1. Introduction

In this project Daniel and Yishan have developed a python based X509 certificate signing system, and a pgp certificate validation system, both integrated into the same application. This is a client/server based system. The client is a python application with a GUI and the server is also python-based. When the client generates requests, these are sent to the server, which processes the requests.

We also run a local certificate authority (CA), which is called *“Dan & Yishan”*. It has a self-signed certificate when it is first generated, and then we generate a root certificate, certificate signing request (CSR), and this CSR is signed by *Phillip Sarah*. This gives us a root certificate. At the same time, we also signed the CSR provided by *Phillip Sarah*, and the CSR provided by *Yibei He*.

From this assignment we expected to learn how to program an application that takes in signatures to a document, and outputs a signed list of these signatures, and then verifies whether a signed file is a valid signature of the given document.

2. Environment

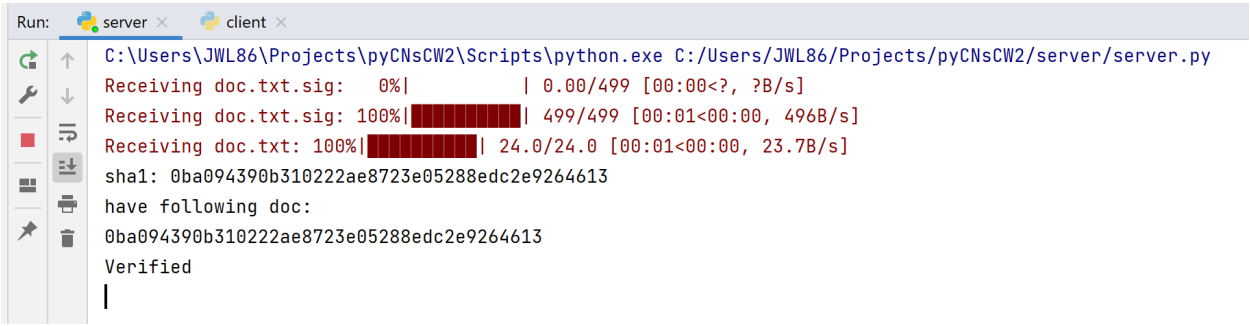
To communicate as a pair we used Discord and MSTeams, and file shared through these two applications. To program, Daniel used Visual Studio Code and Yishan used pycharm. For our application we used the following packages: altgraph==0.17.2; cffi==1.15.0; colorama==0.4.4; cryptography==36.0.0; future==0.18.2; pefile==2021.9.3; PGPy==0.5.4; pyasn1==0.4.8; pycparser==2.21; pyinstaller==4.7; pyinstaller-hooks-contrib==2021.4; pyOpenSSL==21.0.0; pywin32-ctypes==0.2.0; six==1.16.0; tqdm==4.62.3.

3. Application

what your pair did and what you produced, include an introduction section, discussing what you expect to learn from the assignment in general (and for each task), and describe the environment that you used to complete the tasks (e.g., what machines, software and versions)

In the record section, if a pgp signature is to be recorded. The first thing you need to do is to make sure that the server is running and afterwards open the client application.
In the application upload signature, document and public key respectively.

If the verification passes. That server will call the record method and save the signature and the document.The Record() function saves the signatures and documents in a dictionary. When Record() receives a set of documents, it first uses the doc to call the CalcSha1() function to generate a SHA1 string. record uses this string to determine if the document is a new document. If the SHA1 string is duplicated, then the document has been signed before, and a value will be appended to the document's signature list (the value of the key corresponding to the sha1). The length of the list will then be increased by one and an additional signature will be added.

