Al-Najah National University
Department of Engineering and Information technology
Computer Network and Information Security

Secret Key Encryption

Eyad Shamlouli
Mohammed Adnan
Instructor: Dr. Ahmed Awad

28/2/2021

1 Abstract

This experiment examines basic openssl tool command to perform a public-key encryption and how to generate key pair , digital signature, and Public Key Infrastructure (PKI) certificates.

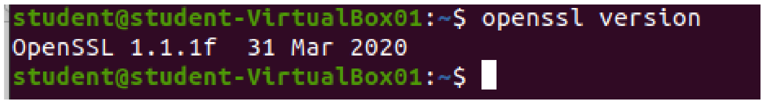
2 Introduction

Public key cryptography refer to a cryptographic system that use a different key for encrypt and decrypt the data. The generation of such key pairs depends on cryptographic algorithms which are based on mathematical problems termed one-way functions. Effective security requires keeping the private key private; the public key can be openly distributed without compromising security. Public key algorithms are fundamental security primitives in modern cryptosystems, including applications and protocols which offer assurance of the confidentiality, authenticity and non-repudiability of electronic communications and data storage. In this experiment we generate certification authority **(CA)** which is an entity that issues digital certificates. A digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (relying parties) to rely upon signatures or on assertions made about the private key that corresponds to the certified public key. A CA acts as a trusted third party—trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. The format of these certificates is specified by the X.509 standard. After we generate our CA we made a Certificate Signing Request **(CSR)** . The CSR contains information identifying the applicant (such as a distinguished name in the case of an X.509 certificate) which must be signed using the applicant's private key. The CSR also contains the public key chosen by the applicant. The CSR may be accompanied by other credentials or proofs of identity required by the certificate authority, and the certificate authority may contact the applicant for further information. Then we made a Public Key Infrastructure **(PKI)** for our Domain, a PKI is an arrangement that binds public keys with respective identities of entities (like people and organizations). The binding is established through a process of registration and issuance of certificates at and by a certificate authority (CA). Depending on the assurance level of the binding, this may be carried out by an automated process or under human supervision. Digital Signatures is also used in this experiment, a digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents, that used private key for sign and public key for verification signature, in the last part of this experiment we compare the time spent between **RSA** public key encryption and symmetric (secret) key encryption .

3 Procedure

3.1 Certificate Authority (CA)

1. We made sure that **OpenSSL** tool has been installed on our machine, as shown in Figure 1.



```
student@student-VirtualBox01:~$ openssl version
OpenSSL 1.1.1f  31 Mar 2020
student@student-VirtualBox01:~$ 
```

Figure 1: Openssl verification.

2. A Certificate Authority (CA) is a trusted entity that issues digital certificates to users. The users are then certified to their ownership of a public key by this certificate. Digital certificates are electronic credentials that bind the identity of the certificate owner to a pair of electronic encryption keys, (one public and one private), that can be used to encrypt and sign information digitally. The main purpose of the digital certificate is to ensure that the public key contained in the certificate belongs to the entity to which the certificate was issued.

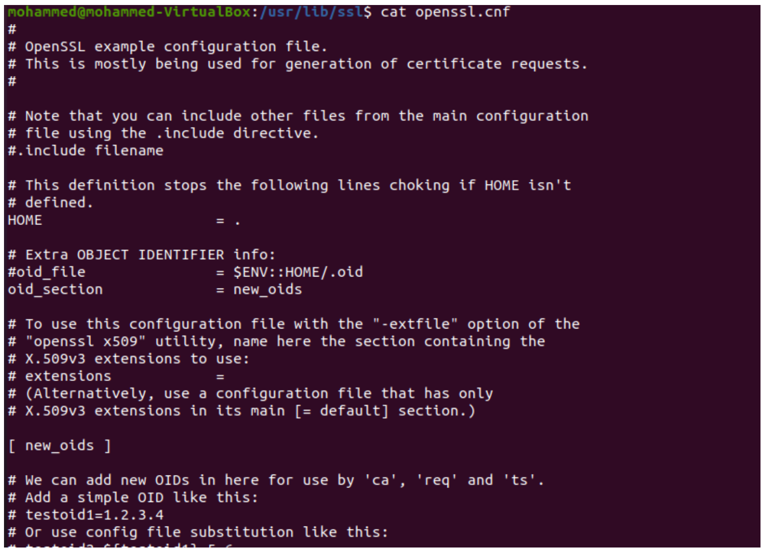
3. **Question:** Provide an example of three commercial CAs

   Let's Encrypt, **Comodo**, and **Symantec** are a good example of commercial CAs.

4. **Question:** What is a root CA? How does a root CA gain its own certificates ?

   A root certificate is a public key certificate that identifies a root certificate authority (CA). Root certificates are self-signed.

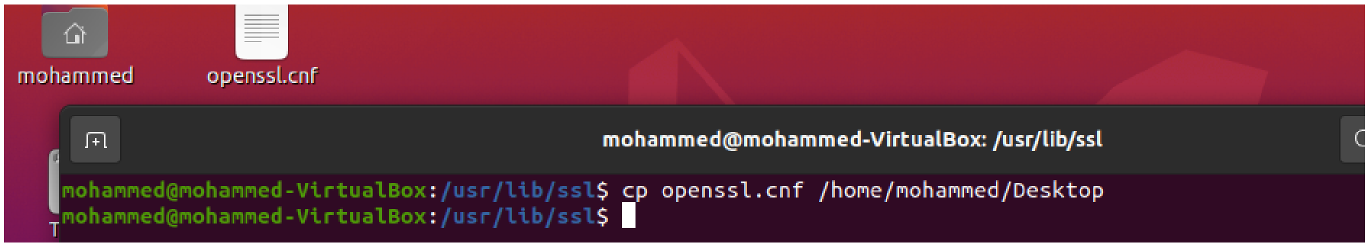
5. We checked the content of the configuration file of openssl under the directory **/usr/lib/ssl/**. The name of the file is **openssl.cnf**, as shown in Figure 2.



```
mohammed@mohammed-VirtualBox:/usr/lib/ssl$ cat openssl.cnf
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# Note that you can include other files from the main configuration
# file using the .include directive.
#.include filename

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .

# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
```

Figure 2: Contents of openssl.cnf.

6. We created a copy of **openssl.cnf** file under our desktop directory, as shown in Figure 3.



```
mohammed@mohammed-VirtualBox: /usr/lib/ssl
mohammed@mohammed-VirtualBox:/usr/lib/ssl$ cp openssl.cnf /home/mohammed/Desktop
mohammed@mohammed-VirtualBox:/usr/lib/ssl$ 
```

Figure 3: openssl.cnf copy creation.

7. We created the sub-directory demoCA in our current directory, then created a file named **index.txt** under demoCA, and created a file named **serial** under demoCA. We also inserted a single number in string format in the file, as shown in Figure 4.

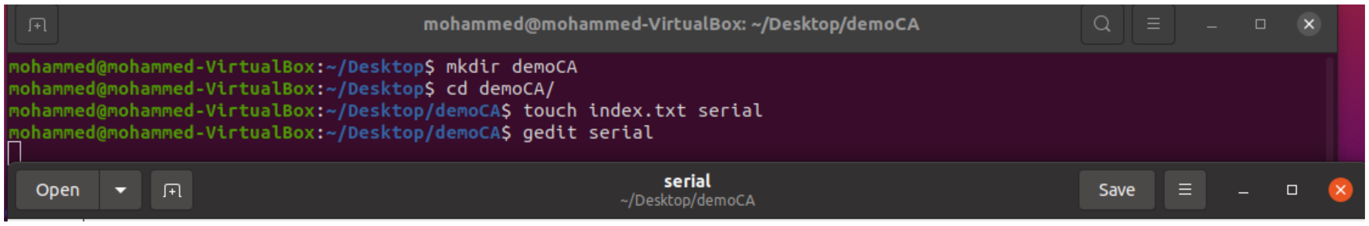

```
mohammed@mohammed-VirtualBox: ~/Desktop/demoCA
mohammed@mohammed-VirtualBox:~/Desktop$ mkdir demoCA
mohammed@mohammed-VirtualBox:~/Desktop$ cd demoCA/
mohammed@mohammed-VirtualBox:~/Desktop/demoCA$ touch index.txt serial
mohammed@mohammed-VirtualBox:~/Desktop/demoCA$ gedit serial
```

Figure 4: Sub-directory creation.

