Assignment 4 OpenSSL

Farhaan Jiwa and Ashlyn Schultz
Modern Cryptography
Professor Zheng
27 March 2020

We declare that we have completed this assignment completely and entirely on our own, without any consultation with others. We have read the UAB Academic Honor Code and understand that any breach of the Honor Code may result in severe penalties.

We also declare that the following percentage distribution faithfully represents individual group members' contributions to the completion of the assignment.

Name	Overall Contribution (%)	Major work items completed by me	Signature	Date
Farhaan Jiwa	60	Worked on the development of the code and program. Also helped with edits	Farhaan Jiwa	3/29/2020
Ashlyn Schultz	40%	Worked mostly on the building of report and editing of the document.	Ashlyn Schultz	3/29/2020

Abstract:

In Python, we built and installed an OpenSSL communication server to begin our creation of a securely encrypted TLS messaging chat. By building an OpenSSL communication server, we were able to use a Certification Authority (CA) to begin the process of establishing the secure encryption chat between a client and server. The Private CA was created as a central hub for key creation and management. Then the CA is called within the establishment request of a server connection to encode the messages sent between both client and server. We would begin the communication with having the server wait for a client connection, then have messages sent between both parties; encoded as they were sent and decoded as they were printed out in the chat.

Background on Building and Installing SSL:

OpenSSL is a free library and toolkit for server and client side cryptographic communication. It is open source and provides two types of security: Secure Sockets Layer (SSL) and Transport Layer Security (TLS). Although our systems already had openssl, below is a standard guideline as to how OpenSSL is installed.

Steps for installing OPenSSL on a Mac OS:

1. First step to building using OpenSSL requires downloading the most recent version.

   ```
   cd /usr/local/src
   curl --remote-name
   https://www.openssl.org/source/openssl-1.1.1a.tar.gz
   ```

2. Then SSL needs to be configured which can be done by simply installing the new version in a different location than the original version of OpenSSL (if there was an older version previously on the system).

   ```
   tar -xzvf openssl-1.1.1a.tar.gz
   cd openssl-1.1.1a
   ```

3. The **`enable-ec_nistp_64_gcc_128`** can be used to optimize certain features within SSL which are sometimes automatically disabled.

4. Complete the install using the make command.

   ```
   ./config --prefix=/usr/local/mac-dev-env/openssl-1.1.1a
   make
   make install
   ```

5. Verify the install.

Generation of Key Pairs:

Three factors determine what kind of key can or will be generated: algorithm, size, and passphrase. OpenSSL has a limited option of algorithm keys: RSA, DSA, and ECDSA. For this assignment we used the **RSA**.

With the method below you are essentially using the CA to sign off on the request and securely connect both parties:

This key generation function will create a new key in RSA 2048 which can be assigned certificates shared between the server and client to encrypt the communication between them.

To generate the **Server RSA key**:

For the server key, we set the location requirements to **US, Alabama, Birmingham and gave the common website the name 'www.farhaanjiwa.com'.** This creates a RSA private key with aes128 and 2048 bit modulus.

```
Arifs-MBP:serverkeys farhaanjiwa$ openssl genrsa -aes128 -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
.........................+++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
Arifs-MBP:serverkeys farhaanjiwa$ openssl req -new -key server.key -out server_reqout.txt
Enter pass phrase for server.key:
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]:Alabama
Locality Name (eg, city) []:Birmingham
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UAB Modern Crypto
Organizational Unit Name (eg, section) []:CS600
Common Name (e.g. server FQDN or YOUR name) []:www.farhaanjiwa.com
Email Address []:server@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Client RSA key:

Client was set with the same location and domain name and given the same size RSA private key with aes128 and a 2048 bit modulus.

