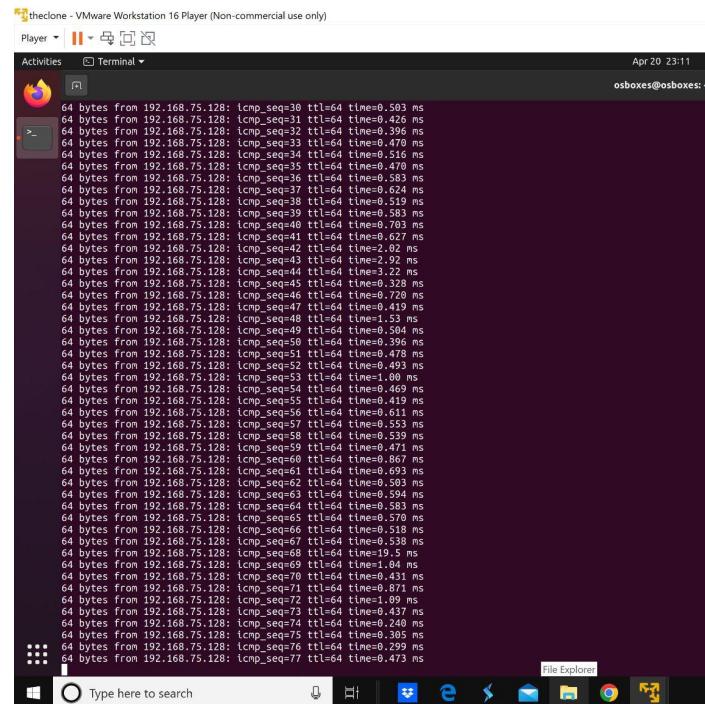


- The first thing I did was create another VM using the image we were given at the start of class. I initially tried to clone it but there wasn't an option for me.
- I then installed wireshark on my server VM using sudo apt update and sudo apt install wireshark
  - I confirmed that my nic was ens33 using ip addr show
  - then I confirmed that the two VMs had different IPs by going to wireshark and typing ICMP in the box and pinged from client to server
  - my original VM has an IP address of 192.168.75.128 while my clone one had 192.168.75.130