8. We created the following sub-directories under demoCA: **certs**, **crl**, **newcerts**, as shown in Figure 5.

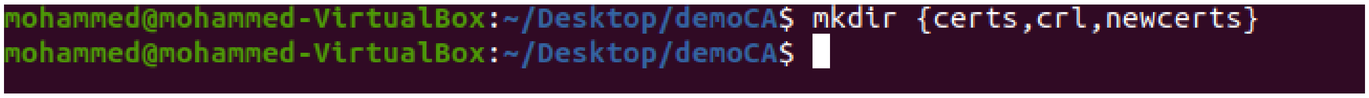

```
mohammed@mohammed-VirtualBox:~/Desktop/demoCA$ mkdir {certs,crl,newcerts}
mohammed@mohammed-VirtualBox:~/Desktop/demoCA$ 
```

Figure 5: Files creation.

9. We opened the file openssl.cnf that we have copied and checked the values that correspond to the names of the files and directories we created, as shown in Figure 6.

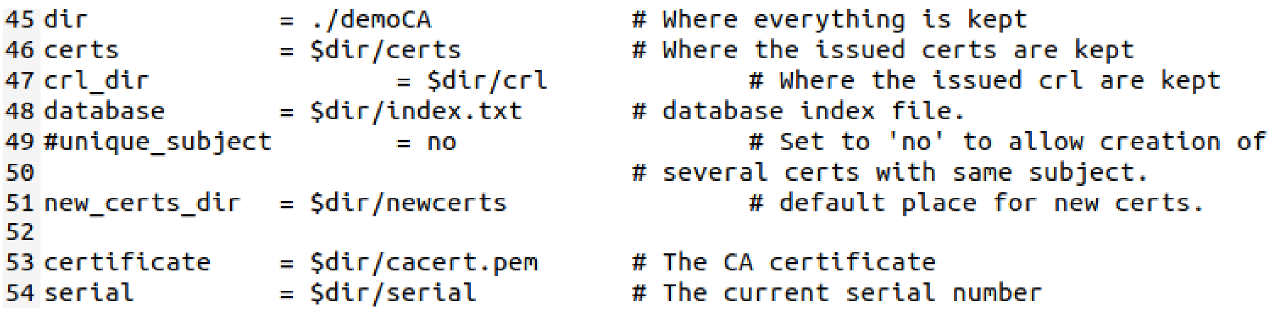
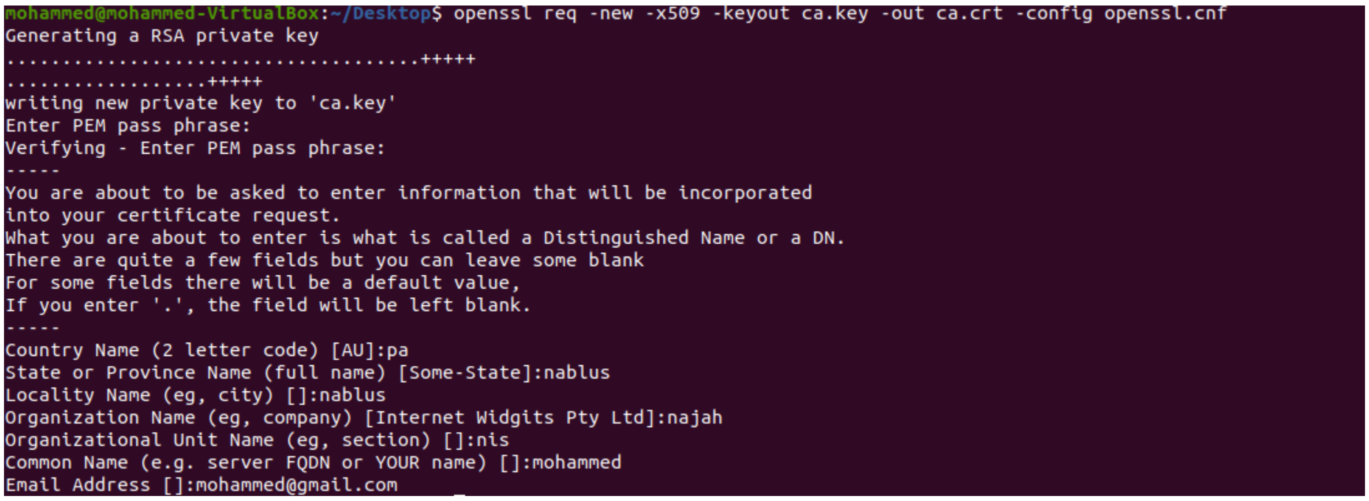

```
dir             = ./demoCA              # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir                 = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject         = no                    # Set to 'no' to allow creation of
                                        # several certs with same subject.
new_certs_dir   = $dir/newcerts                 # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
```

Figure 6: Checking values.

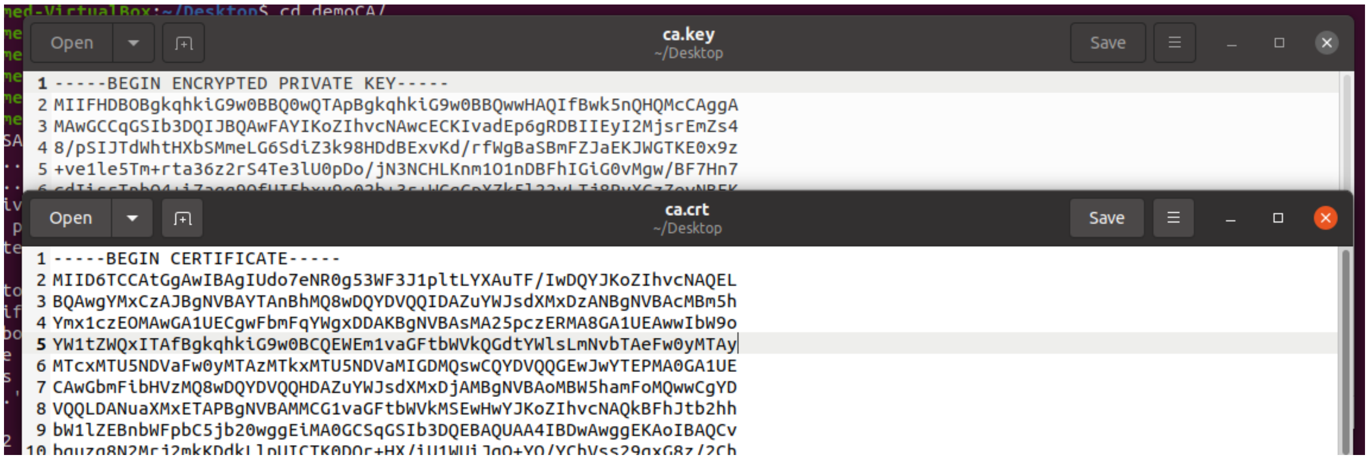
10. We then generated a self-signed certificate for our CA using the following command:
    openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf, as shown in Figure 7.



```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
Generating a RSA private key
.....................................+++++
..................+++++
writing new private key to 'ca.key'
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]:pa
State or Province Name (full name) [Some-State]:nablus
Locality Name (eg, city) []:nablus
Organization Name (eg, company) [Internet Widgits Pty Ltd]:najah
Organizational Unit Name (eg, section) []:nis
Common Name (e.g. server FQDN or YOUR name) []:mohammed
Email Address []:mohammed@gmail.com
```

Figure 7: Certificate generation.

