OPENSSL AND POST-QUANTUM OPENSSL

[Group Assignment - 2]

CS6/745: Modern Cryptography

Submitted to: Dr. Yuliang Zheng

Submitted by:

1. Joyanta Mondal (jmondal)
2. Shouzab Khan (Skhan6)
3. Pooja Srinivas (psriniva)
4. Rajendra Mohan (rnavulur)

March 06, 2024

INTRODUCTION:

Transport Layer Security (TLS) protocol is a foundational element in ensuring the confidentiality and integrity of data in transit across networks. OpenSSL, the most significant library for implementing TLS, has long been the standard for securing communications on the internet. However, with the emerging threat of quantum computers capable of breaking traditional cryptographic algorithms, the imperative for quantum-resistant cryptography has never been more pressing.

In response to this emerging threat, the Open Quantum Safe project has introduced OQS-OpenSSL, an innovative extension of the OpenSSL library that incorporates post-quantum (PQ) cryptographic capabilities.

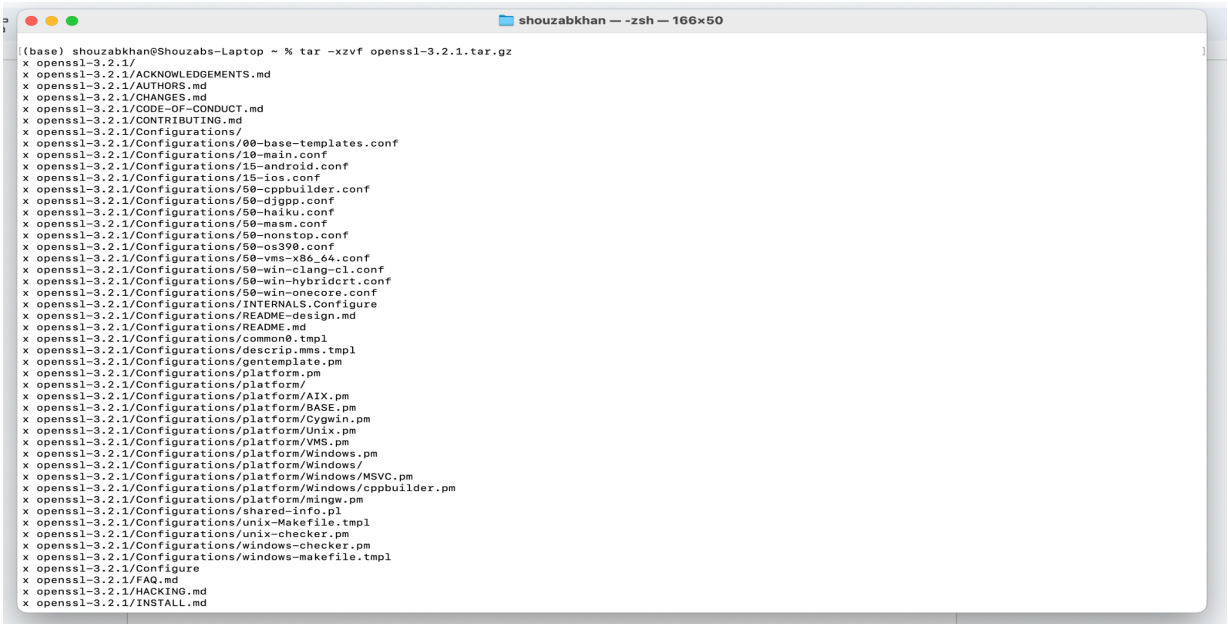
In this report, we explore the integration and application of both OpenSSL and OQS-OpenSSL. We use Mac devices to integrate these.

Part I. OpenSSL:

Download and Extract the OpenSSL Source Code:

First of all, we check on our computer which version is already installed with us. We have to use the command "openssl version" to see the version. It initially shows the "OpenSSL 1.1.1q 5 Jul 2022" version, but for this task, we update the openssl to its latest version. For that, we use the command
"wget https://www.openssl.org/source/openssl-3.2.1g.tar.gz"
After running this command, openssl in the computer updates to version 3.2.1.

Now when it is updated, we have to extract the source code tarball before running the command "./config"
For that, we run the command
"tar -xzvf openssl-3.2.1.tar.gz"



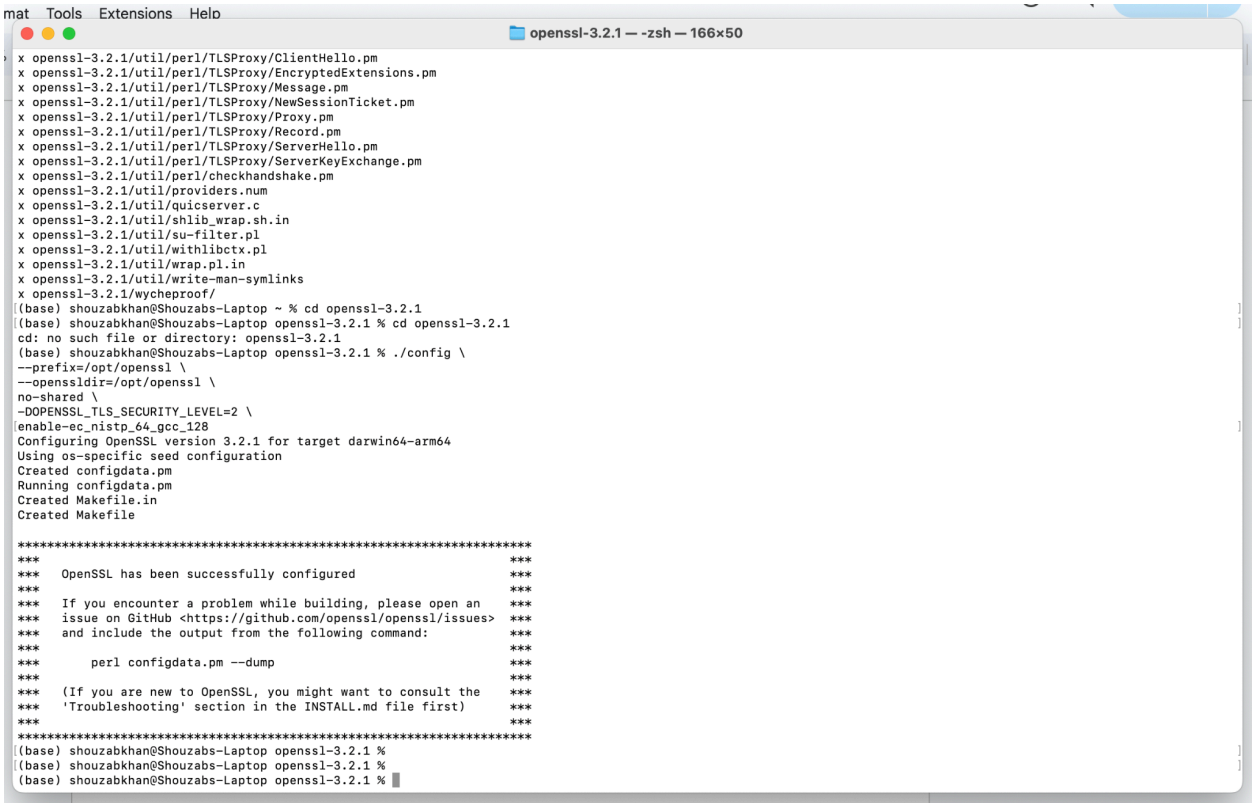
Now after that, we change directory to the extracted OpenSSL directory:

```
(base) shouzabkhan@Shouzabs-Laptop ~ % cd openssl-3.2.1
(base) shouzabkhan@Shouzabs-Laptop openssl-3.2.1 % cd openssl-3.2.1
```

