Programming Assignment 4

CST 311, Introduction to Computer Networks

PLEASE READ THE ASSIGNMENT DESCRIPTION CAREFULLY BEFORE YOU START.

The assignment must be submitted electronically to Canvas.

In this assignment, you will develop TLS-enabled servers on a mininet network.

Instructions

Follow the steps below to set up the mininet network and write programs for the TLS-enabled chat server. Your programs must have sufficient comments to clearly explain how your code works.

1. Setting up the Mininet network.

The first task of this assignment is setting up a Mininet network as described in Figure 1. **You need to implement a Python setup program based on a provided starter code.**

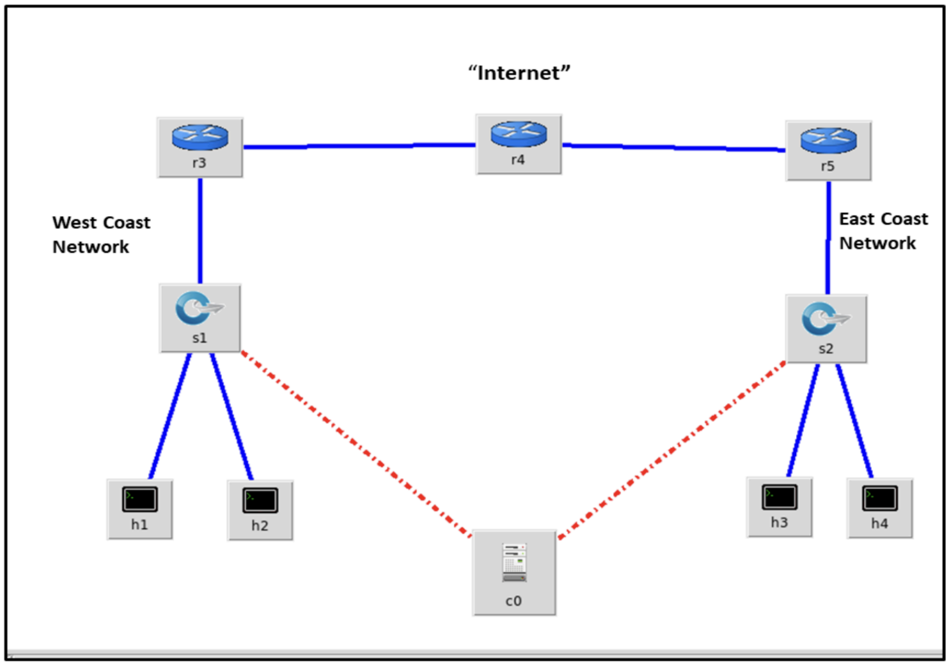
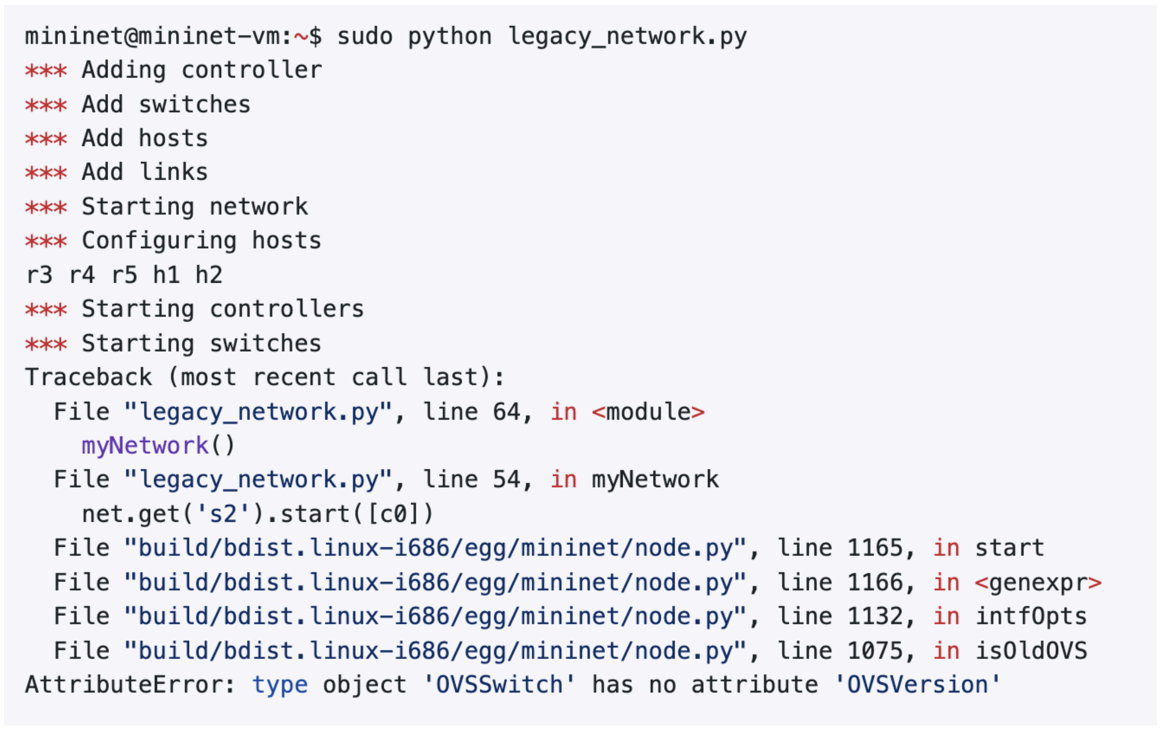


