Q5.

1. Make sure run the Python code in the folder when the Python Cryptography installed.

```
$ pip install cryptography
```

Because the Python library for cryptography is in Python 3 format.

```
chiu@LAPTOP-EON0OEAL:~$ cd __pycache__
chiu@LAPTOP-EON0OEAL:~/__pycache__$ ls
pki_helpers.cpython-36.pyc  server.cpython-36.pyc  symmetric_server.cpython-36.pyc
chiu@LAPTOP-EON0OEAL:~/__pycache__$
```

2. Make sure run the code in Python 3 environment.

Installation for Python 3 in Ubuntu Linux on Windows.

```
# Step 1: Update your repositories
sudo apt-get update

# Step 2: Install pip for Python 3
sudo apt-get install build-essential libssl-dev libffi-dev python-dev
sudo apt install python3-pip

# Step 3: Use pip to install virtualenv
sudo pip3 install virtualenv

# Step 4: Launch your Python 3 virtual environment, here the name of my virtual environment will be env3
virtualenv -p python3 env3

# Step 5: Activate your new Python 3 environment. There are two ways to do this
. env3/bin/activate # or source env3/bin/activate which does exactly the same thing

# you can make sure you are now working with Python 3
python -- version
# this command will show you what is going on: the python executable you are using is now located inside your virtualenv repository
which python

# Step 6: code your stuff

# Step 7: done? leave the virtual environment
deactivate
```

Check to make sure you are in Python 3 environment.

```
chiu@LAPTOP-EON0OEAL:~$ . env3/bin/activate
(env3) chiu@LAPTOP-EON0OEAL:~$ python -V
Python 3.6.9
(env3) chiu@LAPTOP-EON0OEAL:~$
```

vi pik_helpers.py file, put the code for the private key in the file.

```
(env3) chiu@LAPTOP-EON0OEAL:~$ cat pki_helpers.py
# pki_helpers.py
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa

from datetime import datetime, timedelta
from cryptography import x509
from cryptography.x509.oid import NameOID
from cryptography.hazmat.primitives import hashes

def generate_private_key(filename: str, passphrase: str):
    private_key = rsa.generate_private_key(
        public_exponent=65537, key_size=2048, backend=default_backend()
    )

    utf8_pass = passphrase.encode("utf-8")
    algorithm = serialization.BestAvailableEncryption(utf8_pass)

    with open(filename, "wb") as keyfile:
        keyfile.write(
            private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.TraditionalOpenSSL,
                encryption_algorithm=algorithm,
            )
        )

    return private_key
```

Put the code for the public key in to the pik_helpers.py together with the private key def. make sure have all the import at the beginning of the file.

```
def generate_public_key(private_key, filename, **kwargs):
    subject = x509.Name(
        [
            x509.NameAttribute(NameOID.COUNTRY_NAME, kwargs["country"]),
            x509.NameAttribute(
                NameOID.STATE_OR_PROVINCE_NAME, kwargs["state"]
            ),
            x509.NameAttribute(NameOID.LOCALITY_NAME, kwargs["locality"]),
            x509.NameAttribute(NameOID.ORGANIZATION_NAME, kwargs["org"]),
            x509.NameAttribute(NameOID.COMMON_NAME, kwargs["hostname"]),
        ]
    )

    # Because this is self signed, the issuer is always the subject
    issuer = subject

    # This certificate is valid from now until 30 days
    valid_from = datetime.utcnow()
    valid_to = valid_from + timedelta(days=30)

    # Used to build the certificate
    builder = (
        x509.CertificateBuilder()
        .subject_name(subject)
        .issuer_name(issuer)
        .public_key(private_key.public_key())
        .serial_number(x509.random_serial_number())
        .not_valid_before(valid_from)
        .not_valid_after(valid_to)
    )

    # Sign the certificate with the private key
    public_key = builder.sign(
        private_key, hashes.SHA256(), default_backend()
    )

    with open(filename, "wb") as certfile:
        certfile.write(public_key.public_bytes(serialization.Encoding.PEM))

    return public_key
```