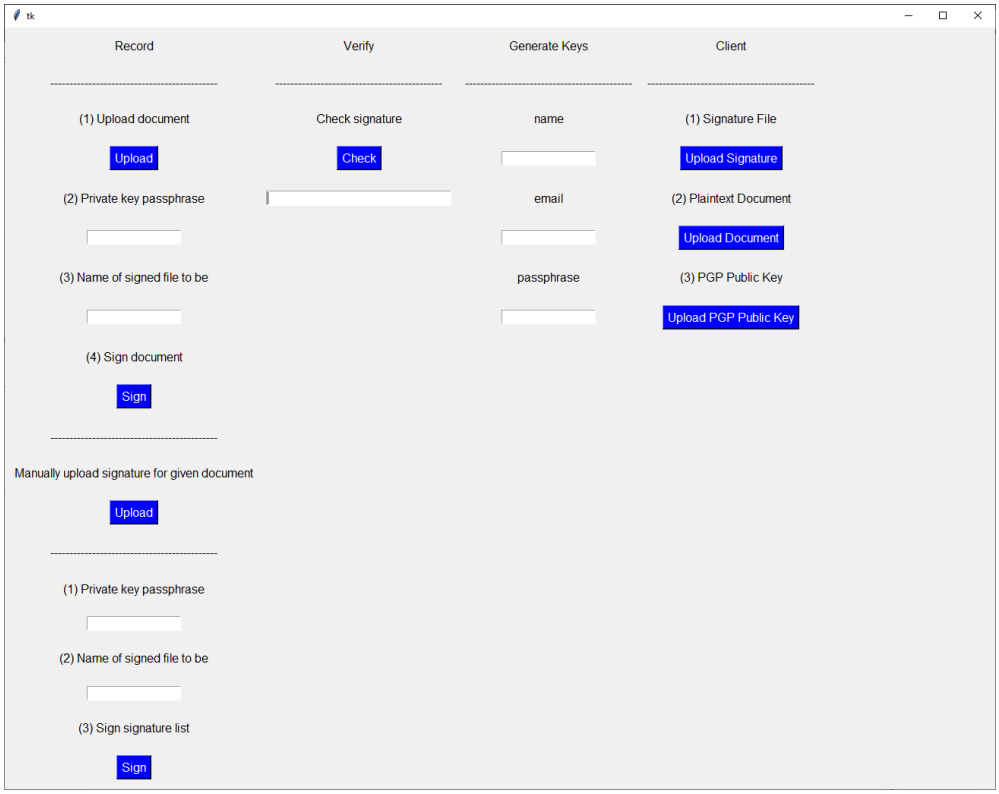


In the record section of the application interface, the user can first upload the document that they would like to add signatures for, and even sign the document by providing the password for their private key, the name of the generated signature file, and then when they press the ‘Sign’ button, they will be asked to choose the private key that they have provided the password for, and which they would like to sign the given document with. The signed document is saved to the project directory.

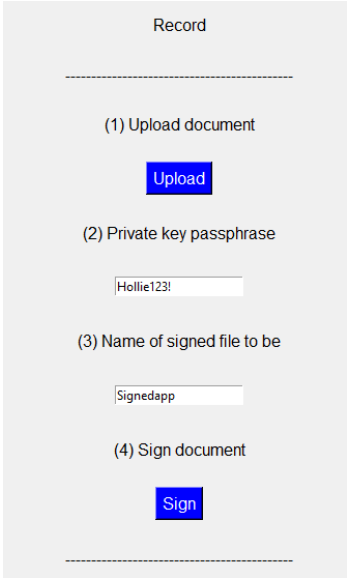
The next usbale feature is the ability to upload signatures for the given document, labelled “Manually upload signature for given document”. The uploaded signature will be added to a list behind the scenes of the appliation. From this, a private key passphrase can be entered, and the list of signatures can be signed and exported by pressing ‘Sign’ and providing a secret key to sign the list.

The validity of a signature can then be verified by pressing ‘Check’ located beneath the ‘Check signature’ label in the verify section. The output to the text box below will either show “Verified” or “Not Verified!”.

We planned to implement the generation of keys as an extra feature, but we lacked the time in the end to fulfill this aim. Our extra feature is the ability to sign a document within the application.



Below you can see the signing of our application file through our app itself. We named the resulting signature “Signedapp.txt.sig” and you can see it below.