Figure 1. Mininet network structure.

Starter code. You can download the starter code here. Please extract and retrieve the `starter_code.py` file from the downloaded zip file. This code is generated by exporting the mininet network shown in Figure 1 from the MiniEdit program. You might want to review the MiniEdit program and its output file format from Lab 4A. Please rename the starter code file to `legacy_network.py` and follow the instructions below to complete the network setup.

1. After downloading the starter code and running the program for the first time, an error will occur, as shown in Figure 2.



```
mininet@mininet-vm:~$ sudo python legacy_network.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
r3 r4 r5 h1 h2
*** Starting controllers
*** Starting switches
Traceback (most recent call last):
  File "legacy_network.py", line 64, in <module>
    myNetwork()
  File "legacy_network.py", line 54, in myNetwork
    net.get('s2').start([c0])
  File "build/bdist.linux-i686/egg/mininet/node.py", line 1165, in start
  File "build/bdist.linux-i686/egg/mininet/node.py", line 1166, in <genexpr>
  File "build/bdist.linux-i686/egg/mininet/node.py", line 1132, in intfOpts
  File "build/bdist.linux-i686/egg/mininet/node.py", line 1075, in isOldOVS
AttributeError: type object 'OVSSwitch' has no attribute 'OVSVersion'
```

Figure 2. An error occurs when running the starter code without being modified.

Your first step is getting rid of this build error. The error occurs because of the order in which MiniEdit creates the switches and legacy routers. You need to move the two lines instantiating switches s1 and s2 ahead of the legacy router instantiations in the section of code labeled `info( '*** Add switches\n').`

2. After fixing the error from the previous step and running the program to initialize the network in Mininet successfully, you need to modify the program further so that the legacy routers can forward packets between the four hosts and each other. To achieve this, you need to assign the IP addresses and specify the routes properly for the hosts and routers in the network.

You need to follow the following rules to assign the IP addresses and specify the routes:

a. Hosts h1, h2, and the router r3 interface connected to s1 will be on a 254-host network with a private IPv4 address of 10.0.x.0/24. (You choose the value of x.)
b. Hosts h3, h4, and the router r5 interface connected to s2 will be on a network with a private IPv4 address of 10.0.y.0/24. (You choose the value of y.)
c. The link between router r3 and router r4 will be a 2-host network with the IPv4 address of 192.168.a.b. (You choose the values of a and b.)
Hint: Remember a 2-host network must also include the IP address of the network number and the broadcast address.
d. The link between router r4 and router r5 will be a 2-host network with the IPv4 address of 192.168.c.d. (You choose the values of c and d.)
e. The starter code has already inserted the commands to make r3, r4, and r5 routers. You will have to add static routes (or static and default routes) to make this network work.