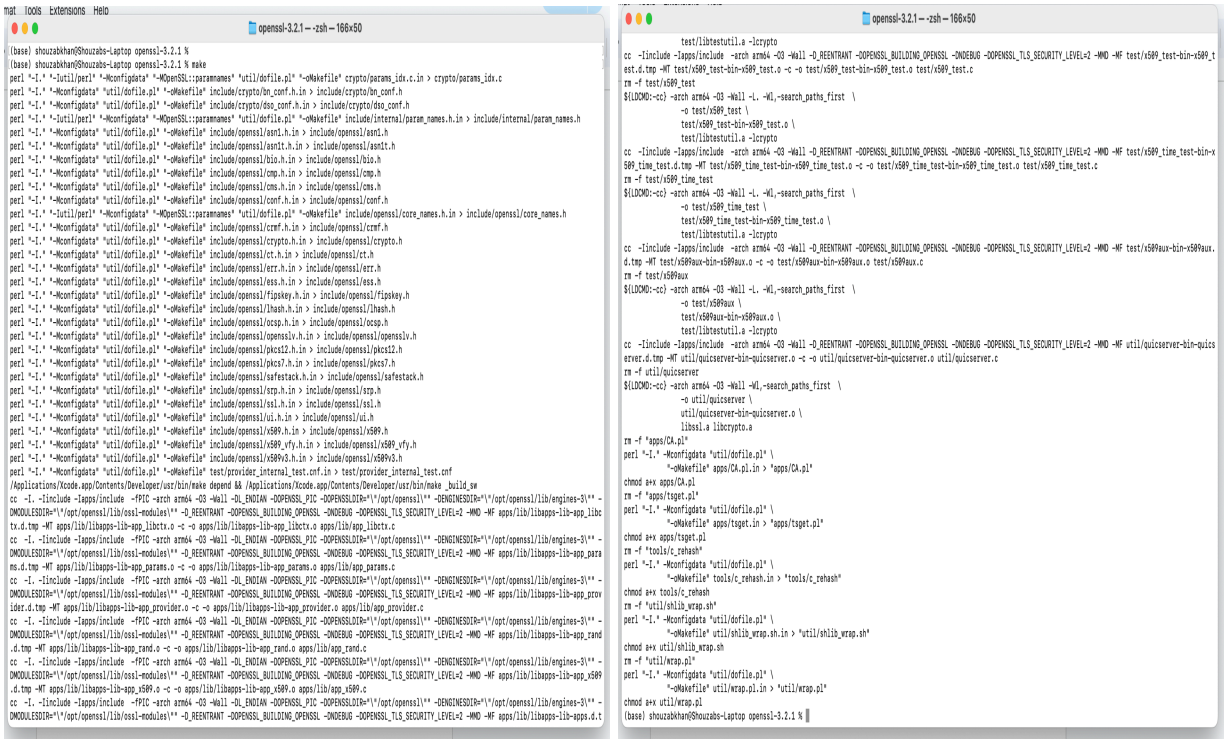
Till this step, we complete downloading and extracting the OpenSSL source code. After this step we have to configure, build and install OpenSSL.

Configure, Build and Install OpenSSL:

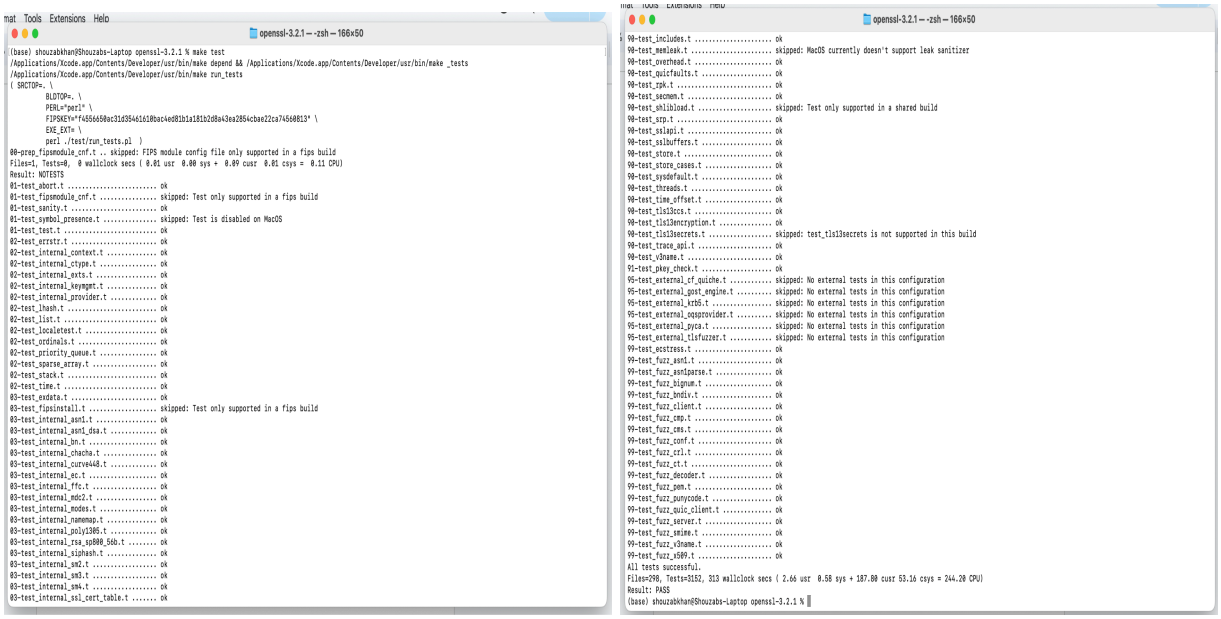
Now, we configure OpenSSL by running the `./config`. By this, OpenSSL will be successfully configured as shown in the screenshot of the terminal below.