Using these two functions, you can generate your private and public key pair quite quickly in Python:

```
chiu@LAPTOP-EON0OEAL:~$ . env3/bin/activate
(env3) chiu@LAPTOP-EON0OEAL:~$ python
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pki_helpers import generate_private_key, generate_public_key
>>> private_key = generate_private_key("ca-private-key.pem", "secret_password")
>>> private key
  File "<stdin>", line 1
    private key
               ^
SyntaxError: invalid syntax
>>> private_key
<cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey object at 0x7fa2a1b105f8>
```

```
>>> generate_public_key(
...   private_key,
...   filename="ca-public-key.pem",
...   country="US",
...   state="Maryland",
...   locality="Baltimore",
...   org="My CA Company",
...   hostname="my-ca.com",
... )
<Certificate(subject=<Name(C=US,ST=Maryland,L=Baltimore,O=My CA Company,CN=my-ca.com)>, ...)>
>>>
```

In your directory you should now have two files:

```
(env3) chiu@LAPTOP-EON0OEAL:~$ ls ca*
ca-private-key.pem  ca-public-key.pem
(env3) chiu@LAPTOP-EON0OEAL:~$
```

Paste the code for generating a CSR into the pki_helpers.py file. Altogether with the old code.

```
    return public_key


def generate_csr(private_key, filename, **kwargs):
    subject = x509.Name(
        [
            x509.NameAttribute(NameOID.COUNTRY_NAME, kwargs["country"]),
            x509.NameAttribute(
                NameOID.STATE_OR_PROVINCE_NAME, kwargs["state"]
            ),
            x509.NameAttribute(NameOID.LOCALITY_NAME, kwargs["locality"]),
            x509.NameAttribute(NameOID.ORGANIZATION_NAME, kwargs["org"]),
            x509.NameAttribute(NameOID.COMMON_NAME, kwargs["hostname"]),
        ]
    )

    # Generate any alternative dns names
    alt_names = []
    for name in kwargs.get("alt_names", []):
        alt_names.append(x509.DNSName(name))
    san = x509.SubjectAlternativeName(alt_names)

    builder = (
        x509.CertificateSigningRequestBuilder()
        .subject_name(subject)
        .add_extension(san, critical=False)
    )

    csr = builder.sign(private_key, hashes.SHA256(), default_backend())

    with open(filename, "wb") as csrfile:
        csrfile.write(csr.public_bytes(serialization.Encoding.PEM))

    return csr
```

Use the same generate_private_key() from when you created your CA's private key. Using the above function and the previous methods defined, you can do the following:

```
(env3) chiu@LAPTOP-EON0OEAL:~$ python
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pki_helpers import generate_csr, generate_private_key
>>> server_private_key = generate_private_key(
...   "server-private-key.pem", "serverpassword"
... )
>>> server_private_key
<cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey object at 0x7f90373116a0>
>>> generate_csr(
...   server_private_key,
...   filename="server-csr.pem",
...   country="US",
...   state="Maryland",
...   locality="Baltimore",
...   org="My Company",
...   alt_names=["localhost"],
...   hostname="my-site.com",
... )
<cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest object at 0x7f90373445f8>
>>>
[3]+  Stopped                 python
(env3) chiu@LAPTOP-EON0OEAL:~$ ls
```

You can view your new CSR and private key from the console:

```
(env3) chiu@LAPTOP-EON0OEAL:~$ ls server*.pem
server-csr.pem  server-private-key.pem  server
(env3) chiu@LAPTOP-EON0OEAL:~$
```

Add another function to your pki_helpers.py file:

```
def sign_csr(csr, ca_public_key, ca_private_key, new_filename):
    valid_from = datetime.utcnow()
    valid_until = valid_from + timedelta(days=30)

    builder = (
        x509.CertificateBuilder()
        .subject_name(csr.subject)
        .issuer_name(ca_public_key.subject)
        .public_key(csr.public_key())
        .serial_number(x509.random_serial_number())
        .not_valid_before(valid_from)
        .not_valid_after(valid_until)
    )

    for extension in csr.extensions:
        builder = builder.add_extension(extension.value, extension.critical)

    public_key = builder.sign(
        private_key=ca_private_key,
        algorithm=hashes.SHA256(),
        backend=default_backend(),
    )

    with open(new_filename, "wb") as keyfile:
        keyfile.write(public_key.public_bytes(serialization.Encoding.PEM))

(env3) chiu@LAPTOP-EON0OEAL:~$
```