- client perspective



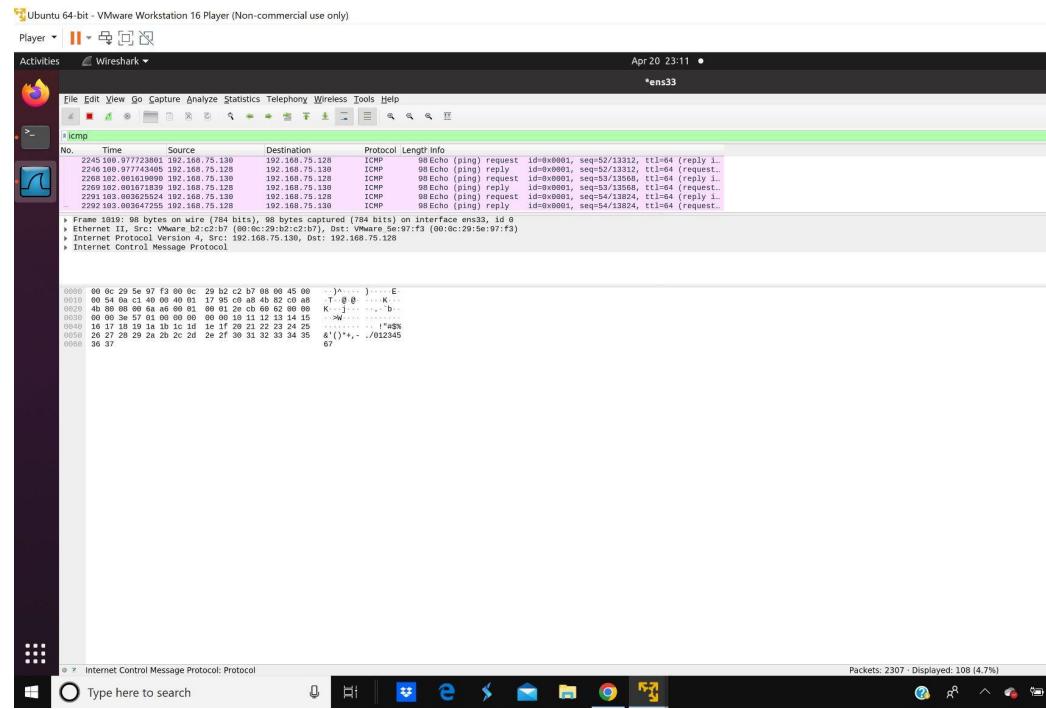
```

theclone - VMware Workstation 16 Player (Non-commercial use only)
Player Terminal Activities Apr 20 23:11
osboxes@osboxes: ~

64 bytes from 192.168.75.128: icmp_seq=30 ttl=64 time=0.593 ms
64 bytes from 192.168.75.128: icmp_seq=31 ttl=64 time=0.426 ms
64 bytes from 192.168.75.128: icmp_seq=32 ttl=64 time=0.595 ms
64 bytes from 192.168.75.128: icmp_seq=33 ttl=64 time=0.478 ms
64 bytes from 192.168.75.128: icmp_seq=34 ttl=64 time=0.516 ms
64 bytes from 192.168.75.128: icmp_seq=35 ttl=64 time=0.478 ms
64 bytes from 192.168.75.128: icmp_seq=36 ttl=64 time=0.583 ms
64 bytes from 192.168.75.128: icmp_seq=37 ttl=64 time=0.583 ms
64 bytes from 192.168.75.128: icmp_seq=38 ttl=64 time=0.519 ms
64 bytes from 192.168.75.128: icmp_seq=39 ttl=64 time=0.583 ms
64 bytes from 192.168.75.128: icmp_seq=40 ttl=64 time=0.793 ms
64 bytes from 192.168.75.128: icmp_seq=41 ttl=64 time=0.627 ms
64 bytes from 192.168.75.128: icmp_seq=42 ttl=64 time=2.02 ms
64 bytes from 192.168.75.128: icmp_seq=43 ttl=64 time=2.39 ms
64 bytes from 192.168.75.128: icmp_seq=44 ttl=64 time=0.22 ms
64 bytes from 192.168.75.128: icmp_seq=45 ttl=64 time=0.328 ms
64 bytes from 192.168.75.128: icmp_seq=46 ttl=64 time=0.720 ms
64 bytes from 192.168.75.128: icmp_seq=47 ttl=64 time=0.419 ms
64 bytes from 192.168.75.128: icmp_seq=48 ttl=64 time=1.53 ms
64 bytes from 192.168.75.128: icmp_seq=49 ttl=64 time=0.545 ms
64 bytes from 192.168.75.128: icmp_seq=50 ttl=64 time=0.394 ms
64 bytes from 192.168.75.128: icmp_seq=51 ttl=64 time=0.478 ms
64 bytes from 192.168.75.128: icmp_seq=52 ttl=64 time=0.493 ms