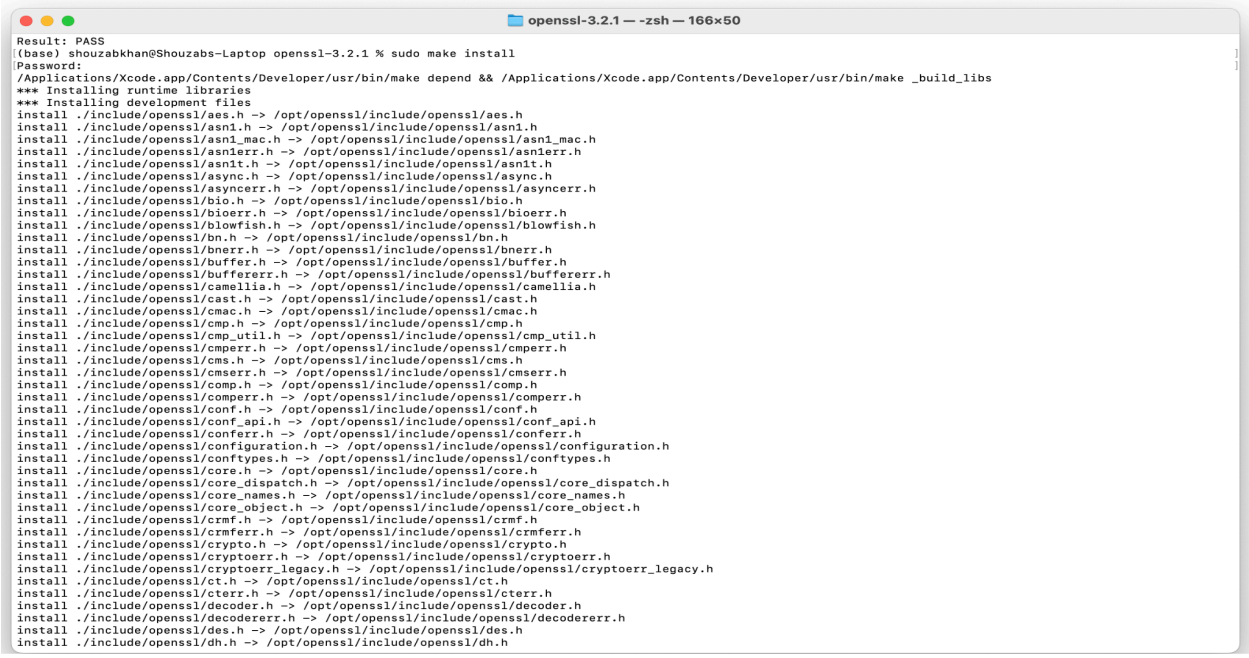
Now when the OpenSSL is configured, we have successfully built and installed OpenSSL. After giving the command `"make"` in the terminal, it starts building OpenSSL.
Below the screenshot are showing the execution of command `"make"`



Now we run tests to ensure the build is successful. For that, we run the command `"make test"`.

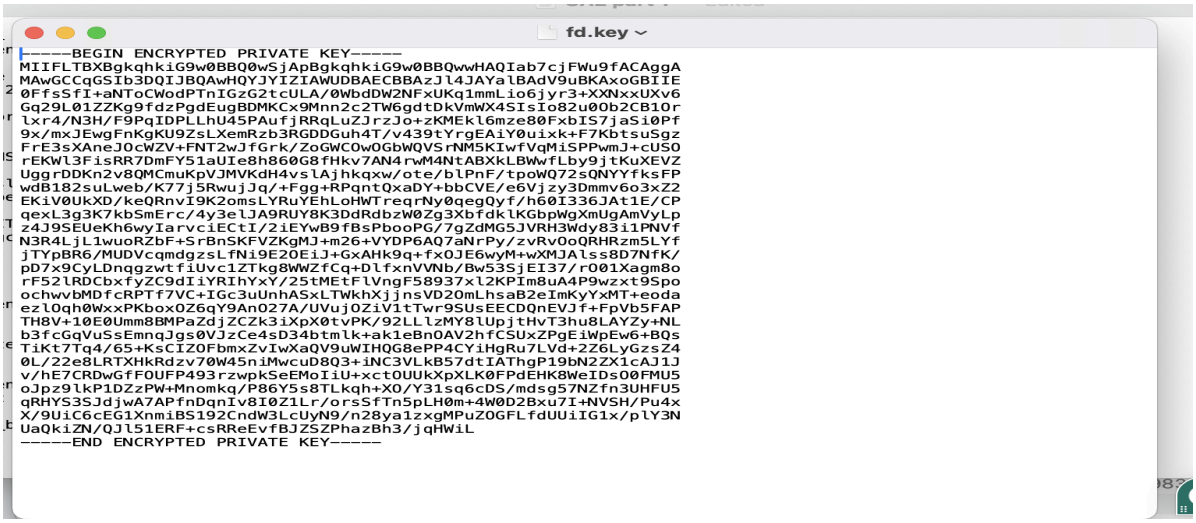


We can see that the result is passed when we run the tests. Now, the final step is to install OpenSSL. We put the command `"sudo make install"` after that it will require the password of your computer and it will start installing OpenSSL. Just shown below in the screenshot.



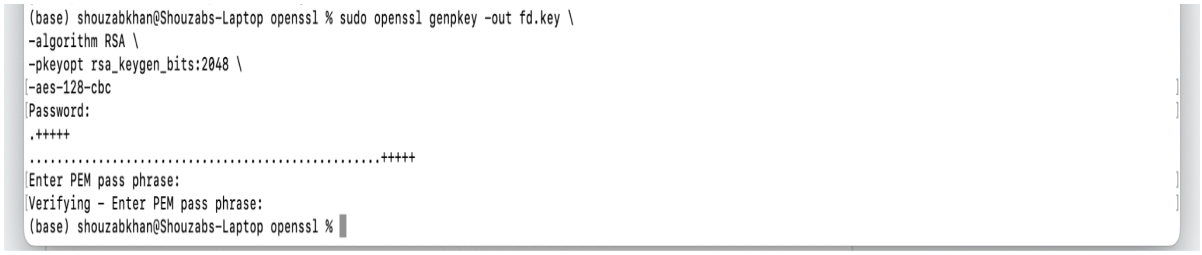
Generate public - private key pairs and certificates:

Now we generate an RSA private key with a key size of 2048 bits and encrypt it using AES-128. The key is saved in a file named `fd.key`. We also protect it with a password called PEM phrase pass.



