Public Key Cryptography in OpenSSL and Extended Public Key Cryptography in OpenSSL

HW6-7-CNS Sapienza

Hassaan Ahmed Qureshi - 1906852

December 19,2019

Contents

1 Introduction

This homework is consist of two parts, HW6 and HW7 combined. In HW6 we are going to generate a private and a public key using OpenSSL and then produce a signature using that private key. We will later try signing a simple generic file like a .txt file and then verify if its signed properly or not.

In order to provide authenticity to an electronic program or a file of any type, Digital Signature can be used to provide its authenticity. This signature is different to the one signing on a simple piece of paper and in order to verify that digital signature we need to be assured of two things, the first one being that the digital program has not changed because it is based on a cryptographic hash of the document which is attached to that document. The second one being the authentication of the person who signed it. i.e.Alice.

In practical way, what we are going to do later in this report is generating a private key using OpenSSL and them extracting it's corresponding public key. Both of these key pairs are encoded in base64 with respective sizes mentioned during the experimentation.

2 Generating the Public and the Private Key

In order to generate our Private using OpenSSL, we use the following command:

genpkey -out privkey.pem -algorithm rsa 2048

In result we get a private key file ending with extention .pem, Upon opening it we see text something like this:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,72FDE6D5D1ED418A

6JLJf4wemuKCHn1z7b2fDD1w+EM/8AoXogv7ls08oy2mRV1ySGcDwExFV5b2udV0
hXwb6X3oJCG0ak7OemY6Vq5JmpwHLSmkCcu5bO2aoyaVv9DFgwHukTujYNJpOF8f
H6Y5ZmFaldcleogM+0ENOcf07jbUVhFfIp+StMafDcrU8x4tVHnOhqyR7JVqXuVV
RNSd30hiPpFNVlznnI4GShLSSRiYNo/JO68h6nqunNoT/NJSLpqv2qBeiY7MDeWa
mq4RyBgy1Oa+75xbsbVLs0iNL0JqX7nNqIlYXt2E/mALQLe75JCBWWh7loBFv3MD
```