64 bytes from 192.168.75.128: icmp_seq=53 ttl=64 time=1.00 ms
64 bytes from 192.168.75.128: icmp_seq=54 ttl=64 time=0.469 ms
64 bytes from 192.168.75.128: icmp_seq=55 ttl=64 time=0.501 ms
64 bytes from 192.168.75.128: icmp_seq=56 ttl=64 time=0.511 ms
64 bytes from 192.168.75.128: icmp_seq=57 ttl=64 time=0.553 ms
64 bytes from 192.168.75.128: icmp_seq=58 ttl=64 time=0.539 ms
64 bytes from 192.168.75.128: icmp_seq=59 ttl=64 time=0.471 ms
64 bytes from 192.168.75.128: icmp_seq=60 ttl=64 time=0.867 ms
64 bytes from 192.168.75.128: icmp_seq=61 ttl=64 time=0.566 ms
64 bytes from 192.168.75.128: icmp_seq=62 ttl=64 time=0.583 ms
64 bytes from 192.168.75.128: icmp_seq=63 ttl=64 time=0.594 ms
64 bytes from 192.168.75.128: icmp_seq=64 ttl=64 time=0.583 ms
64 bytes from 192.168.75.128: icmp_seq=65 ttl=64 time=0.578 ms
64 bytes from 192.168.75.128: icmp_seq=66 ttl=64 time=0.518 ms
64 bytes from 192.168.75.128: icmp_seq=67 ttl=64 time=0.501 ms
64 bytes from 192.168.75.128: icmp_seq=68 ttl=64 time=0.51 ms
64 bytes from 192.168.75.128: icmp_seq=69 ttl=64 time=1.04 ms
64 bytes from 192.168.75.128: icmp_seq=70 ttl=64 time=0.431 ms
64 bytes from 192.168.75.128: icmp_seq=71 ttl=64 time=0.871 ms
64 bytes from 192.168.75.128: icmp_seq=72 ttl=64 time=1.99 ms
64 bytes from 192.168.75.128: icmp_seq=73 ttl=64 time=0.437 ms
64 bytes from 192.168.75.128: icmp_seq=74 ttl=64 time=0.595 ms
64 bytes from 192.168.75.128: icmp_seq=75 ttl=64 time=0.305 ms
64 bytes from 192.168.75.128: icmp_seq=76 ttl=64 time=0.299 ms
64 bytes from 192.168.75.128: icmp_seq=77 ttl=64 time=0.473 ms

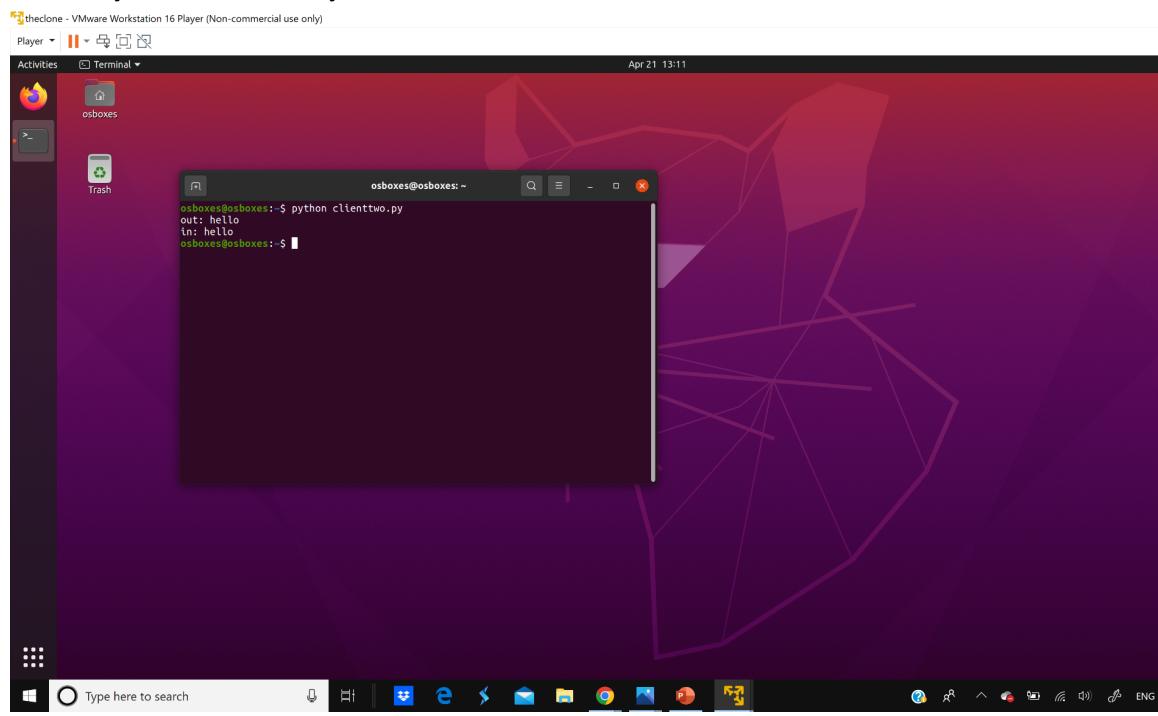
```