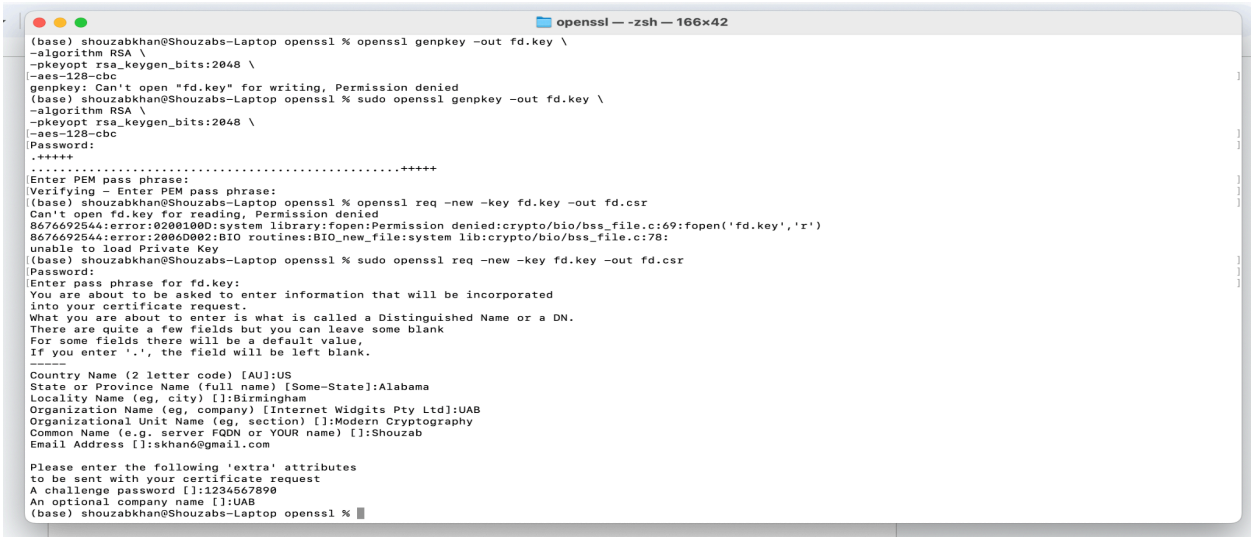
Now, we generate the public key so for that we run this command below:

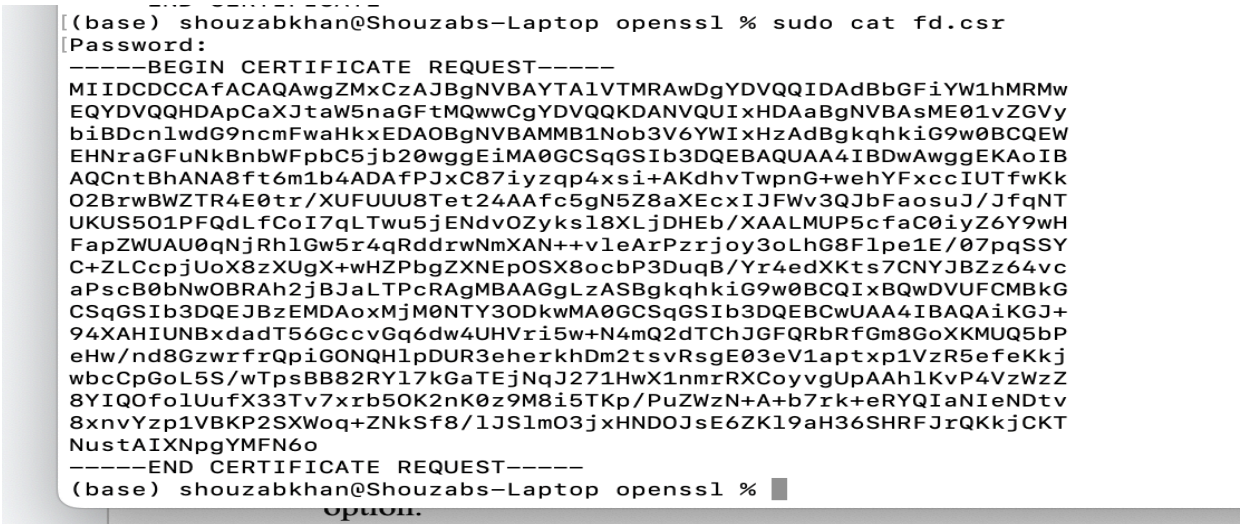
```
openssl pkey -in fd.key -pubout -out fd-public.key
```



- **Generating Certificate Signing Request**

Now when the key is generated, we request for a certificate signing that key using the command `"openssl req"`.
By this command, the terminal will ask for different questions for signing the certificate like PEM pass phrase, country name, state name, locality name, Organization name, Organization unit name, common name, email address, challenge password and optional company name. These were the questions through which the certificate will be signed.





Generating Self Signed Certificate:

Now if we want to self-sign the certificate, we can use the `openssl req` command with the `-x509` option.