11. We have two files generated **ca.key** contains the private key of our CA, while **ca.crt** contains the public key certificate, as shown in Figure 8.



```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFHDBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQIfBwk5nQHQMcCAggA
MAwGCCqGSIb3DQIJBQAwFAYIKoZIhvcNAwcECKIvadEp6gRDBIIEyI2MjsrEmZs4
8/pSIJTdWhtHXbSMmeLG6SdiZ3k98HDdBExvKd/rfWgBaSBmFZJaEKJWGTKE0x9z
+ve1le5Tm+rta36z2rS4Te3lU0pDo/jN3NCHLKnm1O1nDBFhIGiG0vMgw/BF7Hn7
```

```
-----BEGIN CERTIFICATE-----
MIID6TCCAtGgAwIBAgIUdo7eNR0g53WF3J1pltLYXAuTF/IwDQYJKoZIhvcNAQEL
BQAwgYMxCzAJBgNVBAYTAnBhMQ8wDQYDVQQIDAZuYWJsdXMxDzANBgNVBAcMBm5h
Ymx1czEOMAwGA1UECgwFbmFqYWgxDDAKBgNVBAsMA25pczERMA8GA1UEAwwIbW9o
YW1tZWQxITAfBgkqhkiG9w0BCQEWEm1vaGFtbWVkQGdtYWlsLmNvbTAeFw0yMTAy
MTcxMTU5NDVaFw0yMTAzMTkxMTU5NDVaMIGDMQswCQYDVQQGEwJwYTEPMA0GA1UE
CAwGbmFibHVzMQ8wDQYDVQQHDAZuYWJsdXMxDjAMBgNVBAoMBW5hamFoMQwwCgYD
VQQLDANuaXMxETAPBgNVBAMMCG1vaGFtbWVkMSEwHwYJKoZIhvcNAQkBFhJtb2hh
bW1lZEBnbWFpbC5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCv
```

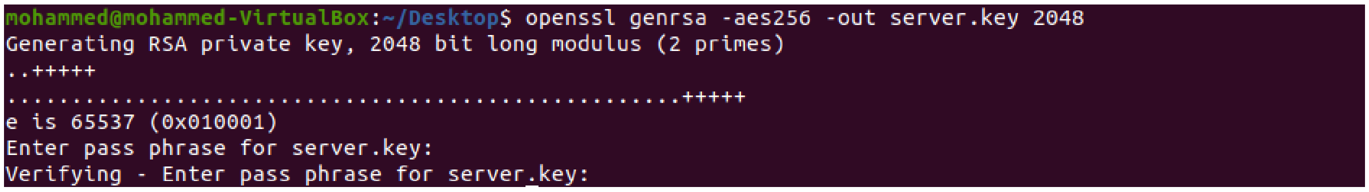
Figure 8: Generated files content.

3.2 Giving Certificates for Clients

We pretended to be a CA root. We assumed that a company called MyCompany.com would like to get a digital certificate from us as CA.

3.2.1 Key-Pair Generation By the Client

1. The company has first to generate its public-private key pair. We let the encryption algorithm be RSA. To do so, we issued the following command: **openssl genrsa -aes128 -out server.key 1024**, as shown in Figure 9.



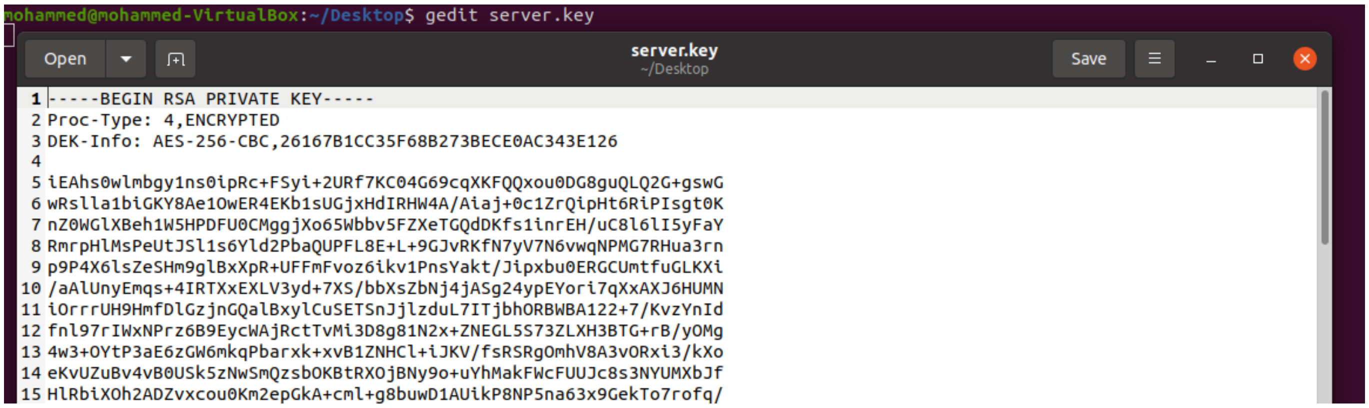
```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl genrsa -aes256 -out server.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
..+++++
...................................................+++++
e is 65537 (0x010001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Figure 9: Key pair generation.

2. **Question:** What is the purpose of AES in the previous command?
   To perform an aes encryption on the generated rsa private key.

3. **Question:** What does 1024 represent in the previous command?
   It represents the rsa key size.

4. **Question:** Can you open the file server.key using a text editor ?
   Yes, we were able to open the file as shown in Figure 10.

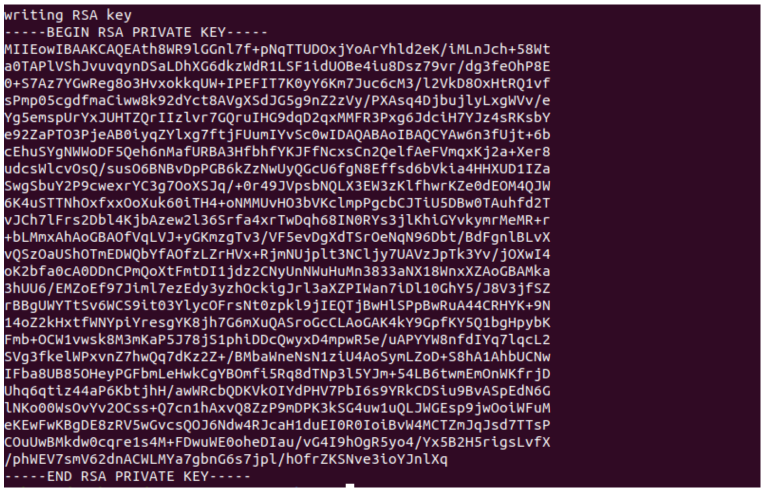


```
mohammed@mohammed-VirtualBox:~/Desktop$ gedit server.key
```

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,26167B1CC35F68B273BECE0AC343E126

iEAhs0wlmbgy1ns0ipRc+FSyi+2URf7KC04G69cqXKFQQxou0DG8guQLQ2G+gswG
wRslla1biGKY8Ae1OwER4EKb1sUGjxHdIRHW4A/Aiaj+0c1ZrQipHt6RiPIsgt0K
nZ0WGlXBeh1W5HPDFU0CMggjXo65Wbbv5FZXeTGQdDKfs1inrEH/uC8l6lI5yFaY
RmrpHlMsPeUtJSl1s6Yld2PbaQUPFL8E+L+9GJvRKfN7yV7N6vwqNPMG7RHua3rn
p9P4X6lsZeSHm9glBxXpR+UFFmFvoz6ikv1PnsYakt/Jipxbu0ERGCUmtfuGLKXi
/aAlUnyEmqs+4IRTXxEXLV3yd+7XS/bbXsZbNj4jASg24ypEYori7qXxAXJ6HUMN
iOrrrUH9HmfDlGzjnGQalBxylCuSETSnJjlzduL7ITjbhORBWBA122+7/KvzYnId
fnl97rIWxNPrz6B9EycWAjRctTvMi3D8g81N2x+ZNEGL5S73ZLXH3BTG+rB/yOMg
4w3+OYtP3aE6zGW6mkqPbarxk+xvB1ZNHCl+iJKV/fsRSRgOmhV8A3vORxi3/kXo
eKvUZuBv4vB0USk5zNwSmQzsbOKBtRXOjBNy9o+uYhMakFWcFUUJc8s3NYUMXbJf
HlRbiXOh2ADZvxcou0Km2epGkA+cml+g8buwD1AUikP8NP5na63x9GekTo7rofq/
```

Figure 10: Opening server.key.