Note that you might not need to specify an IP address for the switches.

Example. Figure 3 shows an example of how to assign IP addresses for the hosts and routers of a similar network with only two hosts.

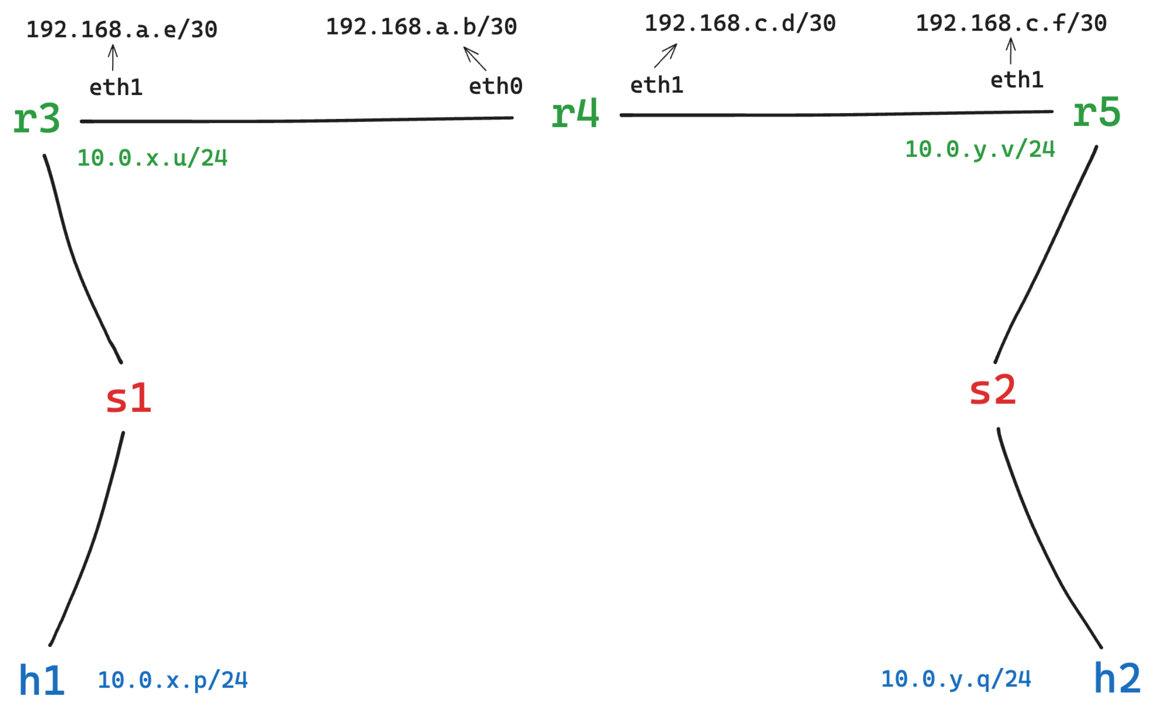


Figure 3. Sample network.

Specifying interfaces' IP addresses: You can set the IP addresses for the hosts and routers in the "`ip`" field of the `net.addHost()` method. Note that the routers can have different interfaces with different IP addresses. For routers r3 and r5, specify the values for the "`ip`" field using the IP addresses mentioned in items a. and b. above. For router r4, specify the value for the "ip" field using the IP address mentioned in item c. above.

After assigning the IP addresses for the hosts and for some interfaces of the routers. You still need to assign IP addresses for the other interfaces of the routers. There are a few different ways to do this. One way is to modify codes in the section labeled `info( '*** Add links\n')`. In this code section , you will see statements like: `net.addLink(r3, r4)`. This particular statement just adds the link between router r3 and router r4. You can modify the statement to:

```
net.addLink(r3,r4,intfName1='r3-eth1',params1={'ip':'192.168.0.1/30'},
intfName2='r4-eth0', params2={'ip':'192.168.0.2/30'})
```

This assigns the IP address of 192.168.0.1/30 to the r3-eth1 interface for r3 and the IP address of 192.168.0.2/30 to the r4-eth0 interface for r4. The number 1 in `intfName1` and `params1` refers to the first node of the r3, r4 pair (which is r3), and the number 2 in `intfName2` and `params2` refers to the second node of the r3, r4 pair (which is r4).