```
Arifs-MBP:serverkeys farhaanjiwa$ openssl genrsa -aes128 -out client.key 2048
Generating RSA private key, 2048 bit long modulus
.............................................+++
...........................+++
e is 65537 (0x10001)
Enter pass phrase for client.key:
Verifying - Enter pass phrase for client.key:
Arifs-MBP:serverkeys farhaanjiwa$ openssl req -new -key client.key -out client_reqout.txt
Enter pass phrase for client.key:
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]:Alabama
Locality Name (eg, city) []:Birmingham
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UAB Modern Crypto
Organizational Unit Name (eg, section) []:CS600
Common Name (e.g. server FQDN or YOUR name) []:www.farhaanjiwa.com
Email Address []:client@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Server and client certificates:
Each certificate was set to be valid for 3650 days through the CA.

server:

```
Arifs-MBP:serverkeys farhaanjiwa$ openssl x509 -req -in server_reqout.txt -days 3650 -sha1 -CAcreateserial -CA serverca.crt -CAkey new_servercakey.pem -out server.crt
Signature ok
subject=/C=US/ST=Alabama/L=Birmingham/O=UAB Modern Crypto/OU=CS600/CN=www.farhaanjiwa.com/emailAddress=server@gmail.com
```

client:

```
Arifs-MBP:serverkeys farhaanjiwa$ openssl x509 -req -in client_reqout.txt -days 3650 -sha1 -CAcreateserial -CA serverca.crt -CAkey new_servercakey.pem -out client.crt
Signature ok
subject=/C=US/ST=Alabama/L=Birmingham/O=UAB Modern Crypto/OU=CS600/CN=www.farhaanjiwa.com/emailAddress=client@gmail.com
```

Cipher Suite Selection:

In order to establish security, the cipher suite command needs to be configured. It allows you to configure what suites are supported and their priority. Then from the list of possible suites, the order can be set.