5. We ran the command **openssl rsa -in server.key -text**, and the content of the file **server.key** was an rsa private key, as shown in Figure 11.



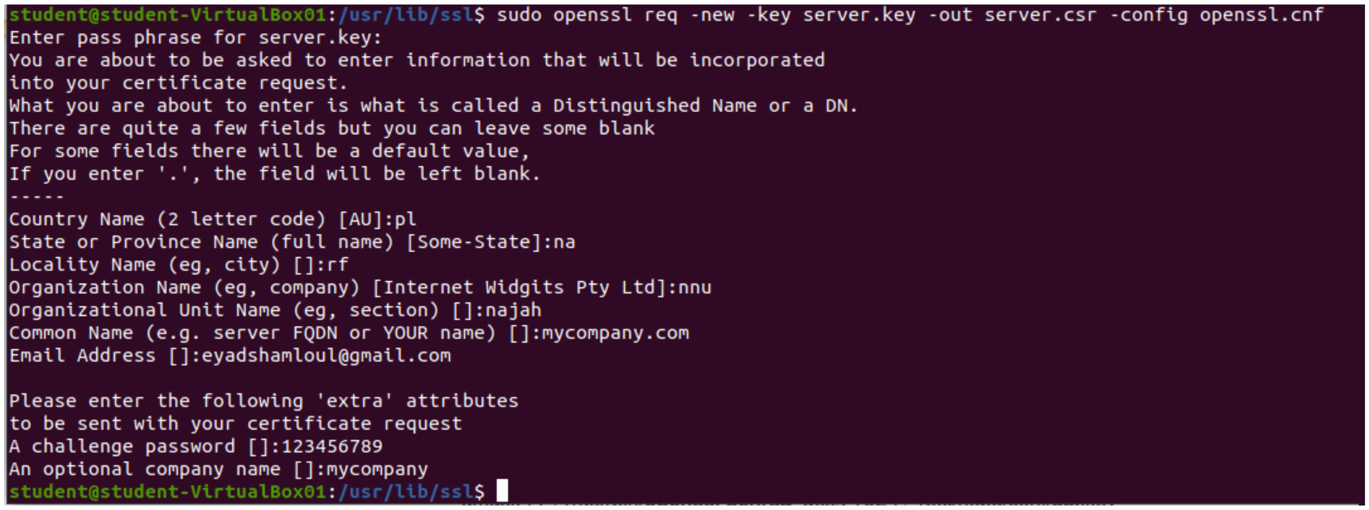
```
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAth8WR9lGGnl7f+pNqTTUDOxjYoArYhld2eK/iMLnJch+58Wt
a0TAPlVShJvuvqynDSaLDhXG6dkzWdR1LSF1idUOBe4iu8Dsz79vr/dg3feOhP8E
0+S7Az7YGwReg8o3HvxokkqUW+IPEFIT7K0yY6Km7Juc6cM3/l2VkD8OxHtRQ1vf
sPmp05cgdfmaCiww8k92dYct8AVgXSdJG5g9nZ2zVy/PXAsq4DjbujlyLxgWVv/e
Yg5emspUrYxJUHTZQrIIzlvr7GQruIHG9dqD2qxMMFR3Pxg6JdciH7YJz4sRKsbY
e92ZaPTO3PjeAB0iyqZYlxg7ftjFUumIYvSc0wIDAQABAoIBAQCYAw6n3fUjt+6b
cEhuSYgNWWoDF5Qeh6nMafURBA3HfbhfYKJFfNcxsCn2QelfAeFVmqxKj2a+Xer8
udcsWlcvOsQ/susO6BNBvDpPGB6kZzNwUyQGcU6fgN8Effsd6bVkia4HHXUD1IZa
SwgSbuY2P9cwexrYC3g7OoXSJq/+0r49JVpsbNQLX3EW3zKlfhwrKZe0dEOM4QJW
6K4uSTTNhOxfxxOoXuk60iTH4+oNMMUvHO3bVKclmpPgcbCJTiU5DBw0TAuhfd2T
vJCh7lFrs2Dbl4KjbAzew2l36Srfa4xrTwDqh68IN0RYs3jlKhiGYvkymrMeMR+r
+bLMmxAhAoGBAOfVqLVJ+yGKmzgTv3/VF5evDgXdTSrOeNqN96Dbt/BdFgnlBLvX
vQSzOaUShOTmEDWQbYfAOfzLZrHVx+RjmNUjplt3NCljy7UAVzJpTk3Yv/jOXwI4
oK2bfa0cA0DDnCPmQoXtFmtDI1jdz2CNyUnNWuHuMn3833aNX18WnxXZAoGBAMka
3hUU6/EMZoEf97Jiml7ezEdy3yzhOckigJrl3aXZPIWan7iDl10GhY5/J8V3jfSZ
rBBgUWYTtSv6WCS9it03YlycOFrsNt0zpkl9jIEQTjBwHlSPpBwRuA44CRHYK+9N
14oZ2kHxtfWNYpiYresgYK8jh7G6mXuQASroGcCLAoGAK4kY9GpfKY5Q1bgHpybK
Fmb+OCW1vwsk8M3mKaP5J78jS1phiDDcQwyxD4mpwR5e/uAPYYW8nfdIYq7lqcL2
SVg3fkelWPxvnZ7hwQq7dKz2Z+/BMbaWneNsN1ziU4AoSymLZoD+S8hA1AhbUCNw
IFba8UB85OHeyPGFbmLeHwkCgYBOmfi5Rq8dTNp3l5YJm+54LB6twmEmOnWKfrjD
Uhq6qtiz44aP6KbtjhH/awWRcbQDKVkOIYdPHV7PbI6s9YRkCDSiu9BvASpEdN6G
lNKo00WsOvYv2OCss+Q7cn1hAxvQ8ZzP9mDPK3kSG4uw1uQLJWGEsp9jwOoiWFuM
eKEwFwKBgDE8zRV5wGvcsQOJ6Ndw4RJcaH1duEI0R0IoiBvW4MCTZmJqJsd7TTsP
COuUwBMkdw0cqre1s4M+FDwuWE0oheDIau/vG4I9hOgR5yo4/Yx5B2H5rigsLvfX
/phWEV7smV62dnACWLMYa7gbnG6s7jpl/hOfrZKSNve3ioYJnlXq
-----END RSA PRIVATE KEY-----
```

Figure 11: server.key content.

3.2.2 Generation of Certificate Signing Request (CSR)

When the client generates the key pair, it has to generate a CSR which includes its public key. Once the CA receives the CSR, it confirms the identity information of the CSR.

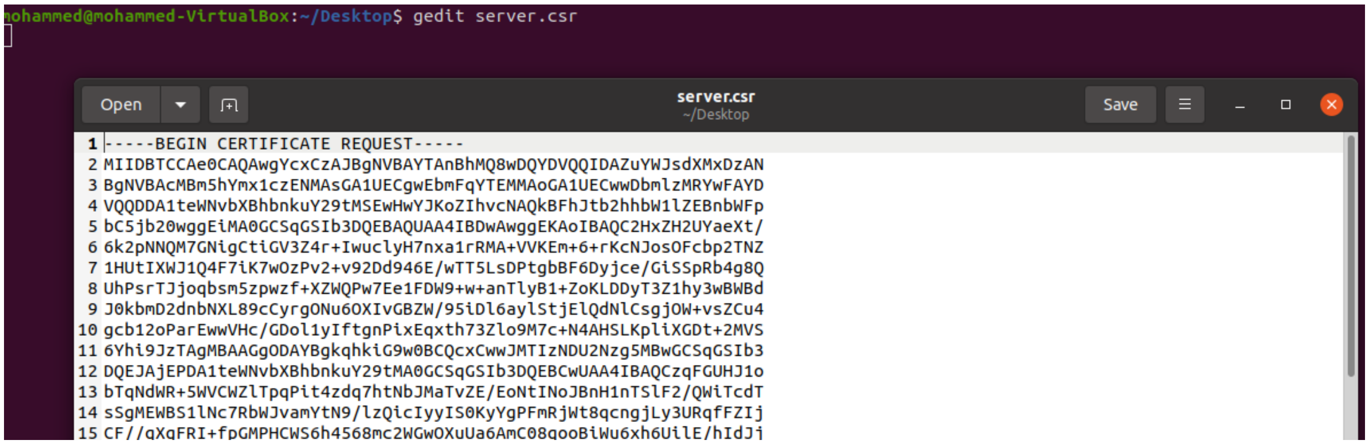
1. We issued the following command **openssl req -new -key server.key -out server.csr-config openssl.cnf**, and this command generates a new certificate request, it will prompt the user for the relevant field values, as shown in Figure 12.



```
student@student-VirtualBox01:/usr/lib/ssl$ sudo openssl req -new -key server.key -out server.csr -config openssl.cnf
Enter pass phrase for server.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]:pl
State or Province Name (full name) [Some-State]:na
Locality Name (eg, city) []:rf
Organization Name (eg, company) [Internet Widgits Pty Ltd]:nnu
Organizational Unit Name (eg, section) []:najah
Common Name (e.g. server FQDN or YOUR name) []:mycompany.com
Email Address []:eyadshamloul@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456789
An optional company name []:mycompany
student@student-VirtualBox01:/usr/lib/ssl$
```