```
h3keQEjkOOmCwA32hPCbBpRa9bIu+tWrg645vi3HeL1r+T98JVsWRW+T9WkhsZqS
g/mrjuOmtKs7aHYIPCE3DVZSGNFAFCaYnXTX3vviKv4yyG92Uoly93LfzMwO2NwY
IPs9aCmquyZyZW/VWvzJPjjRr8FBNEmpBGl9O6gc+HiLKWmQ64fnX4R8YQ+NoFVb
VYRemWwAxtgl1Oxww54p3GOSYdga7AQyRAHBgKoEBBEOIhVHCspYLZkpSh6xBSqm
2imSIWoECrFiCHBsghKFwcLcFtnAB1xO9LDcl2emwjuhZFIgKdCHoBOM/xTmK1tc
zTkPBkXEB7xNzbCbszwl9Vxa3kI79+nU5oyAMpJNkWoPr8JuTWpOhPmGgjOGXmOB
eC5nCJ9fyBwnfZgZYbl0ncaut7yuTLVm0FZun2jLFC6XnMTyGMjTcBodkrzqk9l5
Ur+KGqlelG4IRRndoxjrpaqVGASd61gMU5DjQPbSdVbPJwZco8HmS+0vtlLM9Gyp
5ZsezMuYrysVdVaEOC3zvforXHVZpo3VDvdI0DHtkHclX+B17LDBEH+0SKMqV8Jm
qFc+yl0CCcaq1KglPCR+IJXu5U8M4f8s+COA+rltRGpCAag90gUdnqRSQdXsft3n
qpkt+llfm4xzO+wCklu9AoYVcwobIqSjIqAejZ79vP/zNfi1A2JUpIRCCzt1y96v
KTojLt5rakuSbNvtWTvKB3ih2G+U2dzqwN8qEESFClhBe4+oogcehzd9zgUKxwC6
T5vYNwVk5Gjt7rXy13MSbBnmU9piweLU5WtUlKpwGzP1tXxllh5tclbmw/cUX7ds
dyBL2PsRsz83A+E1AfkVJdLgVx5T4IXn4uQznUt2ftKzQceqHDW9VGchu8UXYb/k
PFyfDwY8dxFntjaPRLWkXDr2g0S53bua+I+IqjJaAD0zrAHFnijyL9cLzGMDvn0C
jDbLu9hTMY2YSbF2Scmz9dN3Qyb8RqDdL0HEZfePfLduTBmJgHNVSQXv/TPck0tY
Hz7Wn+9YSL9y2/CRK0OYfi0ssJIiZiBYmhOxmY1lTVenEhleRfLlGbku3hw+ZYtG
i1b17WvJGmt6/N2ZPI7jyFrLDgF3n+wOMDa++I4v1znveNSgM+rYizbYA/CA6FHJ
1+/bjVwdZA8/kaUpj11R1D7/RQqrYbDsEJ8k7MXVRbgPwiLCQR9mS4ZN296JVCLM
uJIA2FIVE7Z1tQ9/coLU51DKM8b3lQ5Uo91wlrtJsryMpI6Ys7BTh0SwXFEi2Xp/
-----END RSA PRIVATE KEY-----
```

In order to extract our Public Key from the corresponding Private key using OpenSSL, we use the following command:

openssl rsa -in privkey.pem -outform PEM -pubout -out pubkey.pem

In result we get a public key file also ending with extention .pem like private file, Upon opening it we see text something like this:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA565jyjs7Xzh7GOD/sytl
TkwTfdeNWjhwwkWm9IUxNTmExGdzqry9TB+XUg2VNqAvrub394cw1p+2KPODqrQn
5K5wRj78FZwDIdtkwgOomKBPbM9qf8yCWidvGXqgntUuvHQq6qtG4MhKwoNm+MLg
diW7tvvZs5b5ff5k3TEmoKBX1BySjy1YY8TYLlHeW/9p8MxBvx+qO9xx3/1NmiTM
79Bzew+9upIWgeR58mi4TvtV+5++nWURCLqbPYbqd3htP4N3ll5/Wp3bDc/dBmMR
Sdz5UAndiCJ9p6piVoDFE+WQH48vRU+pneOLqdnz+z1xSYYfh0HwcIx9ZgXImu5G
NQIDAQAB
-----END PUBLIC KEY-----
```

3 Self-Signing a simple file using our generated keys

Now that we have generated out public and private keys, we can perform a digital signature on a file let say a .txt file easily. To do so we use the following command:

openssl dgst -sha256 -sign privkey.pem -out sign.sha256 mytext.txt

Upon running this command, we get a sign.SHA256 signed file, which is not very readable so we convert it into a readable file we attach a simple .base64 to its name and run the following command:

openssl enc -base64 -in sign.sha256 -out sign.sha256.base64

```
WRD73qYAubsXoPEg6dBpLI862kEoieVYbmogA5dNLMt8iQVfxyMUdhCJ5YnNM3er
9NM2QBOCwrrKhOhj9zosbA4GvUDPtIpFqrON4j8EOJQZLfvEasAB+sLKAI2IWMOo
O3k5Si1aQH+u9vs9pP6nuSzX2kYBu2Arz+1x5BM8nHo8kVor74U/tnv/KBXO4fjZ
7mNOGLhUOFHdrl/MW192Z8j69gllA/yvVzEblA1FORrcyBpH3SIov3WqzoFBCvGD
dBK4zNaZtYc6p2ka3H4Rtt/AaaOmVQjVe3iYbgaCr+PZOPlUS5czUCeHWCA/8NbK
TiwLF1VUa/2Px/6vbtcmNQ==
```

4 Verification of the Self-Signed file

It is always good idea to do a simple verification as a part of confirmation of our digitally signature using our public key. The hash earlier generated is an important part of this step which verifies if the file we signed has changed over the course or not. To do this we use the simple command:

openssl enc -base64 -d -in sign.sha256.base64 -out sign.sha256

openssl dgst -sha256 -verify pubkey.pem -signature sign.sha256 client

While the first file decodes our base64 signature file, the second one verifies our signature. We get an output in our command line as follows:

"Verified OK" or "Verified failure"

5 Conclusion

Thus in the above simple steps we were able to do a simple cryptographic Digital Signing of a file using OpenSSL.