4. Tasks 1 and 2

Task (1)
We created a local CA with the alias ‘danyishan’ along with a password. By using the command “keytool -genkeypair -alias danyishan -validity 365” we were also able to self-sign at the same time. We also generated keys for this CA

```
jove:~/Desktop/CNSCW2/iii$ keytool -genkeypair -alias danyishan -validity 365
Enter keystore password:
What is your first and last name?
  [Unknown]:  Daniel Yishan
What is the name of your organizational unit?
  [Unknown]:  Dan & Yishan
What is the name of your organization?
  [Unknown]:  Heriot-Watt
What is the name of your City or Locality?
  [Unknown]:  Edinburgh
What is the name of your State or Province?
  [Unknown]:  Lothian
What is the two-letter country code for this unit?
  [Unknown]:  UK
Is CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK correct?
  [no]:  YES

Enter key password for <danyishan>
        (RETURN if same as keystore password):
```

```
jove:~/Desktop/CNSCW2/iii$ openssl req -x509 -new -nodes -out danyishan.pem
Generating a RSA private key
..............................+++++
..+++++
writing new private key to 'privkey.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]:UK
State or Province Name (full name) [Some-State]:Lothian
Locality Name (eg, city) []:Edinburgh
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Heriot-Watt
Organizational Unit Name (eg, section) []:Dan & Yishan
Common Name (e.g. server FQDN or YOUR name) []:dfr1
Email Address []:dfr1@hw.ac.uk
jove:~/Desktop/CNSCW2/iii$ openssl pkcs8 -topk8 -in privkey.pem -outform DER -out privkey.der
Enter Encryption Password:
Verifying - Enter Encryption Password:
Verify failure
jove:~/Desktop/CNSCW2/iii$ openssl pkcs8 -topk8 -in privkey.pem -outform DER -out privkey.der
Enter Encryption Password:
Verifying - Enter Encryption Password:
```

Here you can see proof of self-signature as the owner and issuer are identical:

```
Alias name: danyishan
Creation date: 23-Nov-2021
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Issuer: CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Serial number: 2f9af908
Valid from: Tue Nov 23 13:49:21 GMT 2021 until: Wed Nov 23 13:49:21 GMT 2022
Certificate fingerprints:
         SHA1: 5A:8A:02:B7:32:7A:FD:EF:9E:05:3A:72:32:C7:5B:46:42:EB:1E:BF
         SHA256: 36:F5:E3:3A:D1:F2:46:C6:DD:10:8E:58:00:92:0C:5A:A3:B4:E5:FB:75:85:94:79:A2:45:CF:CD:38:A7:07:9F
Signature algorithm name: SHA256withDSA
Subject Public Key Algorithm: 2048-bit DSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 7D 6E CD 51 65 16 4F 01   FC BA 07 DD 4B 60 FB 68  .n.Qe.O.....K`.h
0010: 83 70 45 50                                        .pEP
]
]
```

We also ensured self signing by using our own certificate signing request and outputting a certificate (note that we renamed bert.cer to danyishan.cer):

```
jove:~/Desktop/CNSCW2/iii$ openssl x509 -req -CA ca.cert.pem -CAkey ca.key.pem -in danyishan.csr -out bert.cer -days
365 -CAcreateserial
Signature ok
subject=C = UK, ST = Lothian, L = Edinburgh, O = Heriot-Watt, OU = Dan & Yishan, CN = Daniel Yishan
Getting CA Private Key
Enter pass phrase for ca.key.pem:
```

```
jove:~/Desktop/CNSCW2/iii$ keytool -printcert -file danyishan.cer
Owner: CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Issuer: EMAILADDRESS=dfr1@hw.ac.uk, CN=dfr1, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Serial number: 7f60a847342487ecb22e7c076a7eafdd6a4e585a
Valid from: Tue Nov 23 14:14:57 GMT 2021 until: Wed Nov 23 14:14:57 GMT 2022
Certificate fingerprints:
         SHA1: FC:A7:DB:52:FA:F3:1D:90:EF:89:F9:AD:69:28:65:4E:D9:35:61:1E
         SHA256: 9B:CE:2A:51:B6:F6:D6:54:46:78:B4:15:47:CD:F1:42:0C:1C:15:CA:9A:44:52:D2:95:8D:C6:D8:3E:D2:43:64
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit DSA key
Version: 1
```

Task (2) To get our certificate signed by another pair we first had to generate a certificate signing request.

```
jove:~/Desktop/CNSCW2/iii$ keytool -certreq -alias danyishan -file danyishan.csr
Enter keystore password:

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard form
at using "keytool -importkeystore -srckeystore /home/cs4/dfr1/.keystore -destkeystore /home/cs4/dfr1/.keystore -dests
toretype pkcs12".
jove:~/Desktop/CNSCW2/iii$ keytool -printcertreq -file danyishan.csr
PKCS #10 Certificate Request (Version 1.0)
Subject: CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Format: X.509
Public Key: 2048-bit DSA key
Signature algorithm: SHA256withDSA

Extension Request:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 7D 6E CD 51 65 16 4F 01   FC BA 07 DD 4B 60 FB 68  .n.Qe.O.....K`.h
0010: 83 70 45 50                                        .pEP
]
]
```