It's important to first be able to identify the different ciphers that are readily available on the openssl with the command – **`openssl ciphers -v 'ALL:COMPLIMENTSOFALL'.`** This command lists all the ciphers that openssl can use to encrypt/decrypt traffic.

```
Arifs-MBP:ca farhaanjiwa$ openssl ciphers -v 'ALL:COMPLIMENTSOFALL'
ADH-SEED-SHA             SSLv3 Kx=DH       Au=None Enc=SEED(128) Mac=SHA1
DHE-RSA-SEED-SHA         SSLv3 Kx=DH       Au=RSA  Enc=SEED(128) Mac=SHA1
DHE-DSS-SEED-SHA         SSLv3 Kx=DH       Au=DSS  Enc=SEED(128) Mac=SHA1
SEED-SHA                 SSLv3 Kx=RSA      Au=RSA  Enc=SEED(128) Mac=SHA1
ADH-AES256-SHA           SSLv3 Kx=DH       Au=None Enc=AES(256)  Mac=SHA1
DHE-RSA-AES256-SHA       SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-DSS-AES256-SHA       SSLv3 Kx=DH       Au=DSS  Enc=AES(256)  Mac=SHA1
AES256-SHA               SSLv3 Kx=RSA      Au=RSA  Enc=AES(256)  Mac=SHA1
ADH-AES128-SHA           SSLv3 Kx=DH       Au=None Enc=AES(128)  Mac=SHA1
DHE-RSA-AES128-SHA       SSLv3 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA1
DHE-DSS-AES128-SHA       SSLv3 Kx=DH       Au=DSS  Enc=AES(128)  Mac=SHA1
AES128-SHA               SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA1
ADH-DES-CBC3-SHA         SSLv3 Kx=DH       Au=None Enc=3DES(168) Mac=SHA1
ADH-DES-CBC-SHA          SSLv3 Kx=DH       Au=None Enc=DES(56)   Mac=SHA1
EXP-ADH-DES-CBC-SHA      SSLv3 Kx=DH(512)  Au=None Enc=DES(40)   Mac=SHA1 export
ADH-RC4-MD5              SSLv3 Kx=DH       Au=None Enc=RC4(128)  Mac=MD5
EXP-ADH-RC4-MD5          SSLv3 Kx=DH(512)  Au=None Enc=RC4(40)   Mac=MD5  export
EDH-RSA-DES-CBC3-SHA     SSLv3 Kx=DH       Au=RSA  Enc=3DES(168) Mac=SHA1
EDH-RSA-DES-CBC-SHA      SSLv3 Kx=DH       Au=RSA  Enc=DES(56)   Mac=SHA1
EXP-EDH-RSA-DES-CBC-SHA SSLv3 Kx=DH(512)  Au=RSA  Enc=DES(40)   Mac=SHA1 export
EDH-DSS-DES-CBC3-SHA     SSLv3 Kx=DH       Au=DSS  Enc=3DES(168) Mac=SHA1
EDH-DSS-DES-CBC-SHA      SSLv3 Kx=DH       Au=DSS  Enc=DES(56)   Mac=SHA1
EXP-EDH-DSS-DES-CBC-SHA SSLv3 Kx=DH(512)  Au=DSS  Enc=DES(40)   Mac=SHA1 export
DES-CBC3-SHA             SSLv3 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=SHA1
DES-CBC-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=DES(56)   Mac=SHA1
EXP-DES-CBC-SHA          SSLv3 Kx=RSA(512) Au=RSA  Enc=DES(40)   Mac=SHA1 export
EXP-RC2-CBC-MD5          SSLv3 Kx=RSA(512) Au=RSA  Enc=RC2(40)   Mac=MD5  export
RC4-SHA                  SSLv3 Kx=RSA      Au=RSA  Enc=RC4(128)  Mac=SHA1
RC4-MD5                  SSLv3 Kx=RSA      Au=RSA  Enc=RC4(128)  Mac=MD5
EXP-RC4-MD5              SSLv3 Kx=RSA(512) Au=RSA  Enc=RC4(40)   Mac=MD5  export
DES-CBC3-MD5             SSLv2 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=MD5
DES-CBC-MD5              SSLv2 Kx=RSA      Au=RSA  Enc=DES(56)   Mac=MD5
EXP-RC2-CBC-MD5          SSLv2 Kx=RSA(512) Au=RSA  Enc=RC2(40)   Mac=MD5  export
RC2-CBC-MD5              SSLv2 Kx=RSA      Au=RSA  Enc=RC2(128)  Mac=MD5
EXP-RC4-MD5              SSLv2 Kx=RSA(512) Au=RSA  Enc=RC4(40)   Mac=MD5  export
RC4-MD5                  SSLv2 Kx=RSA      Au=RSA  Enc=RC4(128)  Mac=MD5
```

While looking through, we see that there are many encryption ciphers available to our disposal, but it's important to sort through the ciphers from strongest encryption standards to the weakest in order for our server to know which cipher to use when communicating with a client. Also by giving the option of not using weak encryption ciphers, it essentially makes sure to use the best ciphers possible for maximum security.

Use **`openssl ciphers -v`** then the list of RSA encryptions that will be applied to the server and client communication output.

The following picture shows the standards we have applied to our server client chat encryption:

```
Arifs-MBP:ca farhaanjiwa$ openssl ciphers -v 'ALL:!aNULL:!ADH:!eNULL:!LOW:!EXP:!
RC4:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:AES128-SHA:AES256-SHA:DES-CBC3-SHA:RC4
+RSA:+HIGH:+MEDIUM'
DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-DSS-AES256-SHA      SSLv3 Kx=DH       Au=DSS  Enc=AES(256)  Mac=SHA1
AES256-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-RSA-AES128-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA1
DHE-DSS-AES128-SHA      SSLv3 Kx=DH       Au=DSS  Enc=AES(128)  Mac=SHA1
AES128-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA1
EDH-RSA-DES-CBC3-SHA    SSLv3 Kx=DH       Au=RSA  Enc=3DES(168) Mac=SHA1
EDH-DSS-DES-CBC3-SHA    SSLv3 Kx=DH       Au=DSS  Enc=3DES(168) Mac=SHA1
DES-CBC3-SHA            SSLv3 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=SHA1
DES-CBC3-MD5            SSLv2 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=MD5
DHE-RSA-SEED-SHA        SSLv3 Kx=DH       Au=RSA  Enc=SEED(128) Mac=SHA1
DHE-DSS-SEED-SHA        SSLv3 Kx=DH       Au=DSS  Enc=SEED(128) Mac=SHA1
SEED-SHA                SSLv3 Kx=RSA      Au=RSA  Enc=SEED(128) Mac=SHA1
RC2-CBC-MD5             SSLv2 Kx=RSA      Au=RSA  Enc=RC2(128)  Mac=MD5
Arifs-MBP:ca farhaanjiwa$
```