You might want to use this approach to assign IP addresses to the other interfaces of r3, r4, and r5 in `net.addLink(r3, r4)` and `net.addLink(r4, r5)`. Please make sure that you do not change the order of these two lines in the starter code since it might affect the network setup.

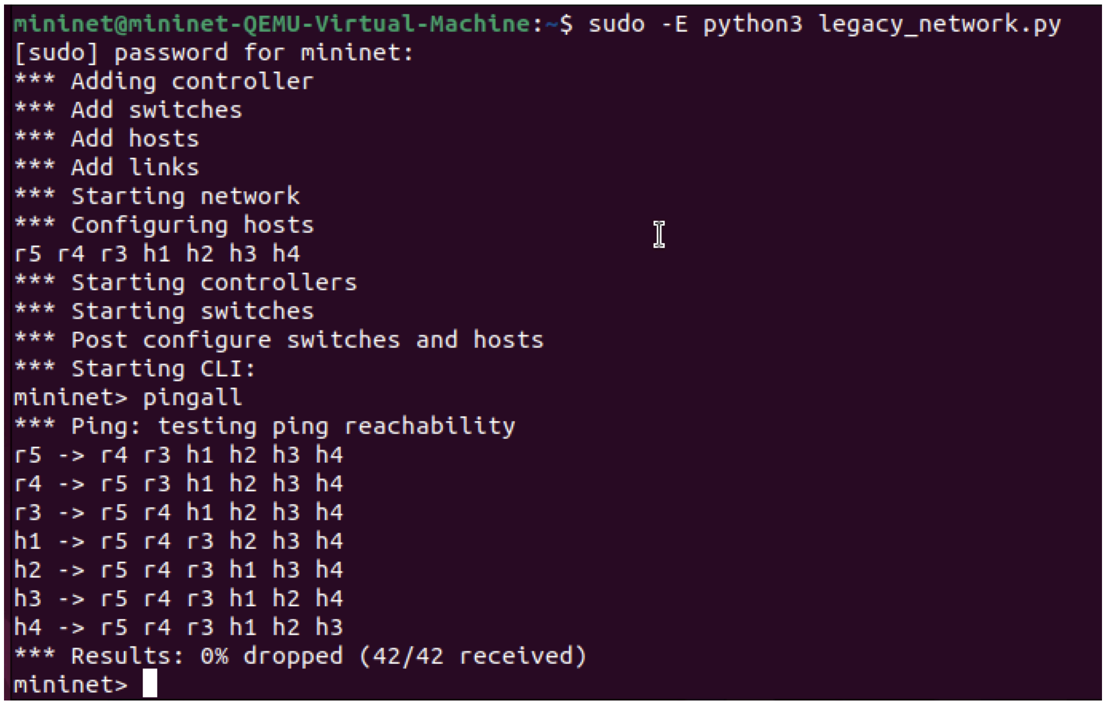
Adding routes:
After finishing assigning IP addresses, you need to add routes to allow communications between the hosts in the network. There are two types of route that you can add:

1. Default route: A default route is the route that takes effect when no other route is available for an IP destination address. In this assignment, you can set the default route for the hosts h1, h2, h3, h4 since all packets from them need to be forwarded to one corresponding router. There are two ways to add default routes for the hosts:
    a. You can add the default route in the "`defaultRoute`" field of the `net.addHost()` method. For example, to set the default route for host h1 to the router r3 at 10.0.0.3, we can change the adding host h1 line to:
    ```
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1/24',
                                            defaultRoute='via 10.0.0.3')
    ```

b. You can also add a default route using command line executing from a host. For syntax and instructions, please refer to this document.