Figure 12: New certificate command.

2. We checked the content of the file **server.csr**, as shown in Figure13.



```
mohammed@mohammed-VirtualBox:~/Desktop$ gedit server.csr
```

```
-----BEGIN CERTIFICATE REQUEST-----
MIIDBTCCAe0CAQAwgYcxCzAJBgNVBAYTAnBhMQ8wDQYDVQQIDAZuYWJsdXMxDzAN
BgNVBAcMBm5hYmx1czENMAsGA1UECgwEbmFqYTEMMAoGA1UECwwDbmlzMRYwFAYD
VQQDDA1teWNvbXBhbnkuY29tMSEwHwYJKoZIhvcNAQkBFhJtb2hhbW1lZEBnbWFp
bC5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC2HxZH2UYaeXt/
6k2pNNQM7GNigCtiGV3Z4r+IwuclyH7nxa1rRMA+VVKEm+6+rKcNJosOFcbp2TNZ
1HUtIXWJ1Q4F7iK7wOzPv2+v92Dd946E/wTT5LsDPtgbBF6Dyjce/GiSSpRb4g8Q
UhPsrTJjoqbsm5zpwzf+XZWQPw7Ee1FDW9+w+anTlyB1+ZoKLDDyT3Z1hy3wBWBd
J0kbmD2dnbNXL89cCyrgONu6OXIvGBZW/95iDl6aylStjElQdNlCsgjOW+vsZCu4
gcb12oParEwwVHc/GDol1yIftgnPixEqxth73Zlo9M7c+N4AHSLKpliXGDt+2MVS
6Yhi9JzTAgMBAAGgODAYBgkqhkiG9w0BCQcxCwwJMTIzNDU2Nzg5MBwGCSqGSIb3
DQEJAjEPDA1teWNvbXBhbnkuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQCzqFGUHJ1o
bTqNdWR+5WVCWZlTpqPit4zdq7htNbJMaTvZE/EoNtINoJBnH1nTSlF2/QWiTcdT
sSgMEWBS1lNc7RbWJvamYtN9/lzQicIyyIS0KyYgPFmRjWt8qcngjLy3URqfFZIj
CF//qXqFRI+fpGMPHCWS6h4568mc2WGwOXuUa6AmC08qooBiWu6xh6UilE/hIdJi
```

Figure 13: Content of server.csr.

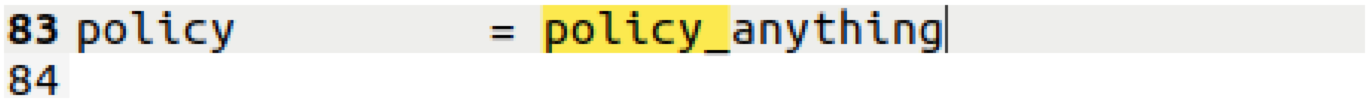
3.2.3 Generating Certificates

The CSR needs to be converted into a certificate. This means, the CSR needs to be signed by the CA.

1. **Question:** What are the needed files in your current directory to sign the CSR by the CA?

   The needed files are **Server.csr**, **Server.crt**, **ca.crt**, **ca.key**, and **openssl.cnf**.

2. We changed the following line in **openssl.cnf** file **policy=policy-match to policy-anything**, as shown in Figure14.

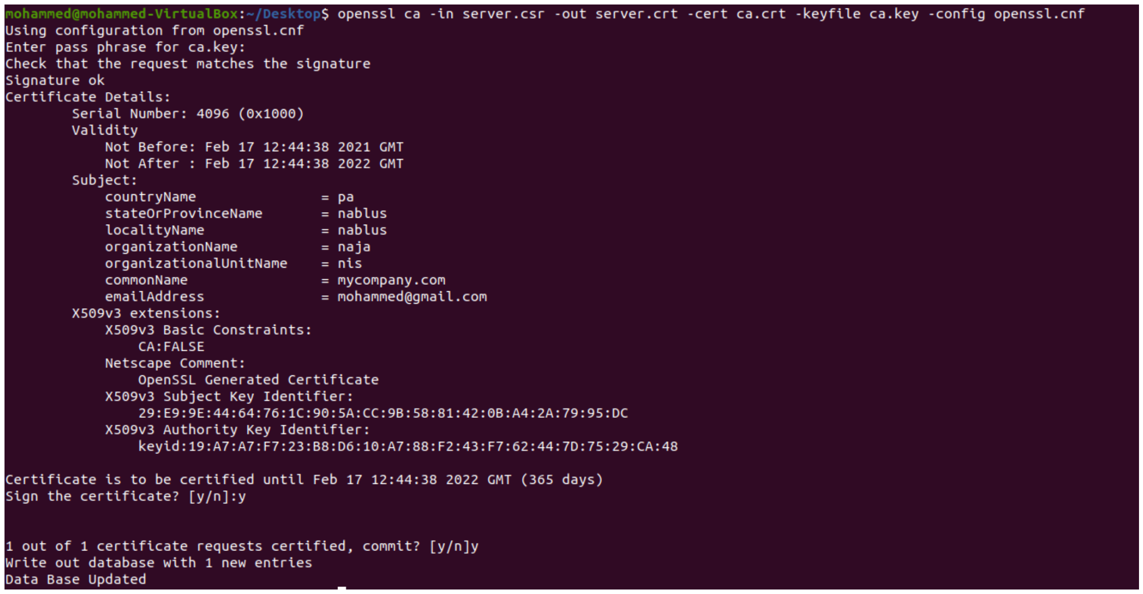


```
policy          = policy_anything
```

Figure 14: Changing policy.

3. **Question:**What is the purpose of this modification?
   Here we define a ”policy_anything” policy where we accept anything, and only require a CN.

4. We issued the following command :
   openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf, and we acquired the generated certificates of the client, as shown in Figure 15.



```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Feb 17 12:44:38 2021 GMT
            Not After : Feb 17 12:44:38 2022 GMT
        Subject:
            countryName               = pa
            stateOrProvinceName       = nablus
            localityName              = nablus
            organizationName          = naja
            organizationalUnitName    = nis
            commonName                = mycompany.com
            emailAddress              = mohammed@gmail.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                29:E9:9E:44:64:76:1C:90:5A:CC:9B:58:81:42:0B:A4:2A:79:95:DC
            X509v3 Authority Key Identifier:
                keyid:19:A7:A7:F7:23:B8:D6:10:A7:88:F2:43:F7:62:44:7D:75:29:CA:48

Certificate is to be certified until Feb 17 12:44:38 2022 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

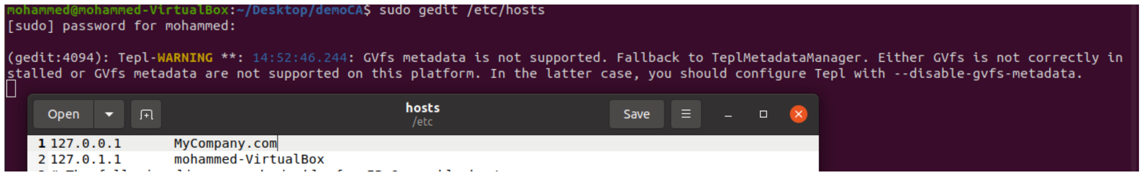
Figure 15: Generated certificate.

Question: Explain the content of index.txt and serial files ?
Index.txt will have our information for the certificate, the **serial** file which contains the string "1000" will be incremented by 1, which means we have requested one certificate.

3.3 Public Key Infrastructure (PKI) for Domains

As mentioned before, we let **MyCompany.com** be the domain of the client whose certificate will be used by web browsers to guarantee secure browsing.

1. We manually created an entry for the domain **MyCompany.com** in our Linux machine. We let the IP address mapped to this domain be the localhost (127.0.0.1), as shown in Figure 16. **Question:** How will you do that?
   We did that by inserting it's ip address and domain in **etc/hosts** file, as shown in Figure 16.