To sign the other pairs certificate we used the command “openssl x509 -req -CA ca.cert.pem -CAkey ca.key.pem -in caps.csr -out caps.cer -days 365 -CAcreateserial”.

```
jove:~/Desktop/CNSCW2/iii$ openssl x509 -req -CA ca.cert.pem -CAkey ca.key.pem -in caps.csr -out caps.cer -days 365 -
CAcreateserial
Signature ok
subject=C = UK, ST = City of Edinburgh, L = Edinburgh, O = CAPS, OU = Computer Network Security, CN = CA PS
Getting CA Private Key
Enter pass phrase for ca.key.pem:
jove:~/Desktop/CNSCW2/iii$
```

To view the certificate we received back from the other pair who we sent the certificate signing request to, we used the command “keytool -printcert -file danyishanfromsarah.cer” and you can see that they successfully signed our certificate.

```
jove:~/Desktop/CNSCW2/iii$ keytool -printcert -file danyishanfromsarah.cer
Owner: CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Issuer: CN=CAPS, OU=Computer Network Security, O=CA Phillip Sarah, L=Edinburgh, ST=City of Edinburgh, C=UK
Serial number: fee6a2ba3707a9c9
Valid from: Tue Nov 23 15:34:24 GMT 2021 until: Wed Nov 23 15:34:24 GMT 2022
Certificate fingerprints:
         SHA1: 40:C0:F3:2F:9C:5E:27:BD:53:8D:CC:AC:E2:89:AF:0C:E5:71:D5:B9
         SHA256: CF:73:AA:F8:9E:0B:2A:0D:DF:CE:33:4B:C8:6C:2D:ED:AE:37:91:DF:7D:19:88:AD:BA:45:E5:60:C1:36:0C:8A
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit DSA key
Version: 1
```

5. Interesting Observations and Difficulties

Task (3) was really hard to understand, and we never felt as though we had a clear direction - this was frustrating. Consequently, we tried our best to produce our interpretation of the task.

We originally tried to use Java to make the application, but we came across many issues such as depreciated libraries, and eventually discovered on the discussion board that it was now recommended to use python, which wasn't the initial recommendation. This caused us to lose a lot of time, which we were already short of given other deadlines.

Signing CSR with python:
When trying to sign a CSR file using python instead of the openssl command, we first installed the necessary packages using the requirements.txt given in the gitlab demo, and then started trying to import the certificate request and pair CA. But we keep getting an error "*Valid PEM but no BEGIN CERTIFICATE REQUEST/END CERTIFICATE REQUEST delimiters. Are you sure this is a CSR?*"
We made a new csr file as we did in week 9a slide to generate a csr, but we still got the error. Then because the version of cryptography in requirements was 35.0.0, we tried install to an older version and still got the error. Eventually we found that the latest version of cryptography was 36.0.0 and after upgrading to the latest version we were able to successfully read the CSR as a file.
This bug has also been confirmed through the update log. The function to sign CSR using python was not implemented because of the amount of time wasted due to this bug.