- wireshark server perspective

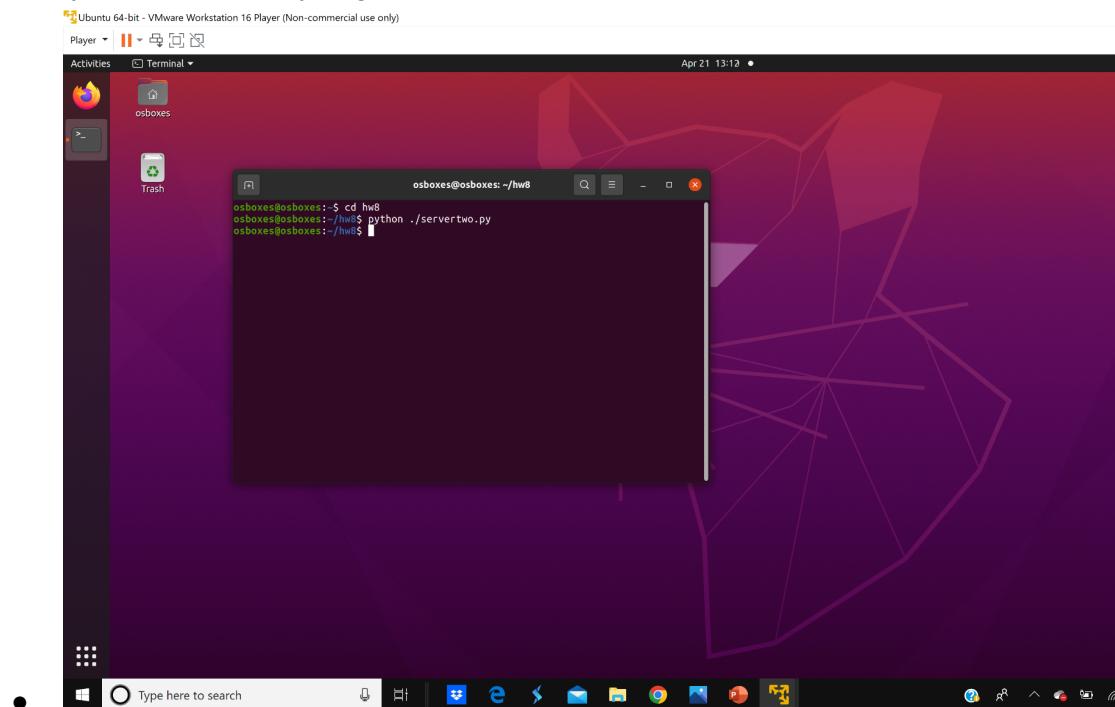


- Next I aimed to create a server.py file on my main VM and a client.py file on my clone VM without ssl at first
  - I used the format of the example code given in the network powerpoint slides and ran into some minor issues like the fact that I had to encode and decode the message for the s.sendall() to work. Or that I had to use the server IP address for the host.
  - Eventually I got the code to work where first you run the server.py on my original VM, then run the client.py on the other. You type your message in the client and then it encodes it and sends it to the server where it receives it and sends it back. Then in the client server it receives it again and decodes it and prints it out

■ this is my client run on my clone VM



■ this is my server run on my original VM



- This is the source code for client and server

```

theclone - VMware Workstation 16 Player (Non-commercial use only)
Player ▾ | Terminal ▾
Activities Terminal Apr 21 13:51
osboxes@osboxes: ~
File Edit Options Buffers Tools Python Help
Import socket
>_
HOST = "192.168.75.128" # The server's hostname or IP address
PORT = 65432 # The port used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Let's implement something very about how my messages get there
s.connect((HOST,PORT))

val = input("out: ")
print("Message is: " + val)

s.sendall(val.encode())
data = s.recv(1024).decode()
print("... " + data)
s.close()

```

```

Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player ▾ | Terminal ▾
Activities Terminal Apr 21 13:52
osboxes@osboxes: ~/hw8
File Edit Options Buffers Tools Python Help
Import socket
>_
HOST = "192.168.75.128"
PORT = 65432 # Port to listen on (non-privileged ports are > 1023)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT)) # specify the "backlog" for this socket
s.listen(1) # accept() will block here until a connection is made
conn, addr = s.accept() # wait and accept the connection
data = conn.recv(1024) # read the content of the message
#print("Server says: " + data);
conn.sendall(data)
conn.close()

```



- Now I decided to create the PKI. I mainly followed the given tutorial and did it on the server side.