Ensure that the following is set up in the **`SSLCipherSuite`** list and that **`SSLhonorcipherorder`** is on to make sure that the server is able to prioritize and use the list provided ranging from the greatest encryption to the least.

```
SSLCipherSuite:"ALL:!aNULL:!ADH:!eNULL:!LOW:!EXP:!RC4:DHE-RSA-AES128-S
HA:DHE-RSA-AES256-SHA:AES128-SHA:AES256-SHA:DES-CBC3-SHA:RC4+RSA:+HIGH
:+MEDIUM"
SSLHonorCipherOrder On
```

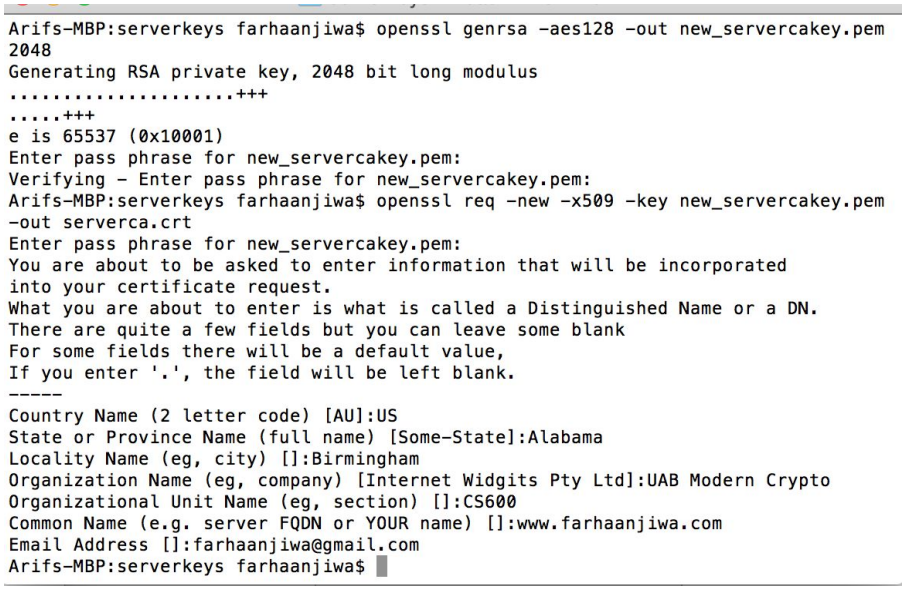
Once done, make sure to restart the service for changes to take effect.

Creating a private certification authority:

Once the key is created ensure that it is outputted to the Certificate Signing Request (CSR) file folder. Then certificates can be generated. The root CA generates the certificate for the Online Certificate Status Protocol (OCSP). Lifetime of a certification can be up to the administrator, but certificates cannot be unissued so it is recommended to use a shorter time period.