6 HW7 Introduction

This part of the homework extends the previous part in which we digitally signed a simeple text file using OpenSSL Cryptographic tools. In HW7, We take our cryptographic experimentation to two virtual machines, one which is the CA (Certification Authority) and other machine who wants to get a document signed from the CA. The communication between these two VMs is done using netcat. Other VM will also ask for revocation of that certificate later on by generating its CRL.

7 Configuring our CA and making our host maching CA

I order to configure our CA, we need to configure some files at our root directory. These files are CA.pl and OpenSSL.cnf files. These files are already present there we just need to configure the CATOP directory where our CA files would be generated. We can do so by editing these files in terminal:

nano /usr/lib/ssl/misc/CA.pl

/usr/lib/ssl/openssl.cnf

Now once we have configured our files providing a directory, we create our CA. For this I have choosen my host machine as a CA. We can create our CA by running the following command:

/usr/lib/ssl/misc/CA.pl -newca

Upon running this command, an interactive session will begin and it will start asking us for the certificate information like CA name which we will leave blank and other information will follow like Country Name, Province name, Locality, etc.

```
CA certificate filename (or enter to create)

Making CA certificate ...
====
openssl req  -new -keyout /root/demoCA/private/cakey.pem -out /root/demoCA/careq.pem
```

```
Generating a RSA private key
.................................+++++
.....................................................+++++
writing new private key to '/root/demoCA/private/cakey.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) [AU]:IT
State or Province Name (full name) [Some-State]:LAZIO
Locality Name (eg, city) []:ROMA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Sapienza
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:hassaan
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
==> 0
====
====
openssl ca  -create_serial -out /root/demoCA/cacert.pem -days 1095 -batch -keyfile /r
Using configuration from /usr/lib/ssl/openssl.cnf
```

At this point it will ask us for a passkey, which we can provide on our choice or leave blank as well. however upon providing we willl need it in the rest of our process as well.

```
Enter pass phrase for /root/demoCA/private/cakey.pem:

Check that the request matches the signature
Signature ok
```

```
Certificate Details:
        Serial Number:
            33:de:36:b5:c5:3f:3a:8e:7b:81:ba:a9:30:af:48:13:80:2c:1c:10
        Validity
            Not Before: Dec 18 12:51:42 2019 GMT
            Not After : Dec 17 12:51:42 2022 GMT
        Subject:
            countryName               = IT
            stateOrProvinceName       = LAZIO
            organizationName          = Sapienza
            commonName                = hassaan
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                52:6C:5F:EC:44:6B:79:F8:C3:92:E1:D1:53:92:81:8C:18:24:A4:E9
            X509v3 Authority Key Identifier:
                keyid:52:6C:5F:EC:44:6B:79:F8:C3:92:E1:D1:53:92:81:8C:18:24:A4:E9

            X509v3 Basic Constraints: critical
                CA:TRUE
Certificate is to be certified until Dec 17 12:51:42 2022 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
==> 0
====
CA certificate is in /root/demoCA/cacert.pem
```

Now we are done with configuring and creating a CA. Our host machine is a CA at this point and to verify if it is configured correctly, we check whether we have all the required files or not. to do so, we go to our folder named demoCA which we provided in configuration files in the first step and see inside what is available, we get the following files and folders in our demoCA folder in root directory:

```
cacert.pem  certs  crlnumber  index.txt.attr  newcerts  serial
careq.pem   crl    index.txt  index.txt.old   private
```

Here we have cacert.pem which is the certificate, careq.pem which is the request for itself. We have index.txt which is the database and some

attributes in index.txt.attr and some other folders to store new certificates. Therefore we have our CA configured correctly and now we can sign certificate requests from other VM.