- first I did openssl genrsa -des3 -out rootCA.key 4096 to create the root key

```

osboxes@osboxes: $ openssl genrsa -out rootCA.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....+
e: 65337 (0x010001)

```

- then i did openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.crt to create and sign the root certificate

```

osboxes@osboxes: $ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.crt
You are about to be asked to enter information that will be incorporated
into your certificate request:
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Georgia
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:192.168.75.128
Email Address []:
osboxes@osboxes: ~

```

● Type here to search

- for common name I made sure to put the server IP address

- then i did openssl genrsa -out mydomain.com.key 2048 to create the certificate key

```
osboxes@osboxes:~$ openssl genrsa -out mydomain.com.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+=====
e is 65537 (0x10001)
osboxes@osboxes:~$
```

- Then I did the one liner version openssl req -new -sha256 -key mydomain.com.key -subj "/C=US/ST=CA/O=MyOrg, Inc./CN=mydomain.com" -out mydomain.com.csr in order to create the csr
- I did openssl req -in mydomain.com.csr -noout -text to certify the csr content

```
e ls 65537 (0x10001)
osboxes@osboxes:~$ openssl req -new -sha256 -key mydomain.com.key -subj "/C=US/ST=CA/O=MyOrg, Inc./CN=mydomain.com" -out mydomain.com.csr
Certificate Request:
Data:
Version: 1 (0x0)
Subject: C = US, ST = CA, O = "MyOrg, Inc.", CN = mydomain.com
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)
            Modulus:
                90:c6:35:e0:94:b7:cd:86:68:96:ab:4c:f8:5d:05:
                72:84:64:i5:2:6a:c2:3a:0e:e4:ee:00:d:c:89:41:c1:
                52:18:ef:54:29:62:00:c1:8e:33:94:5:f:13:a1:5e:
                54:18:1c:7f:2d:20:80:7a:0e:77:33:53:71:bc:6e:
                54:b4:8d:81:f5:4a:9b:bd:c2:fb:d3:7b:fe:3a:b4:
                b9:c5:62:i6:54:92:7c:33:a6:19:2e:f3:ad:ef:
                d5:3f:66:56:a9:07:48:01:0f:b3:0e:92:b9:96:6f:
                98:67:ad:04:fd:b4:a8:c4:43:3d:60:56:ec:28:85:
                a3:4f:44:5d:39:59:30:59:59:59:59:59:59:59:59:
                e9:09:7b:f4:45:55:49:59:ad:59:59:59:59:59:59:
                47:84:be:0a:2f:3b:87:64:9e:41:ce:87:5a:52:95:
                43:c9:90:64:c1:73:32:eb:d4:6a:25:10:c8:50:8a:
                1d:33:8d:cb:46:9d:7c:41:7e:5d:fb:9e:32:e5:fd:
                cd:de:87:50:5c:a3:16:f2:3e:7c:d9:7d:75:65:ce:
                dc:ee:ac:b7:11:73:ab:d5:5:bf:3c:de:49:c4:57:0a:
                5f:ff:cf:82:51:24:c3:00:5b:6e:54:c4:7d:ad:7e:
                66:bf
            Exponent: 65537 (0x10001)
Attributes:
a0:00
Signature Algorithm: sha256WithRSAEncryption
44:4d:33:72:2d:3c:cc:0b:bb:9a:66:fb:fc:93:0e:f3:a0:11:
3e:6b:T5:07:cb:90:39:e7:6c:dc:a0:2d:22:d1:89:3a:f5:e9:
20:50:65:77:fa:56:65:ab:b8:76:4f:ab:a8:dd:1a:16:4f:1b:49:
95:8e:c6:a6:6c:f1:9a:91:d7:6b:6c:2d:eb:65:67:eb:7e:54:
33:fe:75:59:01:ee:7b:41:96:2d:67:01:41:94:7d:39:a7:
72:2c:4e:63:44:53:49:41:49:49:49:49:49:49:49:49:49:49:
16:ca:33:b4:0f:87:45:4e:9c:a2:47:d2:et:f5:b4:8c:f1:24:
dc:2d:62:fc:17:25:2a:0f:af:a4:6d:bf:7d:13:f:67:7b:ea:2e:
06:f5:77:f1:42:41:36:08:ff:4d:c4:64:a9:1f:4c:4f:ab:06:
ec:0c:76:0b:d9:39:64:e7:75:d2:6e:ae:9e:47:d3:76:6e:dc:
84:26:35:51:7b:33:0b:c8:b2:5f:10:29:ec:51:de:89:6e:bb:16:
09:23:2e:1a:90:90:33:b1:ee:70:4d:1f:45:aa:b1:a1:0d:
c4:sdef:a2
```