```
mohammed@mohammed-VirtualBox:~/Desktop/demoCA$ sudo gedit /etc/hosts
[sudo] password for mohammed:

(gedit:4094): Tepl-WARNING **: 14:52:46.244: GVfs metadata is not supported. Fallback to TeplMetadataManager. Either GVfs is not correctly in
stalled or GVfs metadata are not supported on this platform. In the latter case, you should configure Tepl with --disable-gvfs-metadata.
```

```
1 127.0.0.1       MyCompany.com
2 127.0.1.1       mohammed-VirtualBox
```

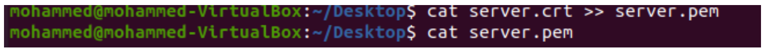
Figure 16: Entry creation.

2. We combined the secret key and the certificate into one file named **server.pem**. To do so we issued the following commands:
   cp server.key server.pem
   cat server.crt >> server.pem

   The first command makes a copy the secret key, and the second one combines it with the certificate, as shown in Figure 17.

   Question: What is the purpose of combining the certificate and the key into one file?

   Sometimes certificate files and private keys are supplied as distinct files but linux and Windows requires certificates with private keys to be in a single PEM , BFX file.

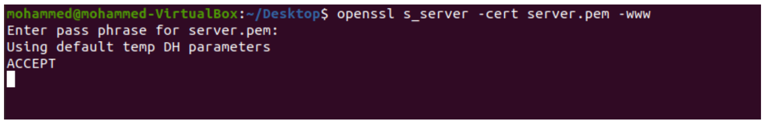


```
mohammed@mohammed-VirtualBox:~/Desktop$ cat server.crt >> server.pem
mohammed@mohammed-VirtualBox:~/Desktop$ cat server.pem
```

Figure 17: Combining key with certificate.

3. We started a web browser using the following command: **openssl s server -cert server.pem -www**, as shown in Figure 18.

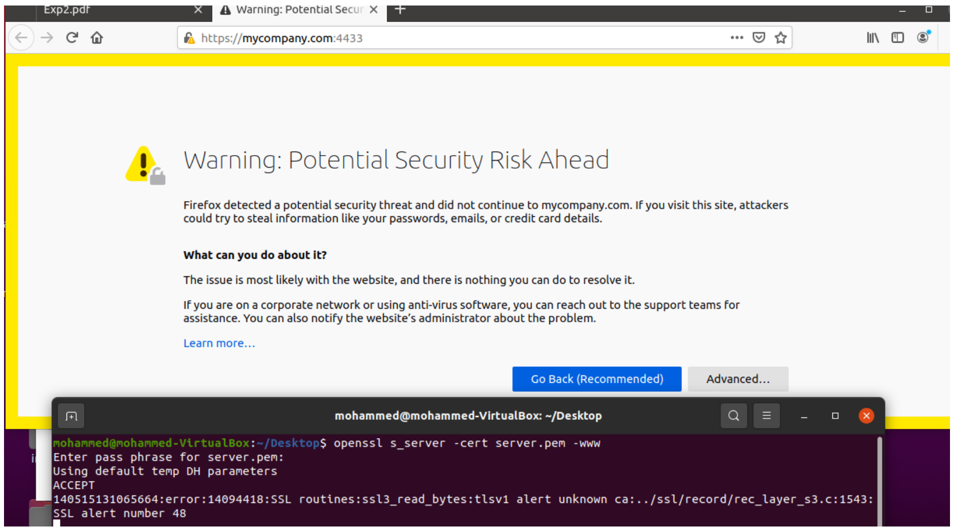
   Question: You will prompted to enter a passphrase several times. Explain why?
   We have to enter the passphrase just one time to open the connection.



```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

Figure 18: Starting browser.

4. The default used port number is 4433. We launched the Firefox and used the following URL:
   https://mycompany.com:4433/, as shown in Figure 19.



Exp2.pdf
Warning: Potential Secur
https://mycompany.com:4433
Go Back (Recommended)
Advanced...
mohammed@mohammed-VirtualBox: ~/Desktop

Warning: Potential Security Risk Ahead

Firefox detected a potential security threat and did not continue to mycompany.com. If you visit this site, attackers could try to steal information like your passwords, emails, or credit card details.

What can you do about it?

The issue is most likely with the website, and there is nothing you can do to resolve it.

If you are on a corporate network or using anti-virus software, you can reach out to the support teams for assistance. You can also notify the website's administrator about the problem.

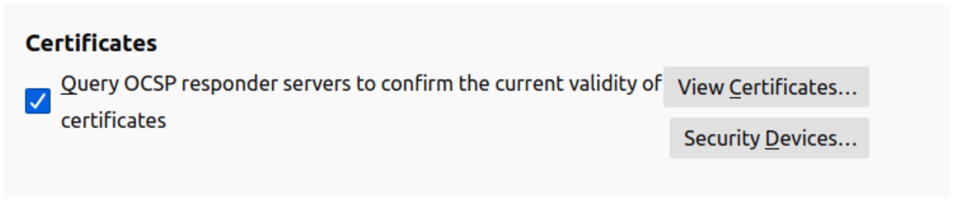
Learn more...

```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
140515131065664:error:14094418:SSL routines:ssl3_read_bytes:tlsv1 alert unknown ca:../ssl/record/rec_layer_s3.c:1543:
SSL alert number 48
```

Figure 19: Firefox launch.

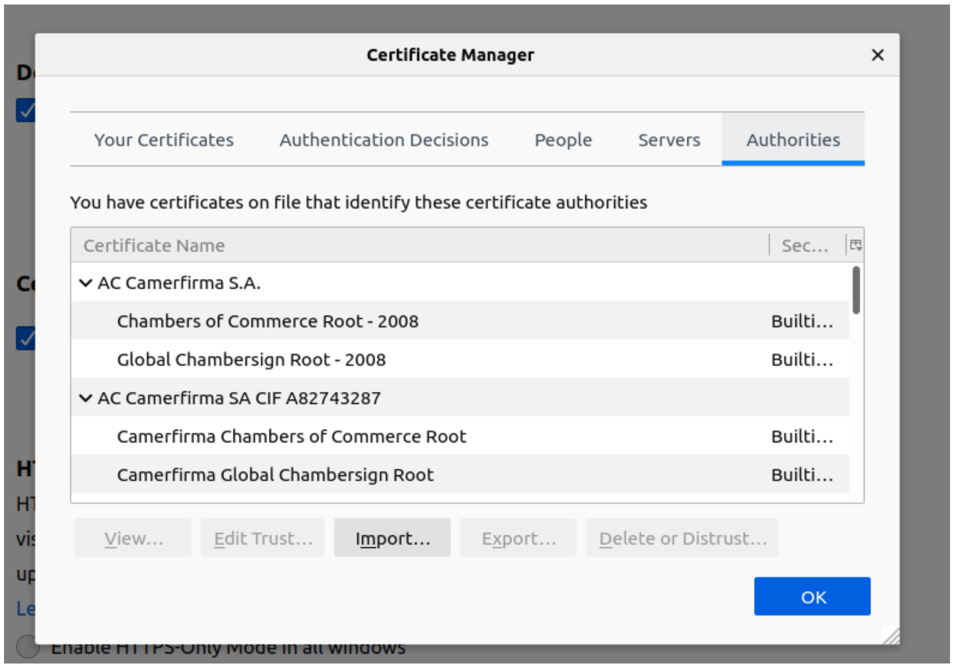
Question: What did you get? Explain why?
It displayed an ssl alert, because it gave us the code 48 for "unknown_ca", which means that verification failed.