6. Appendix

```
Alias name: danyishan
Creation date: 23-Nov-2021
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Issuer: CN=Daniel Yishan, OU=Dan & Yishan, O=Heriot-Watt, L=Edinburgh, ST=Lothian, C=UK
Serial number: 2f9af908
Valid from: Tue Nov 23 13:49:21 GMT 2021 until: Wed Nov 23 13:49:21 GMT 2022
Certificate fingerprints:
         SHA1: 5A:8A:02:B7:32:7A:FD:EF:9E:05:3A:72:32:C7:5B:46:42:EB:1E:BF
         SHA256: 36:F5:E3:3A:D1:F2:46:C6:DD:10:8E:58:00:92:0C:5A:A3:B4:E5:FB:75:85:94:79:A2:45:CF:CD:38:A7:07:9F
Signature algorithm name: SHA256withDSA
Subject Public Key Algorithm: 2048-bit DSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 7D 6E CD 51 65 16 4F 01   FC BA 07 DD 4B 60 FB 68  .n.Qe.O.....K`.h
0010: 83 70 45 50                                        .pEP
]
]
```

```
jove:~/Desktop/CNSCW2/iii$ keytool -genkeypair -alias God -ext BC:c=ca:true
Enter keystore password:
What is your first and last name?
  [Unknown]:  God
What is the name of your organizational unit?
  [Unknown]:  God
What is the name of your organization?
  [Unknown]:  God
What is the name of your City or Locality?
  [Unknown]:  God
What is the name of your State or Province?
  [Unknown]:  God
What is the two-letter country code for this unit?
  [Unknown]:  UK
Is CN=God, OU=God, O=God, L=God, ST=God, C=UK correct?
  [no]:  YES

Enter key password for <God>
        (RETURN if same as keystore password):
Re-enter new password:
```

```
jove:~/Desktop/CNSCW2/iii$ openssl req -config /etc/pki/tls/openssl.cnf -new -x509 -keyout ca.key.pem -out ca.cert.pe
m -days 365
Generating a RSA private key
.............+++++
..+++++
writing new private key to 'ca.key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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) [XX]:UK
State or Province Name (full name) []:Lothian
Locality Name (eg, city) [Default City]:Edinburgh
Organization Name (eg, company) [Default Company Ltd]:Heriot-Watt
Organizational Unit Name (eg, section) []:Dan & Yishan
Common Name (eg, your name or your server's hostname) []:dfr1
Email Address []:dfr1@hw.ac.uk
jove:~/Desktop/CNSCW2/iii$ openssl x509 -text -in ca.cert.pem
```

```
jove:~/Desktop/CNSCW2/iii$ openssl x509 -text -in ca.cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3e:27:f1:f2:52:8d:3b:99:43:22:d9:67:c0:d0:0b:6d:b0:1e:34:a1
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = UK, ST = Lothian, L = Edinburgh, O = Heriot-Watt, OU = Dan & Yishan, CN = dfr1, emailAddress = df
r1@hw.ac.uk
        Validity
            Not Before: Nov 23 14:11:12 2021 GMT
            Not After : Nov 23 14:11:12 2022 GMT
        Subject: C = UK, ST = Lothian, L = Edinburgh, O = Heriot-Watt, OU = Dan & Yishan, CN = dfr1, emailAddress = d
fr1@hw.ac.uk
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:f5:4a:af:b6:8e:a0:66:ae:a7:b7:2d:8b:e6:fb:
                    9a:42:d8:fe:fa:6e:57:98:10:42:9d:27:50:d9:ea:
                    e3:54:80:3b:55:3a:f0:bc:8d:0a:f8:48:f0:0e:f2:
                    45:89:1d:52:f4:52:0c:d8:bc:23:14:87:e9:6b:3f:
                    74:0f:36:25:0e:8e:c5:a3:33:76:fc:13:94:77:07:
                    8d:f5:82:95:b1:5b:70:ad:91:a3:4f:1c:d1:b9:d6:
                    2e:18:54:5b:bc:4b:49:e6:67:55:b3:69:ba:26:a9:
                    c5:01:3c:dd:58:c3:3d:74:64:3d:f8:e5:6a:b4:91:
                    92:0b:07:53:3e:3c:90:23:6f:d2:a1:6b:17:58:19:
                    ca:29:15:93:b9:b5:1c:bd:eb:99:39:36:e3:71:f7:
                    a8:51:63:07:c9:88:12:4d:2a:cc:06:1d:3e:16:75:
                    ad:85:a3:ea:6d:46:bb:0e:98:a9:10:75:ed:a1:4f:
                    60:36:37:11:d7:52:34:d6:5a:21:b3:c9:79:d4:cc:
                    c9:e2:30:04:ab:c4:1b:fd:b3:10:a7:6b:6a:7f:e7:
                    eb:a1:53:7a:08:1a:54:50:3c:13:50:a0:19:24:50:
                    31:df:22:15:dc:b4:25:3b:45:d2:0b:f2:e9:b7:3f:
                    5c:50:96:a5:60:a4:b6:8b:58:3a:cb:54:1c:9c:2e:
                    e4:7d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                87:70:29:F8:18:B6:94:E6:08:E8:99:9B:E2:DB:79:0D:F0:69:74:BF
            X509v3 Authority Key Identifier:
                keyid:87:70:29:F8:18:B6:94:E6:08:E8:99:9B:E2:DB:79:0D:F0:69:74:BF

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         ba:19:17:8e:2f:07:0c:ee:b4:d3:d1:37:73:31:b1:cd:5a:44:
         66:c4:15:bc:9b:a9:e8:e9:55:c0:11:31:07:0f:bb:52:0c:73:
         52:16:40:89:d2:96:a8:de:07:e5:af:77:2e:d4:1f:af:44:22:
         bb:50:88:80:f6:ca:52:e4:04:8b:46:b8:45:19:36:88:39:29:
         76:16:2c:5e:f1:5a:97:8e:3c:aa:4c:8c:86:7f:18:34:41:39:
         89:60:ee:ba:df:d4:12:c4:34:49:4f:a6:4a:a3:be:0b:3a:a3:
         c4:a7:ec:6c:f6:74:82:bb:f5:aa:31:0c:3a:72:0f:0f:a5:36:
         aa:0c:a6:e9:a2:b8:58:74:c8:8d:49:94:98:57:ff:87:35:94:
         b6:05:01:a7:dc:1c:7f:d8:d1:e2:81:59:6e:69:6e:ff:00:8f:
         0b:94:2a:bd:3a:b4:36:4d:91:41:e1:07:f5:75:96:0a:e4:97:
         cd:f4:97:c1:7d:b7:fb:cd:cf:3a:77:2e:08:04:ce:36:77:c4:
         9f:68:3f:a2:93:af:22:03:37:51:66:7c:fc:0c:32:1a:a2:4f:
         f5:1e:ed:98:c1:dd:e5:c5:4d:f2:20:af:db:5c:04:03:b6:fd:
         1a:15:2e:21:5a:74:ea:b5:8e:ba:8f:6f:e6:7b:5b:5f:54:e2:
         c6:7e:8d:9f
```