8 Configuring our VM and requesting CA to sign a certificate

In order to configure our VM, we create a folder at root directory similar to what we did while configuring CA. We then generate a new request using the OpenSSL command:

mkdir user

cd user/

openssl req -new -keyout privateUser.pem -out reqUser.pem

This command will create a new request which will contain the respective private and public keys in privateUser.pem file and put request in reqUser.pem. Upon entering this command it will start asking us for certificate request data again which upon providing we will have our request files generated:

```
Generating a RSA private key
.....................................................+++++
.........+++++
writing new private key to 'privateUser.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) [AU]:IT
```

```
State or Province Name (full name) [Some-State]:LAZIO
Locality Name (eg, city) []:ROMA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Sapienza
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:hassaan.com
Email Address []:

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

Now if we check inside user directory which is the folder we created earlier in this VM, we see these files generated:

```
privateUser.pem  reqUser.pem
```

Also it is to note that in our CA policy, it is clearly mentioned that the three attributes needs to be matched between CA and the requesting generating side in order to sign the certificate from CA, otherwise our certificate will not be signed these three attributes are i.e. countryName, stateOrProvinceName, organizationName. We can check this by looking inside openssl.config at our CA machine:

```
# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

Now the next step is to ask the CA to sign this request that we just generated at our VM. In order to do so, we need to have a communication between the two machines. To establish this connection I've used netcat for sending the required files to the CA asking for signing it. using netcat, we send the reqUser.pem file to the CA authority machine to sign it. We use the following commands:

on listening end we put our CA machine to listening mode:

```
nc -l -p 1234 > reqUser.pem
```

our CA will begin listening on port 1234. Furthermore, on our virtual machine we use the following command to send reqUser.pem file using IP and port provided by the CA machine:

```
nc -w 3 192.168.2.5 1234 < reqUser.pem
```

thus the file will be transferred to the CA and it will stop listening.

9 Getting the doument

Using the received reqUser.pem file from the VM, we sign the request at our CA machine using the following command:

openssl ca -in reqUser.pem

It will prompt us to ask the passkey again which we entered while configuration, upon providing which it will ask CA to verify details and sign the certificate: we will sign it as it matched the 3 attribute values which were mentioned in config file.

```
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            33:de:36:b5:c5:3f:3a:8e:7b:81:ba:a9:30:af:48:13:80:2c:1c:11
        Validity
            Not Before: Dec 18 13:32:34 2019 GMT
            Not After : Dec 17 13:32:34 2020 GMT
        Subject:
            countryName               = IT
            stateOrProvinceName       = LAZIO
            organizationName          = Sapienza
            commonName                = hassaan.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
```

```
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                92:5E:8D:1D:88:D6:08:B0:27:EC:DA:78:A3:5E:0A:56:33:47:45:CB
            X509v3 Authority Key Identifier:
                keyid:52:6C:5F:EC:44:6B:79:F8:C3:92:E1:D1:53:92:81:8C:18:24:A4:E9

Certificate is to be certified until Dec 17 13:32:34 2020 GMT (365 days)
Sign the certificate? [y/n]:y
```

Then it will ask us to update the signed certificate into database, we will say yes and it will commit in database the record of the signed request:

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            33:de:36:b5:c5:3f:3a:8e:7b:81:ba:a9:30:af:48:13:80:2c:1c:11
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=IT, ST=LAZIO, O=Sapienza, CN=hassaan
        Validity
            Not Before: Dec 18 13:32:34 2019 GMT
            Not After : Dec 17 13:32:34 2020 GMT
        Subject: C=IT, ST=LAZIO, O=Sapienza, CN=hassaan.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c8:3d:97:06:8f:21:8d:9c:58:1c:83:b3:a1:69:
                    4e:e5:9f:ab:38:37:c4:c5:0f:7e:4d:a7:8d:61:2c:
                    0b:af:33:09:3a:42:86:15:46:79:f5:82:ff:c9:a6:
                    19:29:a0:af:28:b2:4e:11:43:57:39:d4:e7:b8:4a:
                    30:58:ad:4b:8d:a1:a3:89:d9:ab:58:dc:aa:69:89:
                    ad:e4:d0:5e:ec:09:df:b5:6d:1f:2f:fd:ba:4e:b4:
                    34:7e:75:b9:0b:fc:85:1a:98:ce:16:3c:d8:58:b5:
                    a2:af:2f:87:f9:81:f3:bf:29:69:c5:23:c6:1f:10:
                    72:97:5d:f0:c4:8d:20:64:d2:9d:39:04:83:8f:db:
```

