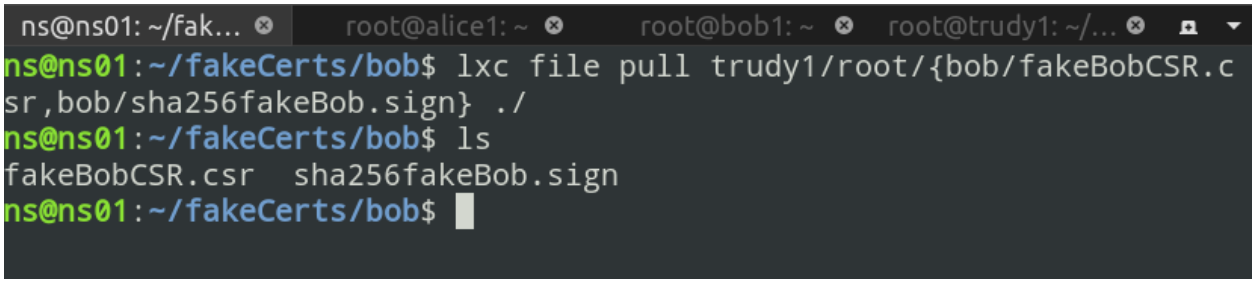


fakeBob:

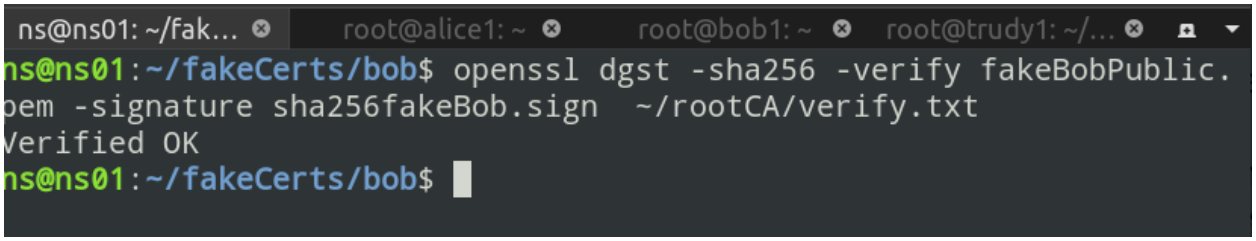
Fake Bob creating digest and signing msg with its Private key:



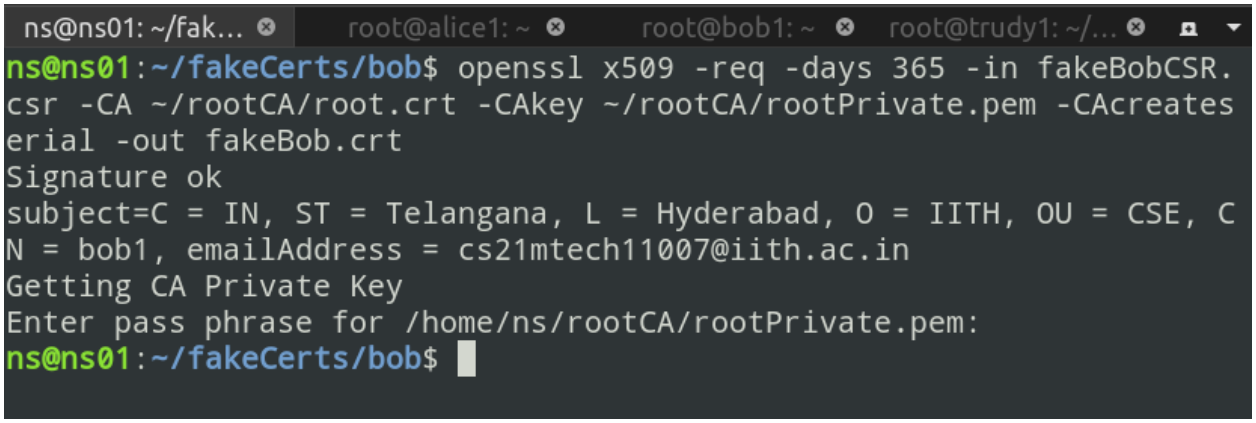
Sending Bob's Fake CSR and signed digest to VM :



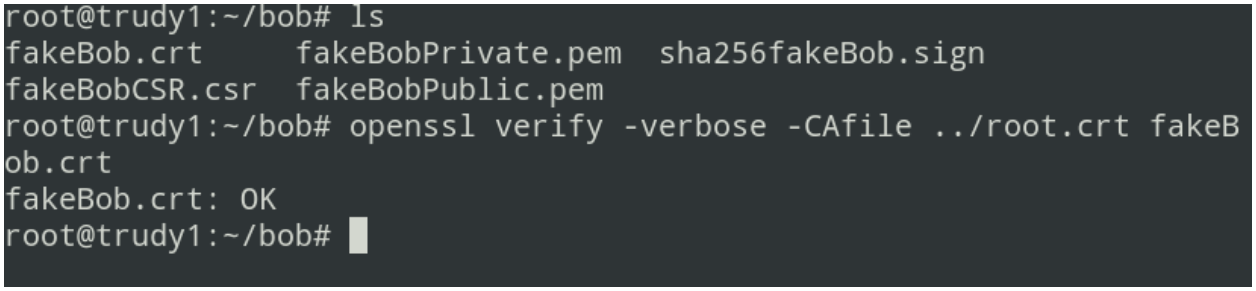
Verifying fake Bob's certificate:



Root CA signs Fake Bob's certificate:



Fake Bob verifying that the certificate is indeed signed by rootCA:



Credit statement

Task	Akash Tadwai	Sai Varshittha Ponnam	Amit Kumar
1	Generated private, public keys for rootCA, issued Certificates	Generated private, public keys for Alice	Generated private, public keys for Bob
2	Designed and Coded Client and Server classes and secure communication modules.	Wrote client and server classes	Helped in designing OOP classes.
3	Debugging & code documentation	Debugging Errors while running on Containers	Written code for Downgrade attack.
4	Generated fake certificates for Trudy	Written module for Man in the middle attack.	Generated fake certificates for Trudy
Report	Proofreading & Task 1,4	Written docs for Task 1,2,3	Written docs for Task1
Readme	Written README.md	Written README.md	-
Makefile	Wrote complete Makefile	-	-

REFERENCES:

REFERENCES used for TASK - 1:

- How to Fix SSH Failed Permission Denied
- How to export public key from Certificate Signing Request?
- LXD - Getting started - command line
- Viewing the Contents of a Certificate Signing Request (CSR) with OpenSSL
- Used commands from openssl handson assignment

REFERENCES used for TASK - 2,3,4

- Adding trusted root certificates to the server
- List all available ssl ca certificates - Unix & Linux Stack Exchange
- 9 Python socket.error: [Errno 104] Connection reset by peer
- Multithreading in Python | Set 1
- ssl — TLS/SSL wrapper for socket objects — Python 3.9.2 documentation
- Python [Errno 98] Address already in use
- Python Exception in thread Thread-1 (most likely raised during interpreter shutdown)?
- How do I abort a socket.recv() from another thread in Python

General

- GNU make documentation
- Markdown Cheat Sheet