You'll need to load your CSR and your CA's private and public key. Begin by loading your CSR

Next up, you'll need to load your CA's public key

The final step is to load your CA's private key

Run the following step altogether:

```
>>> from cryptography import x509
>>> from cryptography.hazmat.backends import default_backend
>>> csr_file = open("server-csr.pem", "rb")
>>> csr = x509.load_pem_x509_csr(csr_file.read(), default_backend())
>>> csr
<cryptography.hazmat.backends.openssl.x509._CertificateSigningRequest object at 0x7fa278e28390>
>>> ca_public_key_file = open("ca-public-key.pem", "rb")
>>> ca_public_key = x509.load_pem_x509_certificate(
...   ca_public_key_file.read(), default_backend()
... )
>>> ca_public_key
<Certificate(subject=<Name(C=US,ST=Maryland,L=Baltimore,O=My CA Company,CN=my-ca.com)>, ...)>
>>> from getpass import getpass
>>> from cryptography.hazmat.primitives import serialization
>>> ca_private_key_file = open("ca-private-key.pem", "rb")
>>> ca_private_key = serialization.load_pem_private_key(
...   ca_private_key_file.read(),
...   getpass().encode("utf-8"),
...   default_backend(),
... )
Password:
>>> private_key
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'private_key' is not defined
>>> private_key
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'private_key' is not defined
>>> ca_private_key
<cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey object at 0x7fa2781b22e8>
>>>
```

Password: secret_password

private_key can’t display use ca_private_key instead

```
chiu@LAPTOP-EON0OEAL:~$ . env3/bin/activate
(env3) chiu@LAPTOP-EON0OEAL:~$ python
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pki_helpers import generate_private_key, generate_public_key
>>> private_key = generate_private_key("ca-private-key.pem", "secret_password")
>>> private key
  File "<stdin>", line 1
    private key
               ^
SyntaxError: invalid syntax
>>> private_key
<cryptography.hazmat.backends.openssl.rsa._RSAPrivateKey object at 0x7fa2a1b105f8>
```

Run the following command to start your brand new Python HTTPS application:

uwsgi --master --https localhost:5683,server-public-key.pem,server-private-key.pem --mount /=server:app

PEM pass phrase: serverpassword

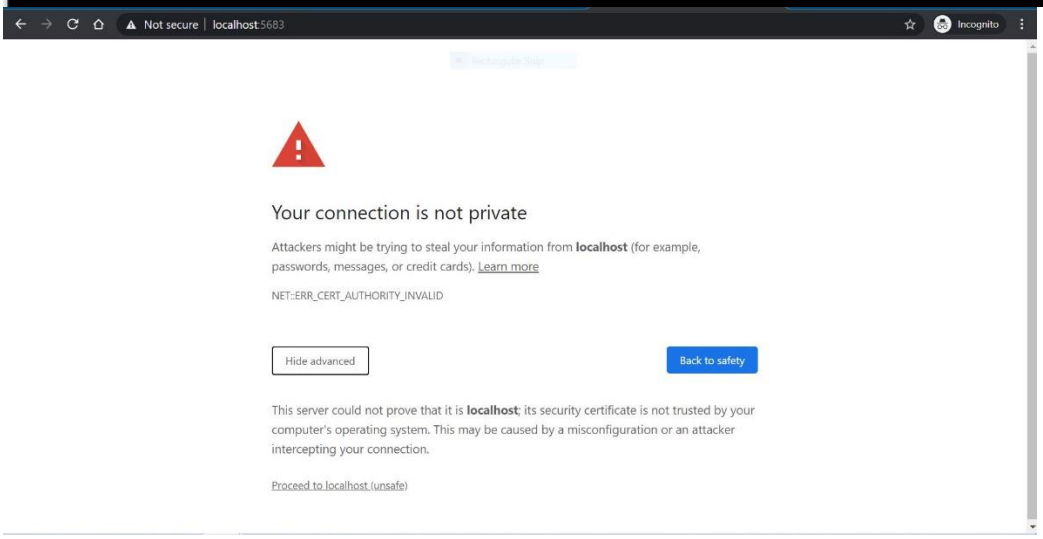
```
chiu@LAPTOP-EON0OEAL:~$ . env3/bin/activate
(env3) chiu@LAPTOP-EON0OEAL:~$ uwsgi --master --https localhost:5683,server-public-key.pem,server-private-key.pem --moun
t /=server:app
Enter PEM pass phrase:
*** Starting uWSGI 2.0.19.1 (64bit) on [Mon Aug  3 19:20:07 2020] ***
compiled with version: 7.5.0 on 21 July 2020 05:48:32
os: Linux-4.4.0-18362-Microsoft #836-Microsoft Mon May 05 16:04:00 PST 2020
nodename: LAPTOP-EON0OEAL
machine: x86_64
clock source: unix
detected number of CPU cores: 4
current working directory: /home/chiu
detected binary path: /home/chiu/env3/bin/uwsgi
!!! no internal routing support, rebuild with pcre support !!!
your processes number limit is 7823
your memory page size is 4096 bytes
detected max file descriptor number: 1024
lock engine: pthread robust mutexes
thunder lock: disabled (you can enable it with --thunder-lock)
TCP_DEFER_ACCEPT setsockopt(): Protocol not available [core/socket.c line 744]
TCP_DEFER_ACCEPT setsockopt(): Protocol not available [core/socket.c line 744]
uWSGI http bound on localhost:5683 fd 6
uwsgi socket 0 bound to TCP address 127.0.0.1:2240 (port auto-assigned) fd 5
Python version: 3.6.9 (default, Apr 18 2020, 01:56:04)  [GCC 8.4.0]
*** Python threads support is disabled. You can enable it with --enable-threads ***
Python main interpreter initialized at 0x7fffd063faf0
your server socket listen backlog is limited to 100 connections
your mercy for graceful operations on workers is 60 seconds
mapped 145840 bytes (142 KB) for 1 cores
*** Operational MODE: single process ***
```