```
                    5d:d2:d0:9f:17:c1:ff:ac:a9:22:83:4e:b9:4a:a8:
                    8c:06:8e:2a:e5:91:61:38:4f:29:dd:7e:6a:5f:9f:
                    a7:54:75:b8:af:7c:4f:06:dc:b6:be:17:c6:6a:d9:
                    3a:01:27:69:18:36:d1:75:4a:19:12:3a:26:ca:1a:
                    7a:cd:d6:d5:c3:c9:a7:cd:14:94:9a:41:fa:3b:73:
                    b5:a8:09:75:15:1d:31:d7:b7:c1:a1:6d:b6:c5:d2:
                    6c:9a:64:b4:17:20:be:e1:61:98:92:38:8c:b8:64:
                    eb:b3:78:cc:00:a7:fd:98:13:11:e3:bd:5f:66:f3:
                    2c:63
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                92:5E:8D:1D:88:D6:08:B0:27:EC:DA:78:A3:5E:0A:56:33:47:45:CB
            X509v3 Authority Key Identifier:
                keyid:52:6C:5F:EC:44:6B:79:F8:C3:92:E1:D1:53:92:81:8C:18:24:A4:E9

    Signature Algorithm: sha256WithRSAEncryption
         bb:15:42:d4:05:5b:01:81:b6:de:90:5d:6b:08:4c:cf:e1:4f:
         82:47:e3:0c:b0:f6:f6:5a:fe:06:6e:37:bf:ce:af:ab:0c:0f:
         af:58:d5:f5:d1:f4:7f:7e:72:76:95:e0:32:04:c1:09:03:e9:
         f6:d7:65:be:ef:69:6d:ab:7e:ff:94:db:9d:bd:b6:ca:f4:5f:
         20:c6:33:ef:d7:b1:7d:8e:c6:4d:38:54:9e:47:99:63:8e:1f:
         8b:80:f6:7b:0f:75:1a:27:ac:de:ed:24:54:a6:65:c6:c1:34:
         58:da:d4:cf:67:3a:00:da:8a:9c:43:31:da:bf:6a:02:f6:d6:
         09:15:45:d9:9f:3a:d2:f2:9e:4d:b1:2f:e8:e2:f3:34:90:b8:
         d0:7a:77:9a:13:81:ae:20:30:8b:8f:57:5d:f0:94:d3:39:84:
         a3:c1:85:7f:cc:6c:ed:f8:b4:14:b8:c1:71:6e:97:b8:c8:4f:
         06:f4:f6:5b:9e:bb:1b:5a:14:7d:9a:25:62:51:9d:a3:a5:2d:
         99:54:5e:36:8d:bb:33:de:9c:bc:91:f3:b4:24:06:47:e5:a8:
         7d:9d:d6:23:14:8e:6a:f6:8d:3f:ca:20:af:07:c3:42:8b:08:
         87:cd:6a:be:12:35:71:07:23:d3:44:5b:f6:b8:cd:fb:7c:54:
         5b:58:8c:c2
-----BEGIN CERTIFICATE-----
MIIDkTCCAnmgAwIBAgIUM942tcU/Oo57gbqpMK9IE4AsHBEwDQYJKoZIhvcNAQEL
BQAwQjELMAkGA1UEBhMCSVQxDjAMBgNVBAgMBUxBWklPMREwDwYDVQQKDAhTYXBp
ZW56YTEQMA4GA1UEAwwHaGFzc2FhbjAeFw0xOTEyMTgxMzMyMzRaFw0yMDEyMTcx
```