The difference between these two was that, in CSR, they asked for the challenging password and optional company name but in self-signed certificates(CRT) they didn't ask for these two.

```
(base) shouzabkhan@Shouzabs-Laptop openssl % sudo openssl req -x509 -new -key fd.key -out fd.crt
Password:
Enter pass phrase for fd.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]:AL
Locality Name (eg, city) []:BHM
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UAB
Organizational Unit Name (eg, section) []:CS
Common Name (e.g. server FQDN or YOUR name) []:Skhan
Email Address []:skhan6@gmail.com
(base) shouzabkhan@Shouzabs-Laptop openssl %
```

```
(base) shouzabkhan@Shouzabs-Laptop openssl % sudo cat fd.crt
-----BEGIN CERTIFICATE-----
MIIDyTCCArGgAwIBAgIUezDQVqfShoiCx3S8AoB0bVbrlE4wDQYJKoZIhvcNAQEL
BQAwdDELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAkFMMQwwCgYDVQQHDANCSE0xDDAK
BgNVBAoMA1VBQjELMAkGA1UECwwCQ1MxDjAMBgNVBAMMBVNraGFuMR8wHQYJKoZI
hvcNAQkBFhBza2hhbjZAZ21haWwuY29tMB4XDTI0MDMxMDE3MTIzNFoXDTI0MDQw
OTE3MTIzNFowdDELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAkFMMQwwCgYDVQQHDANC
SE0xDDAKBgNVBAoMA1VBQjELMAkGA1UECwwCQ1MxDjAMBgNVBAMMBVNraGFuMR8w
HQYJKoZIhvcNAQkBFhBza2hhbjZAZ21haWwuY29tMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAp7QYQDQPH7eptW+AAwHzycQvO4ss6qeMbIvgCnYb08KZ
xvsHoWBcXHCFE38CpDtga8AVmU0eBNLa/11BVFFPE3rduAAH3OYDeWfGlxHMSCRV
r90CWxWqLLifyX6jU1ClEuTtTxUHS3wqCO6i08LuYxDXbzmcpLJfFy4wxxG/1wAC
zFD+XH2gtIsmemPcBxWqWVlAFNKjY0YZRsOa+KkXXa8DZlwDfvr5XgKz8646Mt6C
4RvBZaXtRP9O6akkmAvmSwnKY1KF/M11IF/sB2T24GVzRKTkl/KHGz9w7qgf2K+H
nVyrbOwjWCQWc+uL3Gj7HAdGzcDgUQIdowSWi0z3EQIDAQABo1MwUTAdBgNVHQ4E
FgQU0JMN2F84TsSiWfNsRAy7yjcZc5QwHwYDVR0jBBgwFoAU0JMN2F84TsSiWfNs
RAy7yjcZc5QwDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAP20/
ntK6BZ3OB1f8B2Pd1qbIytbiJG9VrxTHlbPhgVn6UPVy+cTq+h4/3zBYVkeUZgzR
PKMCYtWjhcG/lO0R7v5s0OaPHuCV8AKY5UNH+A+Aa7FdqUJPX5ueNs81jb1hj/q1
NWbLdohRyfpBCvSPF/d34/tbaF9IQkKLz3YBnDw7e3AT3oD1TsZiOAYlte3uBAyG
DBVo0SqdVHDcTVjkpkIzhI7FaT7BWe7Sersjk0nyqkUcqObuuOUEdX1MNSBS9HV6
OVfeVzdnvzhQ/tnSmL88+gmjJstPRSAt/oOTWYv/EyMRLIhdoL4FDoaW31OK5JrL
hJoBwZFt79hyG06V3g==
-----END CERTIFICATE-----
(base) shouzabkhan@Shouzabs-Laptop openssl %
```

Create a private certification authority:

Root CA Configuration:

We include some basic information in a file to Root Certification Authority Configuration. The file name is set to be `"root-ca.conf"`. The first part of the configuration file contains some basic CA information, such as the name and the base URL, and the components of the CA's distinguished name. Because the syntax is flexible, information needs to be provided only once.

```
(base) shouzabkhan@Shouzabs-Laptop openssl % nano root-ca.conf
(base) shouzabkhan@Shouzabs-Laptop openssl % sudo nano root-ca.conf
Password:
(base) shouzabkhan@Shouzabs-Laptop openssl % cat root-ca.conf
# CA information
[default]
name                    = root-ca
domain_suffix           = www.cnn.com

[ca_dn]
countryName             = "US"
organizationName        = "UAB"
commonName              = "Shouzab"

[ca_default]
home                    = .
database                = $home/db/index
serial                  = $home/db/serial
crlnumber               = $home/db/crlnumber
certificate             = $home/$name.crt
private_key             = $home/private/$name.key
RANDFILE                = $home/private/random
new_certs_dir           = $home/certs
unique_subject          = no
copy_extensions         = none
default_days            = 3650
default_crl_days        = 365
default_md              = sha256
policy                  = policy_c_o_match

[policy_c_o_match]
countryName             = match
stateOrProvinceName     = optional
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied

[req]
default_bits            = 4096
encrypt_key             = yes
default_md              = sha256
```

Root CA Directory Structure:

Now we create a new Root CA directory Structure, this structure includes directories for certificates, a database, and private keys, as well as the initialization of some necessary files.
By using the commands, we make the directories in the db directory just like shown in the below screenshot "Index" and "Serial".

```
(base) shouzabkhan@Shouzabs-Laptop root-ca % ls
certs   db      private
(base) shouzabkhan@Shouzabs-Laptop root-ca % touch db/serial
chmod u+w db/serial
touch: db/serial: Permission denied
chmod: db/serial: No such file or directory
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo touch db/serial
chmod u+w db/serial
Password:
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo openssl rand -hex 16 > db/serial
zsh: permission denied: db/serial
(base) shouzabkhan@Shouzabs-Laptop root-ca % openssl rand -hex 16 > db/serial
zsh: permission denied: db/serial
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo touch db/serial
chmod u+w db/serial
(base) shouzabkhan@Shouzabs-Laptop root-ca % ls
certs   db      private
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo openssl rand -hex 16 > db/serial
zsh: permission denied: db/serial
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo touch db/serial
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo chmod 600 db/serial
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo openssl rand -hex 16 > db/serial
zsh: permission denied: db/serial
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo sh -c 'openssl rand -hex 16 > db/serial'
(base) shouzabkhan@Shouzabs-Laptop root-ca % echo 1001 > db/crlnumber
zsh: permission denied: db/crlnumber
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo echo 1001 > db/crlnumber
zsh: permission denied: db/crlnumber
(base) shouzabkhan@Shouzabs-Laptop root-ca % sudo sh -c 'echo 1001 > db/crlnumber'
(base) shouzabkhan@Shouzabs-Laptop root-ca % ls
certs   db      private
(base) shouzabkhan@Shouzabs-Laptop root-ca % ls -l
total 0
drwxr-xr-x  2 root  wheel   64 Mar 10 16:49 certs
drwxr-xr-x  5 root  wheel  160 Mar 10 17:17 db
drwx------  2 root  wheel   64 Mar 10 16:49 private
(base) shouzabkhan@Shouzabs-Laptop root-ca % certs/ db/ private/
zsh: permission denied: certs/
(base) shouzabkhan@Shouzabs-Laptop root-ca % cd db
(base) shouzabkhan@Shouzabs-Laptop db % ls
crlnumber       index           serial
(base) shouzabkhan@Shouzabs-Laptop db %
```