-----BEGIN CERTIFICATE-----
MIID+jCCAuKgAwIBAgIUPifx8lKNO5lDItlnwNALbbAeNKEwDQYJKoZIhvcNAQEL
BQAwgY0xCzAJBgNVBAYTAlVLMRAwDgYDVQQIDAdMb3RoaWFuMRIwEAYDVQQHDAlF
ZGluYnVyZ2gxFDASBgNVBAoMC0hlcmlvdC1XYXR0MRUwEwYDVQQLDAxEYW4gJiBZ
aXNoYW4xDTALBgNVBAMMBGRmcjExHDAaBgkqhkiG9w0BCQEWDWRmcjFAaHcuYWMu
dWswHhcNMjExMTIzMTQxMTEyWhcNMjIxMTIzMTQxMTEyWjCBjTELMAkGA1UEBhMC
VUsxEDAOBgNVBAgMB0xvdGhpYW4xEjAQBgNVBAcMCUVkaW5idXJnaDEUMBIGA1UE
CgwLSGVyaW90LVdhdHQxFTATBgNVBAsMDERhbiAmIFlpc2hhbjENMAsGA1UEAwwE
ZGZyMTEcMBoGCSqGSIb3DQEJARYNZGZyMUBody5hYy51azCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAPVKr7aOoGaup7cti+b7mkLY/vpuV5gQQp0nUNnq
41SAO1U68LyNCvhI8A7yRYkdUvRSDNi8IxSH6Ws/dA82JQ6OxaMzdvwTlHcHjfWC
lbFbcK2Ro08c0bnWLhhUW7xLSeZnVbNpuiapxQE83VjDPXRkPfjlarSRkgsHUz48
kCNv0qFrF1gZyikVk7m1HL3rmTk243H3qFFjB8mIEk0qzAYdPhZ1rYWj6m1Guw6Y
qRB17aFPYDY3EddSNNZaIbPJedTMyeIwBKvEG/2zEKdran/n66FTeggaVFA8E1Cg
GSRQMd8iFdy0JTtF0gvy6bc/XFCWpWCktotYOstUHJwu5H0CAwEAAaNQME4wHQYD
VR0OBBYEFIdwKfgYtpTmCOiZm+LbeQ3waXS/MB8GA1UdIwQYMBaAFIdwKfgYtpTm
COiZm+LbeQ3waXS/MAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBALoZ
F44vBwzutNPRN3Mxsc1aRGbEFbybqejpVcARMQcPu1IMc1IWQInSlqjeB+Wvdy7U
H69EIrtQiID2ylLkBItGuEUZNog5KXYWLF7xWpeOPKpMjIZ/GDRBOYlg7rrf1BLE
NElPpkqjvgs6o8Sn7Gz2dIK79aoxDDpyDw+lNqoMpumiuFh0yI1JlJhX/4c1lLYF
AafcHH/Y0eKBWW5pbv8AjwuUKr06tDZNkUHhB/V1lgrkl830l8F9t/vNzzp3LggE
zjZ3xJ9oP6KTryIDN1FmfPwMMhqiT/Ue7ZjB3eXFTfIgr9tcBAO2/RoVLiFadOq1
jrqPb+Z7W19U4sZ+jZ8=
-----END CERTIFICATE-----