```
MzMyMzRaMEYxCzAJBgNVBAYTAklUMQ4wDAYDVQQIDAVMQVpJTzERMA8GA1UECgwI
U2FwaWVuemExFDASBgNVBAMMC2hhc3NhYW4uY29tMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAyD2XBo8hjZxYHIOzoWlO5Z+rODfExQ9+TaeNYSwLrzMJ
OkKGFUZ59YL/yaYZKaCvKLJOEUNXOdTnuEowWK1LjaGjidmrWNyqaYmt5NBe7Anf
tWOfL/26TrQOfnW5C/yFGpjOFjzYWLWiry+H+YHzvylpxSPGHxByl13wxIOgZNKd
OQSDj9td0tCfF8H/rKkig065SqiMBo4q5ZFhOE8p3X5qX5+nVHW4r3xPBty2vhfG
atk6ASdpGDbRdUoZEjomyhp6zdbVw8mnzRSUmkH6O3O1qAl1FR0x17fBoW22xdJs
mmSOFyC+4WGYkjiMuGTrs3jMAKf9mBMR471fZvMsYwIDAQABo3sweTAJBgNVHRME
AjAAMCwGCWCGSAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0
ZTAdBgNVHQ4EFgQUkl6NHYjWCLAn7Np4o14KVjNHRcswHwYDVR0jBBgwFoAUUmxf
7ERrefjDkuHRU5KBjBgkpOkwDQYJKoZIhvcNAQELBQADggEBALsVQtQFWwGBtt6Q
XWsITM/hT4JH4wyw9vZa/gZuN7/Or6sMD69Y1fXR9H9+cnaV4DIEwQkD6fbXZb7v
aW2rfv+U2529tsr0XyDGM+/XsX2Oxk04VJ5HmWOOH4uA9nsPdRonrN7tJFSmZcbB
NFja1M9nOgDaipxDMdq/agL21gkVRdmfOtLynk2xL+ji8zSQuNB6d5oTga4gMIuP
V13wlNM5hKPBhX/MbO34tBS4wXFul7jITwb09lueuxtaFH2aJWJRnaOlLZlUXjaN
uzPenLyR87QkBkflqH2d1iMUjmr2jT/KIK8Hw0KLCIfNar4SNXEHI9NEW/a4zft8
VFtYjMI=
-----END CERTIFICATE-----
Data Base Updated
```

To check, we can go inside the demoCA directory and inside which there is a newcerts folder which contains signed certificates. Checking which we get two certificates now, one which is the CA itself and other one which we just signed:

```
33DE36B5C53F3A8E7B81BAA930AF4813802C1C10.pem
33DE36B5C53F3A8E7B81BAA930AF4813802C1C11.pem
```

To verify our certificate, we go to the root folder in our CA system and using the following command we provide the new signed certificate path to the file:

openssl verify -CAfile demoCA/cacert.pem demoCA/newcerts/33DE36B5C53F3A8E7B81BAA930AF48

```
demoCA/newcerts/33DE36B5C53F3A8E7B81BAA930AF4813802C1C11.pem: OK
```

So we know now our certificate is signed.

10 Revoking certificate at CA

Upon request from the VM in the same way using netcat, In order to revoke signed certificate at CA machine, we use the following command at CA:

openssl ca -revoke demoCA/newcerts/33DE36B5C53F3A8E7B81BAA930AF4813802C1C11.pem

This will prompt us asking the passkey for private key, which on providing it will revoke the signed certificate:

```
Revoking Certificate 33DE36B5C53F3A8E7B81BAA930AF4813802C1C11.
Data Base Updated
```

But as we know this is not enough, we need to generate crl of revoked certificate in crl directory, to get that, we use:

openssl ca -gencrl -out demoCA/crl/crl.pem

Now if we check the VMs previously signed certificate in the CA machine, this time we also check using the CRL check whether its valid or not, we use the following command:

openssl verify -CAfile demoCA/cacert.pem -CRLfile demoCA/crl/crl.pem -crl_check demoCA/newcerts/33DE36B5C53F3A8E7B81BAA930AF4813802C1C11.pem

We get this error which says certificate revoked:

```
C = IT, ST = LAZIO, O = Sapienza, CN = hassaan.com
error 23 at 0 depth lookup: certificate revoked
error demoCA/newcerts/33DE36B5C53F3A8E7B81BAA930AF4813802C1C11.pem: verification fail
```

11 Conclusion

In this way we can perform a certificate signing using two saperate machines. In summary, we make one machine a CA and generate files and configure it as required. The other machine generates a request and then sends its request to the CA to sign it. The CA checks if it's valid and signs it. Upon request fron the VM to revoke the signed certificate, the CA revokes it and updates in the database.