5. Usually, a CA requests browsing companies (such as Mozilla for Firefox) to include its own certificates. However, as this is not possible here, we manually loaded our certificate (ca.cert) file into Firefox, as shown in Figures 20, 21, and 22.


Certificates

✓ Query OCSP responder servers to confirm the current validity of certificates

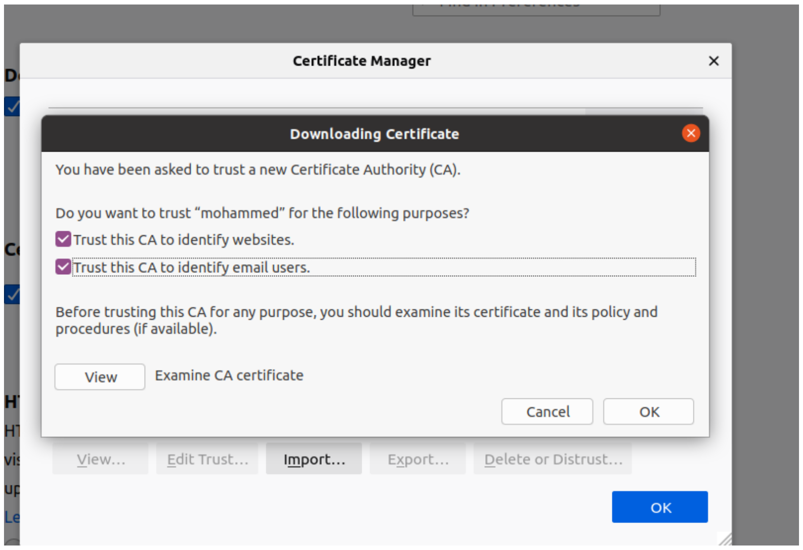
Figure 20: Confirm the validity.


Certificate Manager

You have certificates on file that identify these certificate authorities

Certificate Name	Sec…
AC Camerfirma S.A.	
Chambers of Commerce Root - 2008	Builti…
Global Chambersign Root - 2008	Builti…
AC Camerfirma SA CIF A82743287	
Camerfirma Chambers of Commerce Root	Builti…
Camerfirma Global Chambersign Root	Builti…

Figure 21: Certificate manager.



You have been asked to trust a new Certificate Authority (CA).

Do you want to trust "mohammed" for the following purposes?

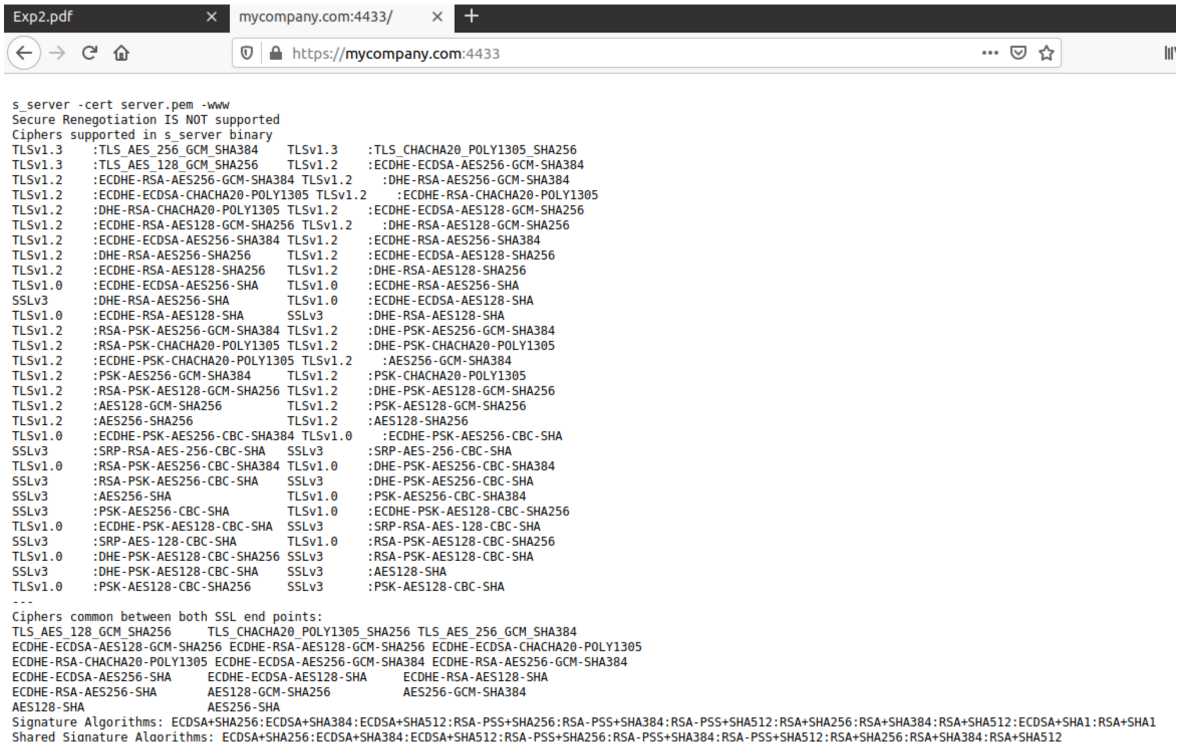
☑ Trust this CA to identify websites.

☑ Trust this CA to identify email users.

Before trusting this CA for any purpose, you should examine its certificate and its policy and procedures (if available).

Figure 22: Downloading certificate.