Certificate verify failed:

```
(env3) chiu@LAPTOP-EON0OEAL:~$ cat client.py
# client.py
import os
import requests

def get_secret_message():
    response = requests.get("https://localhost:5683")
    print("The secret message is: ")
    print(response.text)

if __name__ == "__main__":
    get_secret_message()
(env3) chiu@LAPTOP-EON0OEAL:~$ python client.py
  File "/home/chiu/env3/lib/python3.6/site-packages/requests/adapters.py", line 514, in send
    raise SSLError(e, request=request)
requests.exceptions.SSLError: HTTPSConnectionPool(host='localhost', port=5683): Max retries exceeded with url: / (Caused
 by SSLError(SSLError(1, '[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:852)'),))
(env3) chiu@LAPTOP-EON0OEAL:~$
```



Use the following command for the server:

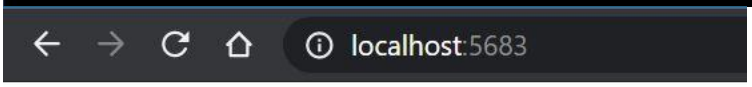
uwsgi --http-socket localhost:5683 --mount /=server:app

Avoid error message, then you have to tell requests about your Certificate Authority! Point requests at the ca-public-key.pem file that generated earlier:

```
(env3) chiu@LAPTOP-EON0OEAL:~$ uwsgi --http-socket localhost:5683 --mount /=server:app
*** Starting uWSGI 2.0.19.1 (64bit) on [Mon Aug  3 20:12:01 2020] ***
compiled with version: 7.5.0 on 21 July 2020 05:48:32
os: Linux-4.4.0-18362-Microsoft #836-Microsoft Mon May 05 16:04:00 PST 2020
nodename: LAPTOP-EON0OEAL
machine: x86_64
clock source: unix
detected number of CPU cores: 4
(env3) chiu@LAPTOP-EON0OEAL:~$ python client.py
The secret message is:
fluffy tail
(env3) chiu@LAPTOP-EON0OEAL:~$ cat client.py
# client.py
import os
import requests

def get_secret_message():
    response = requests.get("http://localhost:5683", verify="ca-public-key.pem")
    print("The secret message is: ")
    print(response.text)

if __name__ == "__main__":
    get_secret_message()
(env3) chiu@LAPTOP-EON0OEAL:~$
```



fluffy tail