2. Static route: Static routing is when a router uses predefined routes instead of dynamically learning routes from network traffic. In this assignment, you should add static routes for the **routers**. Follow the instructions in the same document above to add static routes in your program. You should place these codes between the following lines in the starter program:
```
net.build()
<add codes for adding static routes here>
info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()
```
For our network, you might need to add six (6) static routes for the routers.

Requirements. When you set up the Mininet network successfully, you need to perform the `pingall` command, and it is expected that all hosts and routers can reach each other, as shown in Figure 4. Please make sure to use the "-E" flag when running the program. The flag is used to ignore the system's Python environment variables and is useful when you want to run Python in a well-defined, isolated environment, without being affected by the system's Python configuration or external libraries.


```
mininet@mininet-QEMU-Virtual-Machine:~$ sudo -E python3 legacy_network.py
[sudo] password for mininet:
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
r5 r4 r3 h1 h2 h3 h4
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
r5 -> r4 r3 h1 h2 h3 h4
r4 -> r5 r3 h1 h2 h3 h4
r3 -> r5 r4 h1 h2 h3 h4
h1 -> r5 r4 r3 h2 h3 h4
h2 -> r5 r4 r3 h1 h3 h4
h3 -> r5 r4 r3 h1 h2 h4
h4 -> r5 r4 r3 h1 h2 h3
*** Results: 0% dropped (42/42 received)
mininet>
```
Figure 4. Successfully configured network.

Deliverables. For this section, you are expected to produce the following deliverables:

- [dlv1]: Your network design with all interfaces labeled with interface names (e.g., s1-eth1) and interface IP addresses, similar to Figure 3. You need to submit the design in PDF or PNG formats. Consider using the Excalidraw web tool to create your network design. You can re-use Figure 3 using this link. The tool allows you to export your work as a PNG file.
- [dlv2]: A text (.txt) file listing the changes that you made in the legacy_network.py from the given starter code and explaining the reasons.
- [dlv3]: A demo video showing that the network is set up correctly and the network works as expected when executing the "`pingall`" command.

2. Group Chat Service

The next task is to extend the chat service programs that you developed from Team Programming Assignment 3 (TPA3) to support a group chat feature. In particular, you need to modify the server and client programs from TPA3 so that three different clients can participate in a chat session at the same time (our programs in TPA3 only support two clients).

Then you need to modify the `legacy_network.py` file so that one of the hosts (h4) runs the updated chat server and is **TLS-enabled**; and the other hosts (h1, h2, and h3) run the updated chat client and are able to securely connect to the TLS-enabled chat server. For the TLS-enabled server requirement, please refer to the Certificate Generation section below. To programmatically run commands on an Xterm terminal of a host, use the `makeTerm` api as follows:

```
from mininet.term import makeTerm

makeTerm(<node>, title='Node', term='xterm', display=None,
cmd='<command>; bash')
```

, where `<node>` is the host and `<command>` is the command to execute on the `<node>`. Reference

For example, if we want to start the server program in an Xterm terminal of host h4, we can use the following line of code:

```
makeTerm(h4, title='Node', term='xterm', display=None,
cmd='python3 tpa4_chat_server.py; bash')
```

You should place these codes in the code section labeled `info( '*** Post configure switches and hosts\n')`, before line `CLI(net)`.

If you decide to use the above makeTerm implementation, be sure to include the following command after the net.stop() line in legacy_network.py:

```
net.stopXterms()
```

This will ensure that the terminal windows are closed when you exit mininet. Please also make sure to use the "-E" flag when running the program as explained above.

Deliverables. For this section, you are expected to produce the following deliverables:

- [dlv4]: A demo video showing that the chat service supports a group chat of three clients simultaneously, similar to the demo video that you did in TPA3.
- [dlv5]: The updated chat server and chat client programs. Please name them `tpa4_chat_server.py` and `tpa4_chat_client.py`, respectively.