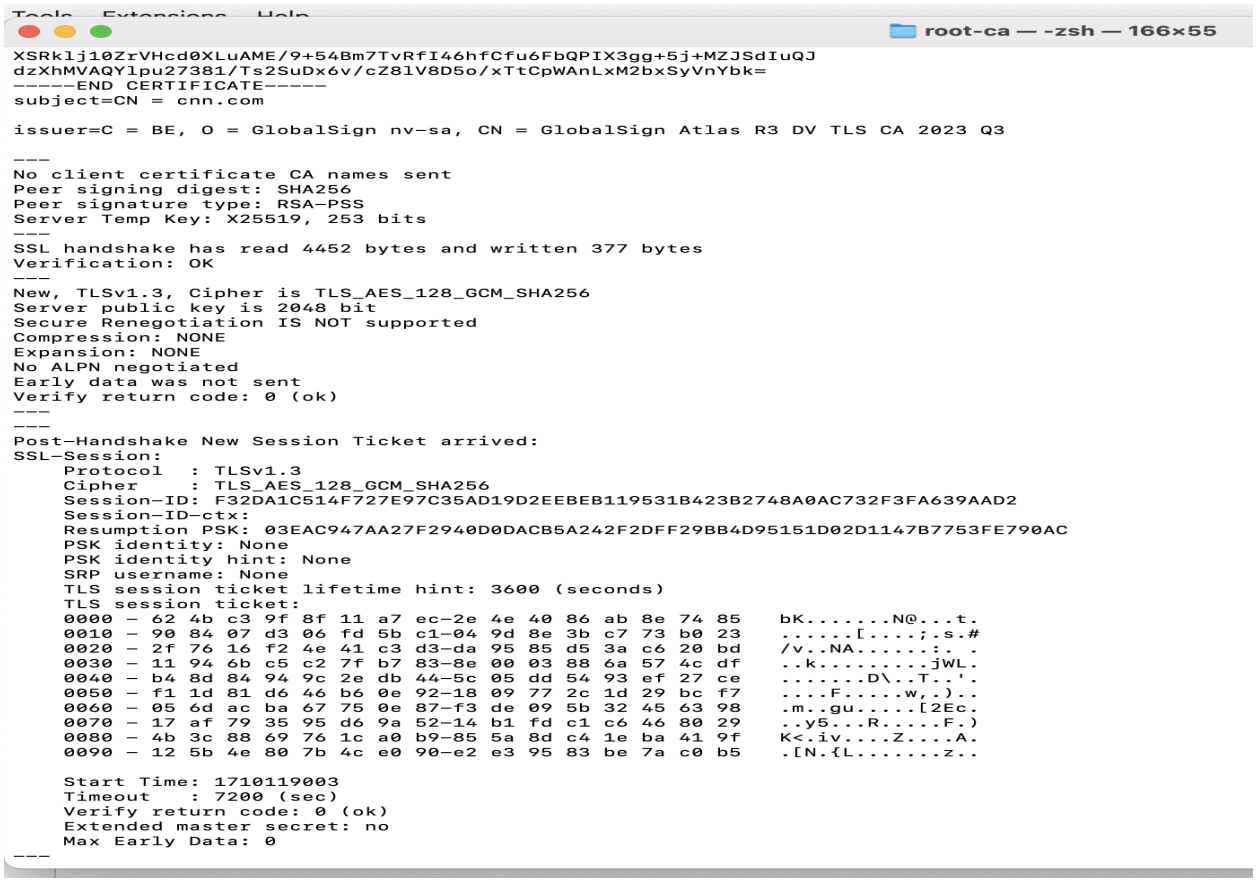
Root CA Generation:
Now, we generate the root certificate authority private key and certificate.

```
(base) shouzabkhan@Shouzabs-Laptop openssl % sudo openssl req -new \
    -config root-ca.conf \
    -out root-ca.csr \
    -keyout private/root-ca.key
Password:
Generating a RSA private key
.....................................+++
.....................................................................+++
writing new private key to 'private/root-ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
(base) shouzabkhan@Shouzabs-Laptop openssl %
```

Test with OpenSSL:
By completing all the steps above from step 1 to step 3 in our step 4 is to test any domain with OpenSSL to see if we can get the server certificate of not so by executing the command `"openssl s_client -connect www.cnn.com:443"` by this you can see in the screenshot below that we actually get the server certificate. It indicates that the whole process was successfully executed.


root-ca — openssl s_client -connect www.cnn.com:443 — 166×55
(base) shouzabkhan@Shouzabs-Laptop root-ca % openssl s_client -connect www.cnn.com:443
CONNECTED(00000005)
depth=2 OU = GlobalSign Root CA - R3, O = GlobalSign, CN = GlobalSign
verify return:1
depth=1 C = BE, O = GlobalSign nv-sa, CN = GlobalSign Atlas R3 DV TLS CA 2023 Q3
verify return:1
depth=0 CN = cnn.com
verify return:1