- then I made the certificate using mydomain csr and the CA Root key with openssl x509 -req -in mydomain.com.csr -CA rootCA.crt -CAkey rootCA.key -CAcreateserial -out mydomain.com.crt -days 500 -sha256
- Finally I verified the certificates content with openssl x509 -in mydomain.com.crt -text -noout

```

Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player ▾ | └ osboxes@osboxes: ~
Activities Terminal Apr 20 23:25
osboxes@osboxes: ~ openssl req -new -x509 -keyout mydomain.com.key -out mydomain.com.crt -days 500
Signature Algorithm: sha256WithRSAEncryption
Subject: C=US, ST=CA, O=MyOrg, Inc., CN=mydomain.com
Getting CA Private Key
osboxes@osboxes: ~ openssl x509 -in mydomain.com.crt -text -noout
Certificate
    Data:
        Version: 1 (0x8)
        Serial Number:
            67:80:fb:f8:d2:e7:0:8:1e:16:a9:b4:f4:0e:5d:54:b2:c1:a7:d8:87:17:e1
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=Georgia, CN=192.168.75.128
        Validity
            Not Before: Apr 21 03:24:27 2022 GMT
            Not After: May 21 03:24:27 2023 GMT
        Subject: C=US, ST=CA, O=MyOrg, Inc., CN=mydomain.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                    Modulus:
                        00:c6:35:5e:04:b7:c4:86:08:96:ab:4c:f8:5d:05
                        72:2f:4a:99:72:5b:05:80:12:12:7f:05:4b:2d:b6:8c
                        52:18:ef:54:29:62:0b:0c:18:e3:33:94:0f:13:a1:5e:
                        54:18:cf:9b:59:2d:fc:0e:30:5d:09:48:06:3f:43:
                        2c:00:0d:76:76:2e:0d:04:0a:25:10:c8:50:8a:
                        54:b4:8d:81:54:14:9b:0b:27:fb:43:1e:13:b4:
                        b9:c5:62:16:54:92:7c:1c:83:3d:65:19:2e:f3:ad:ef:
                        d5:3f:66:56:a9:07:74:01:0f:53:0e:42:09:96:6f:
                        90:3d:01:01:01:01:01:01:01:01:01:01:01:01:01:
                        a3:4f:4a:99:72:5b:05:80:12:12:7f:05:4b:2d:b6:8c:
                        e0:c0:17:bf:fe:65:1d:9a:ad:de:2:84:3b:f3:72:ee:
                        47:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:
                        43:c9:90:64:c1:17:13:2e:0b:04:0a:25:10:c8:50:8a:
                        1d:33:8d:cb:46:94:7c:41:7e:5d:f8:9e:32:c5:fd:
                        cd:0d:07:58:3e:3d:3c:0d:0d:0d:0d:0d:0d:0d:0d:
                        dc:fe:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:0d:
                        9f:84:ff:82:31:24:c1:3d:05:bce:54:c4:7d:ad:7e:
                        66:bf:
        Expired at: 65371 (0x10801)
        Signature Algorithm: sha256WithRSAEncryption
        b6:3c:24:19:b1:b5:0:57:7a:52:a0:1f:09:86:f3:98:84:6d:fd:
        21:8b:9b:1f:9b:7a:8b:05:37:9b:c9:72:c8:2f:42:dd:9d:33:29:
        86:5f:4b:cf:f5:47:4:3b:1e:8a:0b:08:99:96:e4:9e:76:0b:
        0d:63:13:13:77:5b:0b:10:05:74:1a:0c:09:dc:0c:45:19:85:
        03:3d:3d:3d:3d:3d:3d:3d:3d:3d:3d:3d:3d:3d:3d:3d:
        47:de:9b:26:ea:24:5:ff:16:39:6d:ac:62:cfc:5:bc:62:9e:8a:
        9c:e3:34:3c:16:9:cc:b1:37:ad:ee:56:1c:0b:a1:69:ec:b7:09:
        66:bf:

```

- After doing all of this I hardcopied my rootCA.crt to my client VM
- I now have the PKI set up so now I can go ahead and wrap my socket with SSL
  - for client I constructed my context by
    - context = ssl.SSLContext(ssl.PROTOCOL\_TLS)
    - context.verify\_mode = ssl.CERT\_REQUIRED
    - context.check\_hostname = False
    - context.load\_verify\_locations("Downloads/rootCA.crt")
  - then I wrapped my socket with context.wrap\_socket(s, server\_hostname=192.168.75.128)

```

File Edit Options Buffers Tools Python Help
Activities Terminal Apr 21 13:50
osboxes@osboxes: ~
import socket,ssl
context = ssl.SSLContext(ssl.PROTOCOL_TLS)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = False
context.load_verify_locations("Downloads/rootCA.crt")

HOST = "192.168.75.128" # The server's hostname or IP address
PORT = 65432 # The port used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connstream = context.wrap_socket(s, server_hostname="192.168.75.128")
# Let the Internet worry about how my messages get there
connstream.connect((HOST, PORT))

val = input('out: ')
print("Message is: " + val)

connstream.sendall(val.encode())
data = connstream.recv(1024).decode()
print("In: " + data)
connstream.close()

```

- I really only needed the rootCA.crt for the client side of the PKI
- for server I created my context by
  - context = ssl.create\_default\_context(ssl.Purpose.CLIENT\_AUTH)
  - context.load\_cert\_chain(certfile="mydomain.com.crt", keyfile="mydomain.com.key")
  - I then wrapped my socket around using
    - connstream = context.wrap\_socket(s, server\_side=True)

```

Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player ▾ || ⌂ ⌃ ⌄ ⌅
Activities Terminal ▾
File Edit Options Buffers Tools Python Help
Import socket, ssl
>-
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
context.load_cert_chain(certfile="mydomain.com.crt", keyfile="mydomain.com.key")
HOST = "192.168.75.128"
PORT = 65432 # Port to listen on (non-privileged ports are > 1023)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

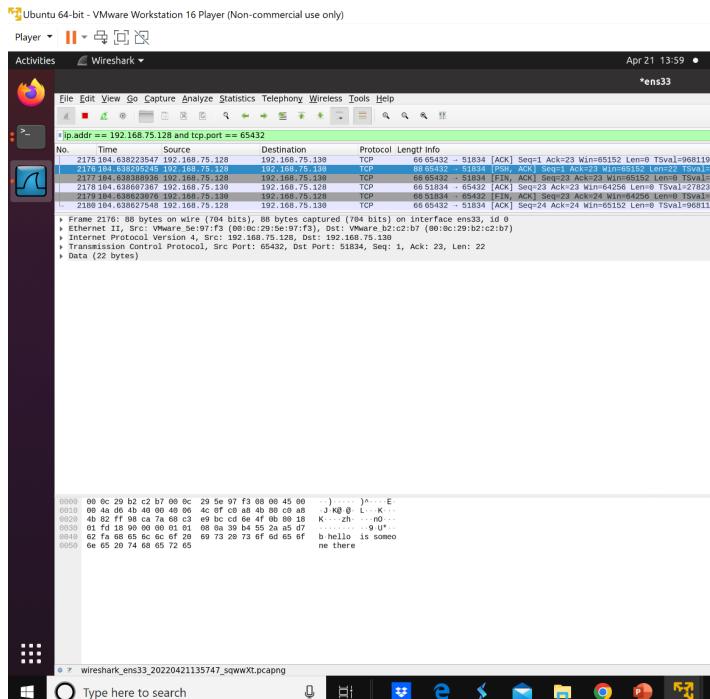
connstream = context.wrap_socket(s, server_side=True)

connstream.bind((HOST, PORT))
connstream.listen() # specify the "backlog" for this socket
conn, addr = connstream.accept() # wait and accept the connection
data = conn.recv(1024) # read the content of the message
conn.sendall(data)
#print("Server says: " + data)
conn.close()