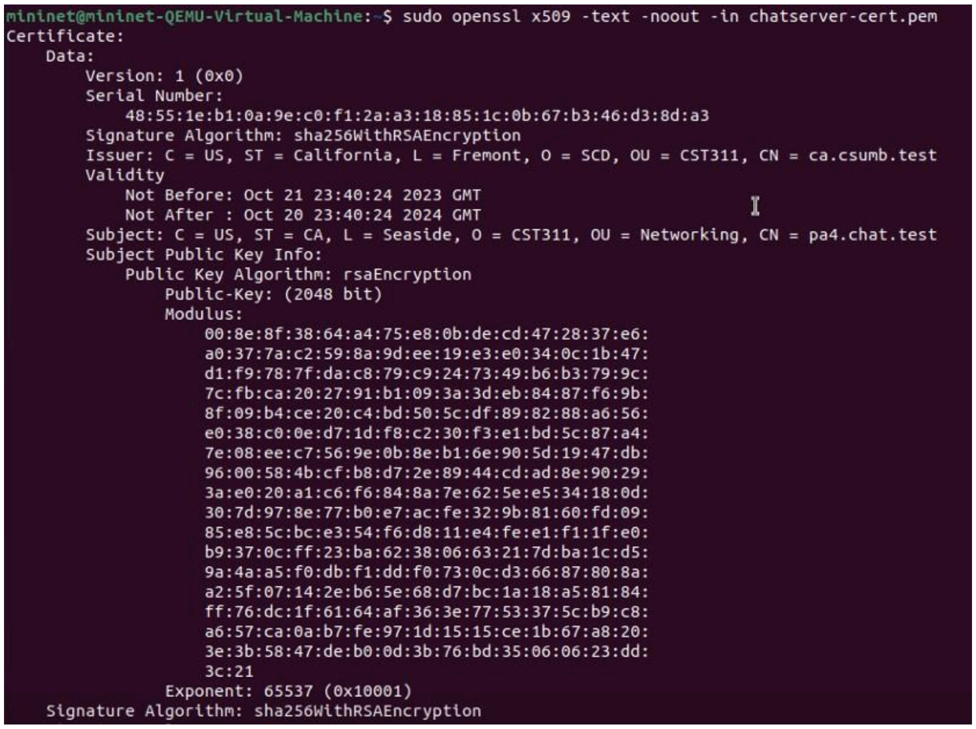
3. Certification Generation

The final task is to generate certificates for the chat server, using the Certificate Authority (CA) created in Lab 6A. This will enable secure communication between the server and clients. To do this, you will need to create a Python script titled `certificate_generation.py`. The script will prompt for the necessary information, modify system configurations, generate private keys, create certificate signing requests (CSRs), and produce server certificates using the Certificate Authority (CA) from Lab 6A.

1. Start your Python script by importing the **subprocess** module, which will allow you to run shell commands from within your Python script.
2. Prompt for the common name for your chat server.
   (*Note* - When you run the certificate generation, use the common name "**tpa4.chat.test**" and the passphrase "**CST311**".)
3. Write the **common name to a txt file**. Use Python file handling for this. This will be used later in the script to reference the common name.
4. Write a function or set of commands to add the IP addresses and common name of the server to the **/etc/hosts** file. This step requires administrative privileges, so make sure to use **sudo**.
5. Define a function to generate a **private key** for the server using the `openssl genrsa` command.
6. Create a function to generate **certificate signing requests (CSRs)** for the server. You'll need to use the `openssl req` command with the appropriate options and subject details.

7. Write a function to generate a **certificate from the CSRs** using the `openssl x509` command. This will also involve specifying the CA certificate and private key.
8. Once you have finished the certificate_generation.py file, you will need to **add the following line of code to your legacy_network.py** file in order to ensure that the certificate generation is called when the network is created. This code will go at the beginning of your network file, right after the import statements:
   `subprocess.run(["sudo", "-E", "python3", "certificate_generation.py"])`