Certificate chain
0 s:CN = cnn.com
i:C = BE, O = GlobalSign nv-sa, CN = GlobalSign Atlas R3 DV TLS CA 2023 Q3
1 s:C = BE, O = GlobalSign nv-sa, CN = GlobalSign Atlas R3 DV TLS CA 2023 Q3
i:OU = GlobalSign Root CA - R3, O = GlobalSign, CN = GlobalSign
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIKqDCCCZCgAwIBAgIQAZ3dCTUFVNcaZ4TM/m6DFTANBgkqhkiG9w0BAQsFADBY
MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEuMCwGA1UE
AxMlR2xvYmFsU2lnbiBBdGxhcyBSMyBEViBUTFMgQ0EgMjAyMyBRMzAeFw0yMzA5
MTIxOTM4MDVaFw0yNDEwMTMxOTM4MDRaMBIxEDAOBgNVBAMMB2Nubi5jb20wggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDsZniL9RpV7hDYPJvS4TGa39w5
BLHGsPhi4lV4HVtyIme0/NMMmszIeNoY+aaDSM2dn0gw29GIq1prZSAQK8BgDU6a
otU5mWG8J+xABnn75DQ1BHjXZFl4EfjL4mIhMaVY34O+0wG06owvFDUgxRzYnwlb
y6WEJfTRyv70MF6EIq0zZxW2cMgfyuq8ZEtgYddSr4I/2/xVxACBUDFYNqYbr9AR
qmJKvzglrSYULaBJ84oY3RnBnDCVUkMW3qYT1mIDop+Jz4wLyMyvHq0QA0wY/BhI
ByhJTkdQy7xH2N8O2MohQmaVo6x6w01cqsZyIHND1JSL3lAJiMtU8aMl3+edAgMB
AAGjggeyMIIHrjCCBGcGA1UdEQSCBF4wggRaggdjbm4uY29tgg0qLmFwaS5jbm4u
Y29tggwqLmFwaS5jbm4uaW+CHSouYXBpLmVsZWN0aW9udHJhY2tlci5jbm4uY29t
ghYqLmFwaS5wbGF0Zm9ybS5jbm4uY29tghAqLmFyYWJpYy5jbm4uY29tghQqLmFy
dGVtaXMudHVybmVyLmNvbYIPKi5ibG9ncy5jbm4uY29tghgqLmNsaWVudC5hcHBs
ZXR2LmNubi5jb22CCSouY25uLmNvbYIIKi5jbm4uaW+CDyouY25uYXJhYmljLmNv
bYIOKi5jbm5tb25leS5jb22CESouY25ucG9saXRpY3MuY29tghYqLmNvbmZpZy5v
dXR0dXJuZXIuY29tghEqLmRhdGEuYXBpLmNubi5pb4IRKi5lZGl0aW9uLmNubi5j
b22CFyouZWRpdGlvbi5pLmNkbi5jbm4uY29tghwqLmVkaXRpb24uc3RhZ2UubmV4
dC5jbm4uY29tgh0qLmVkaXRpb24uc3RhZ2UyLm5leHQuY25uLmNvbYIdKi5lZGl0
aW9uLnN0YWdlMy5uZXh0LmNubi5jb22CEyouZWxlY3Rpb25zLmNubi5jb22CGSou
ZWxlY3Rpb250cmFja2VyLmNubi5jb22CDCouZ28uY25uLmNvbYIPKi5pLmNkbi5j
bm4uY29tghYqLm1hcmtldHMubW9uZXkuY25uLmlvgg8qLm1vbmV5LmNubi5jb22C
DioubmV4dC5jbm4uY29tghYqLm9kbS5wbGF0Zm9ybS5jbm4uY29tgg8qLm91dHR1
cm5lci5jb22CEioucGxhdGZvcm0uY25uLmNvbYIfKi5zZWN0aW9uLWNvbnRlbnQu
bW9uZXkuY25uLmNvbYIUKi5zdGFnZS5uZXh0LmNubi5jb22CFSouc3RhZ2UyLm5l
eHQuY25uLmNvbYIVKi5zdGFnZTMubmV4dC5jbm4uY29tghEqLnN0ZWxsYXIuY25u
LmNvbYIUKi50ZXJyYS5uZXh0LmNubi5jb22CECoudHJhdmVsLmNubi5jb22CEyou
d3d3LmkuY2RuLmNubi5jb22CD2FwaS5ldHAuY25uLmNvbYIWY2xpZW50LmFwcGxl
dHYuY25uLmNvbYINY25uYXJhYmljLmNvbYIMY25ubW9uZXkuY29tgg9jbm5wb2xp
dGljcy5jb22CDWRjZmFuZG9tZS5jb22CHGdyYXBocWwudmVydGljYWxzLmFwaS5j
bm4uaW+CFGkuY2RuLnRyYXZlbC5jbm4uY29tghlwcmV2aWV3LmRldi5tb251eS5j
bm4uY29tghhwcmV2aWV3LnFhLm1vbmV5LmNubi5jb22CGXByZXZpZXcucmVmLm1v
bmV5LmNubi5jb22CG3ByZXZpZXcudHJhaW4ubW9uZXkuY25uLmNvbYIacHJldml
dzIucmVmLm1vbmV5LmNubi5jb22CD3VuZGVyc2NvcmVkLmNvbTAOBgNVHQ8BAf8E
BAMCBaAwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMB0GA1UdDgQWBBT9
Fy8eFhWRk9UjmQVNdVD8lZEhFTBXBgNVHSAEUDBOMAgGBmeBDAECATBCBgorBgEE
AaAyCgEDMDQwMgYIKwYBBQUHAgEWJmh0dHBzOi8vd3d3Lmdsb2JhbHNpZ24uY29t
L3JlcG9zaXRvcnkvMAwGA1UdEwEB/wQCMAAwgZ4GCCsGAQUFBwEBBIGRMIGOMEAG



Getting the output as expected indicates that the SSL handshake process has completed successfully.

Part II. OQS-OpenSSL:

Step 1: Installation and build of OQS OpenSSL

1) On MacOS, firstly, we install brew package manager for simplified installation and managing software.

2) Then use, `brew install cmake ninja libtool openssl@1.1`

The above command is for brew package manager on macOS. It installs four software packages commonly used for building and managing software projects. This command equips the system with the necessary tools to build and manage software projects that might involve cryptography or rely on external libraries.

3) *Command*:

```
git clone --branch OQS-OpenSSL_1_1_1-stable
https://github.com/open-quantum-safe/openssl.git <OPENSSL_DIR>
```

This command uses the git tool to clone a specific version of a software repository. In simpler terms, this command downloads the source code for the `OQS-OpenSSL_1_1_1-stable` branch of the open-quantum-safe/openssl repository and places it in the directory specified.

4) *Command*:

```
git clone --branch main
https://github.com/open-quantum-safe/liboqs.git
cd liboqs
mkdir build && cd build
cmake -GNinja -DCMAKE_INSTALL_PREFIX=<OPENSSL_DIR>/oqs ..
ninja
ninja install
```

- Download the source code for the liboqs library with PQC algorithms.
- Configure and build the library using cmake and ninja.
- Install the built library and potentially executables into a specific directory within OpenSSL installation.
- By completing these steps, the liboqs library with its PQC algorithms will be available on the system.

5) *Command*:

```
./Configure no-shared darwin64-arm64-cc
make -j
```

This command sequence configures the build process to create a static library for an ARM64-based macOS system and then uses make to compile the source code into the library.

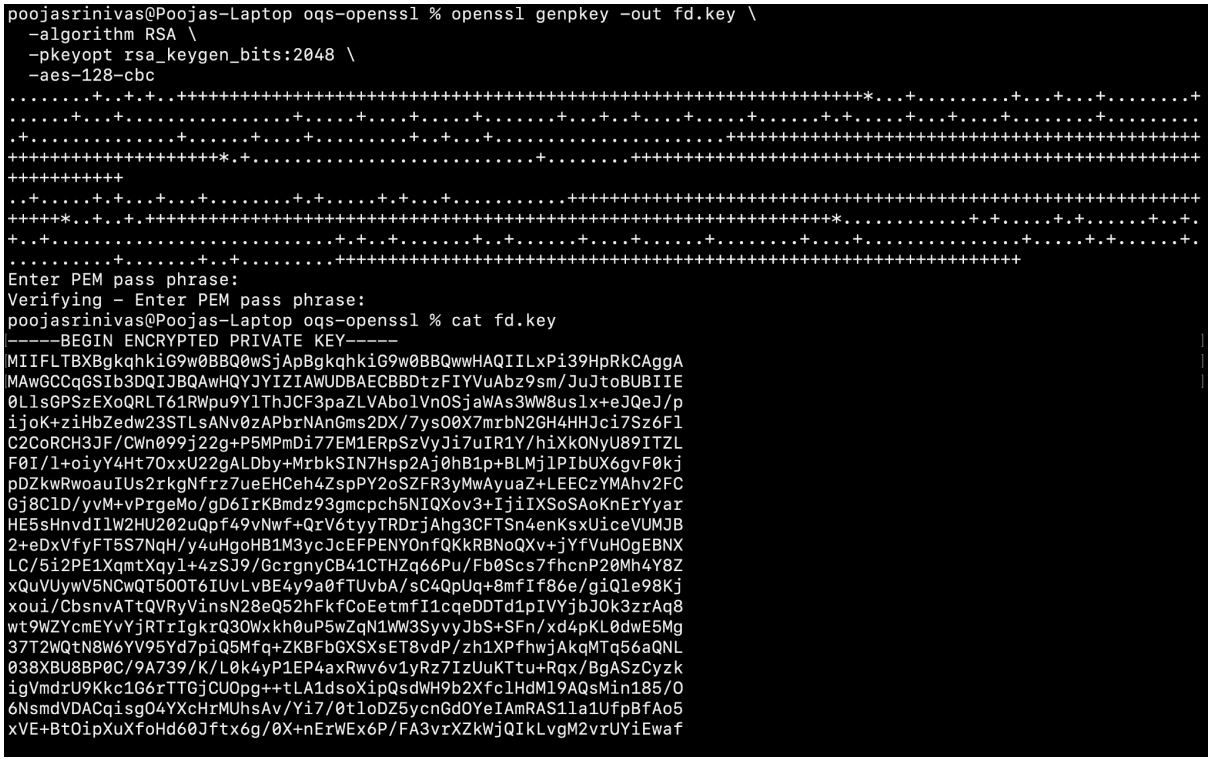
Step 2: Generate PQ public-private key pairs and certificates