```

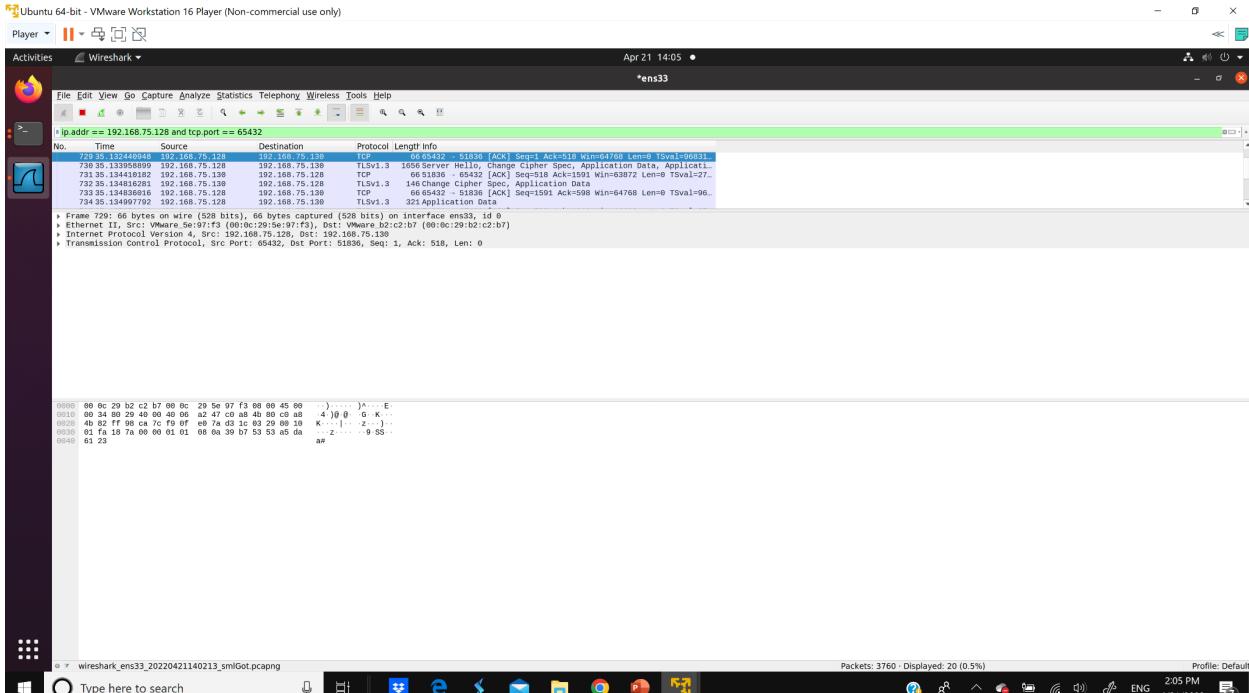
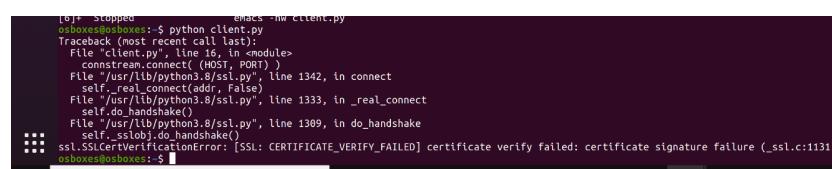
- After doing this the program runs the exact same as before

- I then decided to verify that the encryption works by using wireshark on the server first with the unprotected no ssl version
  - I filtered it with ip.addr == 192.168.75.128 and tcp.port == 65432



- as you can see the plaintext is visible with no ssl

- Now I did the same for the ssl wrapped version and when I checked through them all, I can find no plaintext that had the message, which means that it was encrypted

- 
- If you modify the server.key I believe the client can't connect to the server and you get the certificate verify failed error code because the server.key doesn't match
  - 
  - If you change the CA file on the client side, I would assume that it would not work as well because the rootCA.crt does not correspond with the one on the server side
  - 
  -