Requirements. After completing the certificate_generation.py file and calling it in your legacy_network.py file, you should now be able to retrieve the decrypted chat server certificate, similar to how you did in Lab 6A. See an example output in Figure 5 below.



```
mininet@mininet-QEMU-Virtual-Machine:~$ sudo openssl x509 -text -noout -in chatserver-cert.pem
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            48:55:1e:b1:0a:9e:c0:f1:2a:a3:18:85:1c:0b:67:b3:46:d3:8d:a3
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, ST = California, L = Fremont, O = SCD, OU = CST311, CN = ca.csumb.test
        Validity
            Not Before: Oct 21 23:40:24 2023 GMT
            Not After : Oct 20 23:40:24 2024 GMT
        Subject: C = US, ST = CA, L = Seaside, O = CST311, OU = Networking, CN = pa4.chat.test
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:8e:8f:38:64:a4:75:e8:0b:de:cd:47:28:37:e6:
                    a0:37:7a:c2:59:8a:9d:ee:19:e3:e0:34:0c:1b:47:
                    d1:f9:78:7f:da:c8:79:c9:24:73:49:b6:b3:79:9c:
                    7c:fb:ca:20:27:91:b1:09:3a:3d:eb:84:87:f6:9b:
                    8f:09:b4:ce:20:c4:bd:50:5c:df:89:82:88:a6:56:
                    e0:38:c0:0e:d7:1d:f8:c2:30:f3:e1:bd:5c:87:a4:
                    7e:08:ee:c7:56:9e:0b:8e:b1:6e:90:5d:19:47:db:
                    96:00:58:4b:cf:b8:d7:2e:89:44:cd:ad:8e:90:29:
                    3a:e0:20:a1:c6:f6:84:8a:7e:62:5e:e5:34:18:0d:
                    30:7d:97:8e:77:b0:e7:ac:fe:32:9b:81:60:fd:09:
                    85:e8:5c:bc:e3:54:f6:d8:11:e4:fe:e1:f1:1f:e0:
                    b9:37:0c:ff:23:ba:62:38:06:63:21:7d:ba:1c:d5:
                    9a:4a:a5:f0:db:f1:dd:f0:73:0c:d3:66:87:80:8a:
                    a2:5f:07:14:2e:b6:5e:68:d7:bc:1a:18:a5:81:84:
                    ff:76:dc:1f:61:64:af:36:3e:77:53:37:5c:b9:c8:
                    a6:57:ca:0a:b7:fe:97:1d:15:15:ce:1b:67:a8:20:
                    3e:3b:58:47:de:b0:0d:3b:76:bd:35:06:06:23:dd:
                    3c:21
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
```

Figure 5. Screenshot of decrypted chat server certificate.

Deliverables. For this section, you are expected to produce the following deliverables:
- [dlv6]: A demo video showing that your script successfully issues a server certificate for the chat server. Include your decrypted certificate in the video.
- [dlv7]: The completed `certificate_generation.py` file.

Contributions

Team leaders are responsible for assessing the performance of each team member using the provided evaluation form, which can be downloaded here as an .xlsx file.

After assigning ratings to the spreadsheet following the provided instructions in the file, please save the completed document for the submission.

It's crucial for team leaders to gather feedback from all team members during the evaluation process. Additionally, review the completed spreadsheet with all team members before submission for transparency and consensus.

What to Hand in

You will hand in the completed `legacy_network.py` file and all deliverables mentioned in this document.

Your submission should include the following files:

- The completed `legacy_network.py` py file.
- The team evaluation file as described in the Contributions section.
- [dlv1] The network design. File type: PDF or PNG.
- [dlv2] The text file listing the changes that you made in the legacy_network.py from the given starter code and explaining the reasons. File type: TXT
- [dlv3][dlv4][dlv6] For these deliverables, please include all of the required demos into a single video file and upload the file to Google drive. Submit a text file with the link to the uploaded demo video. File type: TXT

  A sample demo video is available at this link.

- [dlv5] The `tpa4_chat_server.py` and `tpa4_chat_client.py` files.
- [dlv7] The `certificate_generation.py` file.

Please do NOT compress the required files to a Zip file.

Grading Rubric

Task	**Point**
Network Design Diagram	10
Network Setup: Implementation, Explanation and Demo	40
Extended Chat Server and Client Programs: Implementation and Demo	25
TLS Certification Setup: Implementation and Demo	20
Well-documented programs.	5
Total	**100**