1) Private Key: The below mentioned command generates a 2048-bit RSA private key and stores it in the file named fd.key.

Command for private key generation:

```
openssl genpkey -out fd.key \
  -algorithm RSA \
  -pkeyopt rsa_keygen_bits:2048 \
  -aes-128-cbc

Enter PEM pass phrase: (ex: hello)
Verifying - Enter PEM pass phrase: (ex: hello)
cat fd.key
```



2) Public key generation Command:

```
openssl pkey -in fd.key -pubout -out fd-public.key
Enter pass phrase for fd.key:
cat fd-public.key
```

These commands extract the public key from a private key and display the public key's contents. When running this command, will be prompted to enter the passphrase associated with the private key in fd.key (if a passphrase was set during its creation). It provides the correct passphrase for successful extraction.

3) Certificate Generation Command:

```
openssl req -new -key fd.key -out fd.csr
openssl req -text -in fd.csr -noout
```

This command generates a Certificate Signing Request (CSR), a document used to request a digital certificate from a Certificate Authority (CA) and then display its contents for verification.

```
poojasrinivas@Poojas-Laptop oqs-openssl % openssl req -text -in fd.csr -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: ST = Alabama, L = Birmingham, O = uab
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d2:e8:dc:8d:fa:1f:68:ff:a8:6f:fa:bd:2f:b0:
                    ea:8f:d0:f1:72:88:16:e4:04:2d:7f:d9:b3:f3:5f:
                    f5:d9:97:e7:c6:be:56:2f:d3:82:cd:8b:a8:e5:1b:
                    26:d9:cc:76:3e:45:8b:b9:8c:6c:d4:fe:84:f2:00:
                    83:74:54:f5:ec:77:16:92:62:a9:a7:e6:6c:95:55:
                    91:a2:97:fc:9f:8c:c7:be:48:92:30:34:d8:d7:af:
                    6b:e2:a6:b7:30:8f:62:37:93:5d:e4:9a:95:71:85:
                    7e:43:1d:a0:2b:18:f2:29:2b:e8:f4:59:26:77:00:
                    85:42:1c:29:73:67:b9:40:a2:77:16:af:a1:15:40:
                    d4:0c:db:ac:d8:63:bc:06:7b:48:a4:60:14:2b:2d:
                    3f:3e:02:74:08:d0:67:94:c0:aa:65:2f:da:2b:dd:
                    48:7c:f4:8e:6b:cd:f3:42:65:eb:ee:3b:09:3d:ea:
                    87:73:b7:3a:39:06:f6:6f:b8:a8:aa:e8:3d:a8:94:
                    d6:7d:e9:32:88:73:60:93:7f:9d:9c:f4:71:a1:cc:
                    37:d8:6f:f9:62:31:3f:92:6c:09:d4:9a:79:d8:65:
                    73:45:ee:ea:67:44:1e:b3:7a:17:9c:0c:0a:cb:29:
                    09:75:c9:b0:6b:06:c0:20:64:53:c3:38:a2:4a:15:
                    a0:75
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        04:c7:ec:24:27:ce:ca:92:ee:19:7f:3c:7d:9e:2f:99:4c:94:
        ee:10:29:ef:f5:83:63:5f:78:db:a6:d2:fc:88:fb:83:0a:21:
        19:b1:a1:14:41:23:a3:ec:06:88:ca:75:0d:06:0d:62:8f:36:
```

```
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        04:c7:ec:24:27:ce:ca:92:ee:19:7f:3c:7d:9e:2f:99:4c:94:
        ee:10:29:ef:f5:83:63:5f:78:db:a6:d2:fc:88:fb:83:0a:21:
        19:b1:a1:14:41:23:a3:ec:06:88:ca:75:0d:06:0d:62:8f:36:
        ce:e1:e4:89:e9:de:4f:67:3a:fe:f4:21:fd:bd:9c:24:56:f1:
        bb:10:04:23:97:85:13:c7:60:36:31:67:9e:83:df:3d:dd:ba:
        24:4d:98:54:66:2b:e1:86:8f:af:61:c6:f4:db:0e:be:76:fb:
        1c:62:cf:6e:0a:94:76:ea:02:56:8f:cd:33:5e:e3:8a:98:45:
        8b:22:18:dd:c0:26:5b:e7:ed:a9:ae:9e:48:80:e8:c4:e5:92:
        59:aa:8e:7f:73:c5:22:6c:0a:5b:43:f2:3e:a8:1a:ac:dd:eb:
        cb:66:82:3b:67:3a:e9:dc:07:d3:8e:5e:80:ad:dc:b3:11:44:
        ff:28:fd:ad:03:8a:bb:95:c8:5d:6e:7c:28:39:a3:c6:30:d3:
        52:fe:95:ed:04:f9:94:02:e8:82:61:44:ed:7f:34:27:d4:7c:
        6a:b6:d2:5f:f8:00:c8:0c:99:b0:03:78:2b:56:cb:ee:6e:ea:
        f2:38:c0:d4:31:b8:95:a3:7e:b4:a3:59:53:b5:40:2e:c9:a1:
        03:77:35:c0
poojasrinivas@Poojas-Laptop oqs-openssl %
```