First, we have to create an RSA key for our CA, the following command basically creates a new RSA key with aes128 and a 2048 bit long modulus compared to the standard RSA private key of 512 bit:

```
openssl genrsa -aes128 -out new_servercakey.pem 2048
```



```
Arifs-MBP:serverkeys farhaanjiwa$ openssl genrsa -aes128 -out new_servercakey.pem
2048
Generating RSA private key, 2048 bit long modulus
.....................+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for new_servercakey.pem:
Verifying - Enter pass phrase for new_servercakey.pem:
Arifs-MBP:serverkeys farhaanjiwa$ openssl req -new -x509 -key new_servercakey.pem
-out serverca.crt
Enter pass phrase for new_servercakey.pem:
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]:Alabama
Locality Name (eg, city) []:Birmingham
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UAB Modern Crypto
Organizational Unit Name (eg, section) []:CS600
Common Name (e.g. server FQDN or YOUR name) []:www.farhaanjiwa.com
Email Address []:farhaanjiwa@gmail.com
Arifs-MBP:serverkeys farhaanjiwa$
```

Next, our group opted to not create a CSR because, the following command creates a self-signed certificate starting with a key alone as seen above:

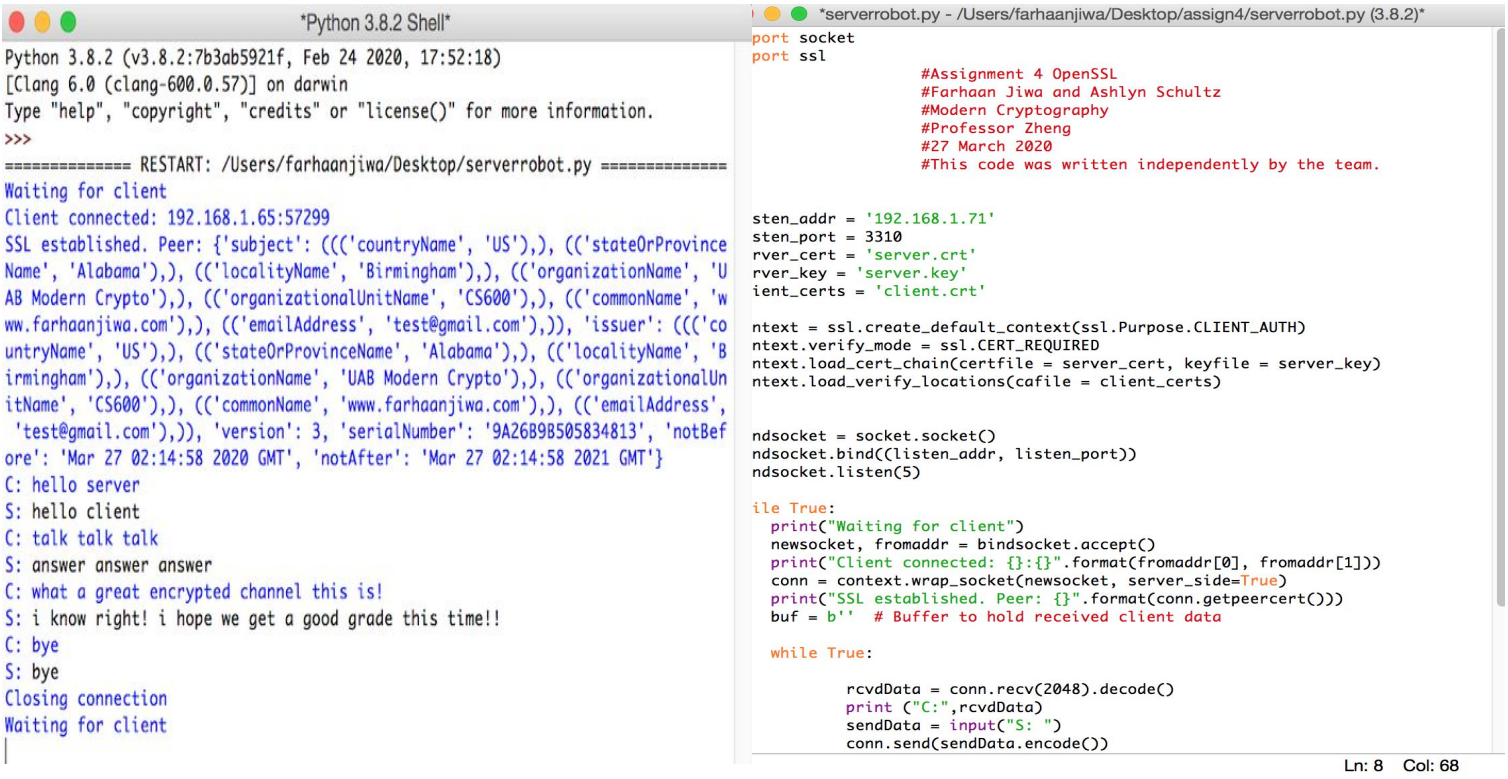
```
$ openssl req -new -x509 -days 365 -key new_servercakey.pem -out
server ca.crt
```

Testing with OpenSSL:

The SSL communication was set to use both the server certification and the server key or client certification and client key appropriate to the side of the communication. The established connection begins with a waiting on client message. Once the client has securely joined, the conversation is set and the SSL is established. The conversation can be carried until one party of communication introduces the ‘bye’ command which calls to close the secure communication chat. Once the call is confirmed by the other side, the connection is closed.

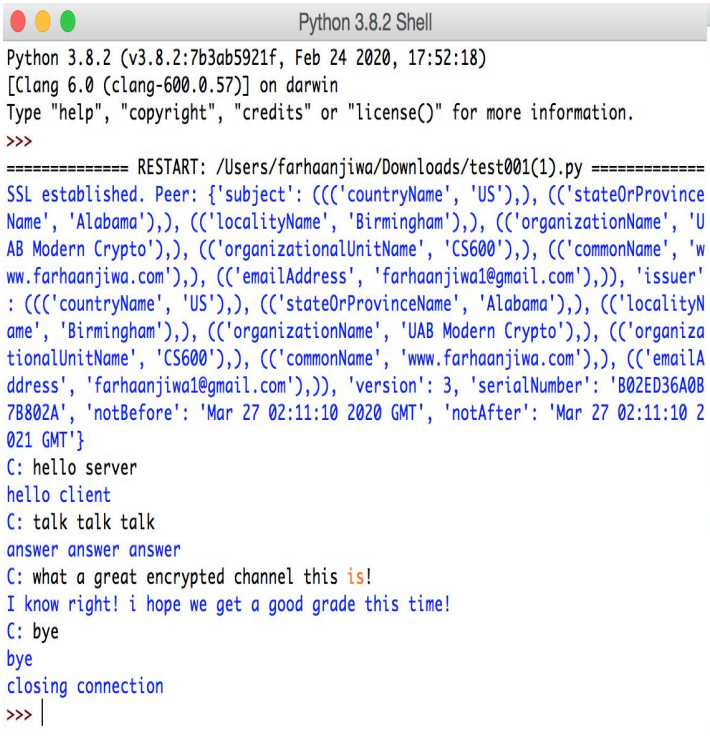
Server side:

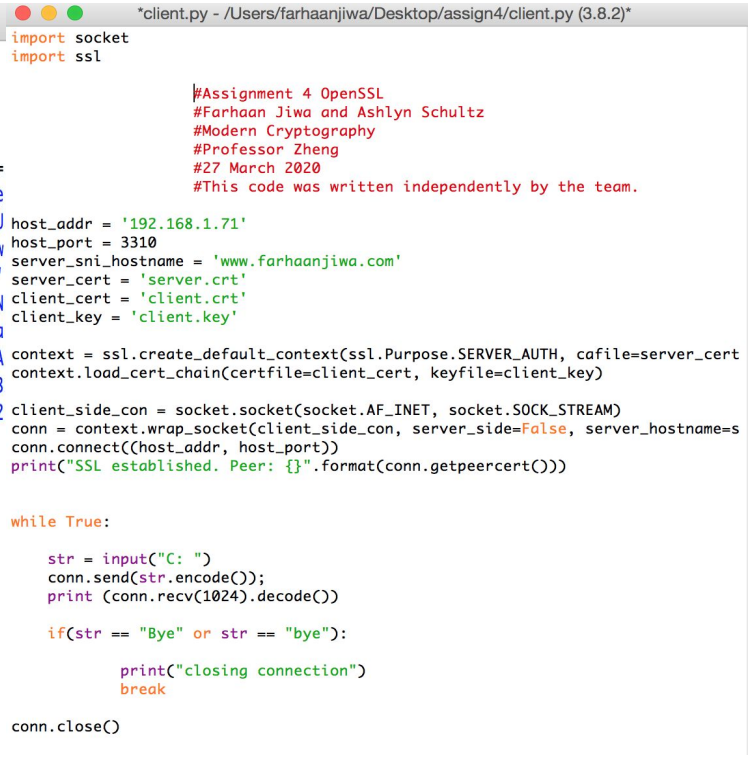
Below is the conversation from the server side of the OpenSSL. The secure connection is established, conversation carried, and then the connection is closed by the client.



Client side:

Then the same conversation from the client's side of the conversation.





Encryption and Decryption of Files:

We used AES128 encryption to encrypt and decrypt the communication between the server and client communication. Using the genrsa command in OpenSSL, we created two encryption keys: one for the server and one for the client. These keys were encrypted with a 2048 bit modulus and each given their own name ‘server.key’ and ‘client.key’.

```
Openssl genrsa -aes128 -out keynamehere.key 2048
```

The genrsa command generates a private RSA encryption key with AES128 and the 2048 bit modulus.

Advance Encryption Standard 128 (AES128) is a symmetric key algorithm therefore this key can be used to encrypt and decrypt the communication. There is no need for a second key to decrypt the encrypted chat. For the 128 bit key we used, the cipher repeats through 10 rounds of transformation before the output is encrypted (and vice versa for the decryption).

In order to establish a secure connection between the client and the server, the code calls for a **`context.load_cert_chain`.** The context represents the server authorization and set the server certification to the ‘server.crt’ file previously generated.

```
context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH,
cafile=server_cert
```

The chain command then loads in the client certification and the client key to secure the connection. The certifications set the key that is used to encrypt and decrypt the chat messages.

```
context.load_cert_chain(certfile=client_cert, keyfile=client_key)
```

Once the certification connection is complete, the host address and port are established and the secure connection begins.

```
conn.connect((host_addr, host_port))
print("SSL established. Peer: {}".format(conn.getpeercert()))
```

The security remains until the connection is closed by one side of the conversation using an if statement to register when the ‘bye’ command is typed in chat.

```
while True:
    str = input("C: ")
    conn.send(str.encode());
    print (conn.recv(1024).decode())

    if(str == "Bye" or str == "bye"):
            print("closing connection")
            break
```

While `True` the encoded messages are decoded from their encryption to display appropriately in plaintext for the other side to properly read the message. This loop continues to cycle and send the conversation messages until the ‘bye’ command triggers the end of the loop which leads to the `conn.close()` function and the secure connection is ended.

Lessons Learned:

- How to generate rsa keys and key pairs using OpenSSL.
- How to establish root certificates on a server.
- How to make a Certificate Authority(CA)
- Use Certificate Authorities to output certificates for servers and clients.
- The use of Cipher Suite selection and how to manipulate/ prioritize what ciphers are used by the server.
- A better understanding of the process in which encryption/ decryption of messages works in a server-client relationship.

References:

Ristic, Ivan. *OpenSSL Cookbook*, Fiesty Duck, 15 Mar. 2020, www.feistyduck.com/library/openssl-cookbook/online/.

"Using OpenSSL to Encrypt Messages and Files on Linux." *Linux Tutorials - Learn Linux Configuration*, Linuxconfig, 12 Apr. 2016, linuxconfig.org/using-openssl-to-encrypt-messages-and-files-on-linux.