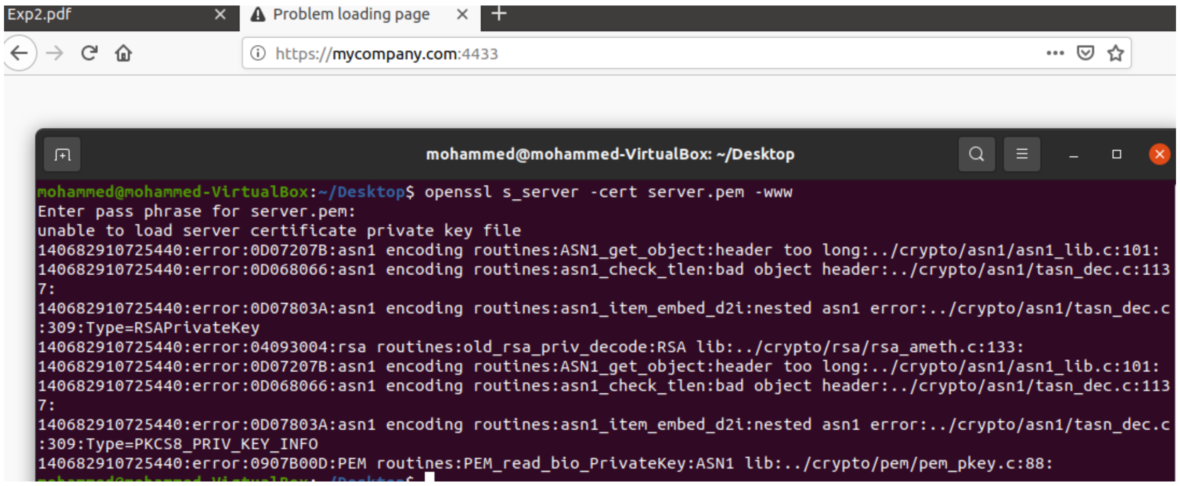
6. We used the URL **https://mycompany.com:4433/** again in our Firefox, and this time it worked because we loaded our own certificate into firefox, as shown in Figure 23.



```
s_server -cert server.pem -www
Secure Renegotiation IS NOT supported
Ciphers supported in s_server binary
TLSv1.3    :TLS_AES_256_GCM_SHA384    TLSv1.3    :TLS_CHACHA20_POLY1305_SHA256
TLSv1.3    :TLS_AES_128_GCM_SHA256    TLSv1.2    :ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2    :DHE-RSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-RSA-CHACHA20-POLY1305
TLSv1.2    :DHE-RSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2    :DHE-RSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-ECDSA-AES256-SHA384 TLSv1.2    :ECDHE-RSA-AES256-SHA384
TLSv1.2    :DHE-RSA-AES256-SHA256     TLSv1.2    :ECDHE-ECDSA-AES128-SHA256
TLSv1.2    :ECDHE-RSA-AES128-SHA256   TLSv1.2    :DHE-RSA-AES128-SHA256
TLSv1.0    :ECDHE-ECDSA-AES256-SHA    TLSv1.0    :ECDHE-RSA-AES256-SHA
SSLv3      :DHE-RSA-AES256-SHA        TLSv1.0    :ECDHE-ECDSA-AES128-SHA
TLSv1.0    :ECDHE-RSA-AES128-SHA      SSLv3      :DHE-RSA-AES128-SHA
TLSv1.2    :RSA-PSK-AES256-GCM-SHA384 TLSv1.2    :DHE-PSK-AES256-GCM-SHA384
TLSv1.2    :RSA-PSK-CHACHA20-POLY1305 TLSv1.2    :DHE-PSK-CHACHA20-POLY1305
TLSv1.2    :ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2    :AES256-GCM-SHA384
TLSv1.2    :PSK-AES256-GCM-SHA384     TLSv1.2    :PSK-CHACHA20-POLY1305
TLSv1.2    :RSA-PSK-AES128-GCM-SHA256 TLSv1.2    :DHE-PSK-AES128-GCM-SHA256
TLSv1.2    :AES128-GCM-SHA256         TLSv1.2    :PSK-AES128-GCM-SHA256
TLSv1.2    :AES256-SHA256             TLSv1.2    :AES128-SHA256
TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA384 TLSv1.0    :ECDHE-PSK-AES256-CBC-SHA
SSLv3      :SRP-RSA-AES-256-CBC-SHA   SSLv3      :SRP-AES-256-CBC-SHA
TLSv1.0    :RSA-PSK-AES256-CBC-SHA384 TLSv1.0    :DHE-PSK-AES256-CBC-SHA384
SSLv3      :RSA-PSK-AES256-CBC-SHA    SSLv3      :DHE-PSK-AES256-CBC-SHA
SSLv3      :AES256-SHA                TLSv1.0    :PSK-AES256-CBC-SHA384
SSLv3      :PSK-AES256-CBC-SHA        TLSv1.0    :ECDHE-PSK-AES128-CBC-SHA256
TLSv1.0    :ECDHE-PSK-AES128-CBC-SHA  SSLv3      :SRP-RSA-AES-128-CBC-SHA
SSLv3      :SRP-AES-128-CBC-SHA       TLSv1.0    :RSA-PSK-AES128-CBC-SHA256
TLSv1.0    :DHE-PSK-AES128-CBC-SHA256 SSLv3      :RSA-PSK-AES128-CBC-SHA
SSLv3      :DHE-PSK-AES128-CBC-SHA    SSLv3      :AES128-SHA
TLSv1.0    :PSK-AES128-CBC-SHA256     SSLv3      :PSK-AES128-CBC-SHA
---
Ciphers common between both SSL end points:
TLS_AES_128_GCM_SHA256     TLS_CHACHA20_POLY1305_SHA256 TLS_AES_256_GCM_SHA384
ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-RSA-CHACHA20-POLY1305 ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA     ECDHE-ECDSA-AES128-SHA     ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES256-SHA       AES128-GCM-SHA256          AES256-GCM-SHA384
AES128-SHA                 AES256-SHA
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA1:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512
```

Figure 23: Certificate verification success.

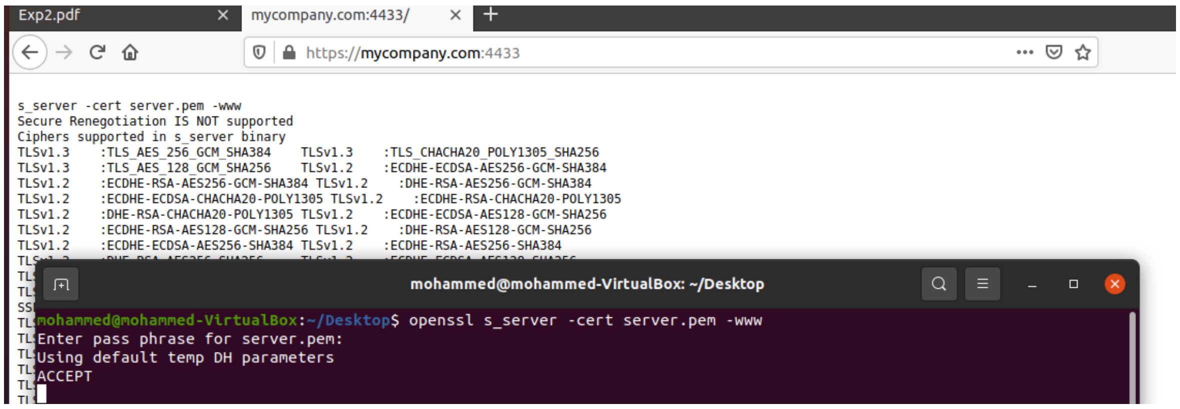
7. We modified a single byte of the file **server.pem** and restarted the server, then reloaded the URL, and we noticed that it was unable to load server certificate private key file, as shown in Figure 24.


Exp2.pdf
Problem loading page
https://mycompany.com:4433
mohammed@mohammed-VirtualBox: ~/Desktop

```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
unable to load server certificate private key file
140682910725440:error:0D07207B:asn1 encoding routines:ASN1_get_object:header too long:../crypto/asn1/asn1_lib.c:101:
140682910725440:error:0D068066:asn1 encoding routines:asn1_check_tlen:bad object header:../crypto/asn1/tasn_dec.c:113
7:
140682910725440:error:0D07803A:asn1 encoding routines:asn1_item_embed_d2i:nested asn1 error:../crypto/asn1/tasn_dec.c
:309:Type=RSAPrivateKey
140682910725440:error:04093004:rsa routines:old_rsa_priv_decode:RSA lib:../crypto/rsa/rsa_ameth.c:133:
140682910725440:error:0D07207B:asn1 encoding routines:ASN1_get_object:header too long:../crypto/asn1/asn1_lib.c:101:
140682910725440:error:0D068066:asn1 encoding routines:asn1_check_tlen:bad object header:../crypto/asn1/tasn_dec.c:113
7:
140682910725440:error:0D07803A:asn1 encoding routines:asn1_item_embed_d2i:nested asn1 error:../crypto/asn1/tasn_dec.c
:309:Type=PKCS8_PRIV_KEY_INFO
140682910725440:error:0907B00D:PEM routines:PEM_read_bio_PrivateKey:ASN1 lib:../crypto/pem/pem_pkey.c:88:
```

Figure 24: Error loading key.

8. We restored the original **server.pem** file and restarted the server again, then reloaded the URL and we noticed that the certificate verification passed unlike the preivious step duo to changing a single byte, as shown in Figure 25.


Exp2.pdf
mycompany.com:4433/
https://mycompany.com:4433
mohammed@mohammed-VirtualBox: ~/Desktop

```
s_server -cert server.pem -www
Secure Renegotiation IS NOT supported
Ciphers supported in s_server binary
TLSv1.3    :TLS_AES_256_GCM_SHA384    TLSv1.3    :TLS_CHACHA20_POLY1305_SHA256
TLSv1.3    :TLS_AES_128_GCM_SHA256    TLSv1.2    :ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2    :DHE-RSA-AES256-GCM-SHA384
TLSv1.2    :ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-RSA-CHACHA20-POLY1305
TLSv1.2    :DHE-RSA-CHACHA20-POLY1305 TLSv1.2    :ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2    :DHE-RSA-AES128-GCM-SHA256
TLSv1.2    :ECDHE-ECDSA-AES256-SHA384 TLSv1.2    :ECDHE-RSA-AES256-SHA384
```

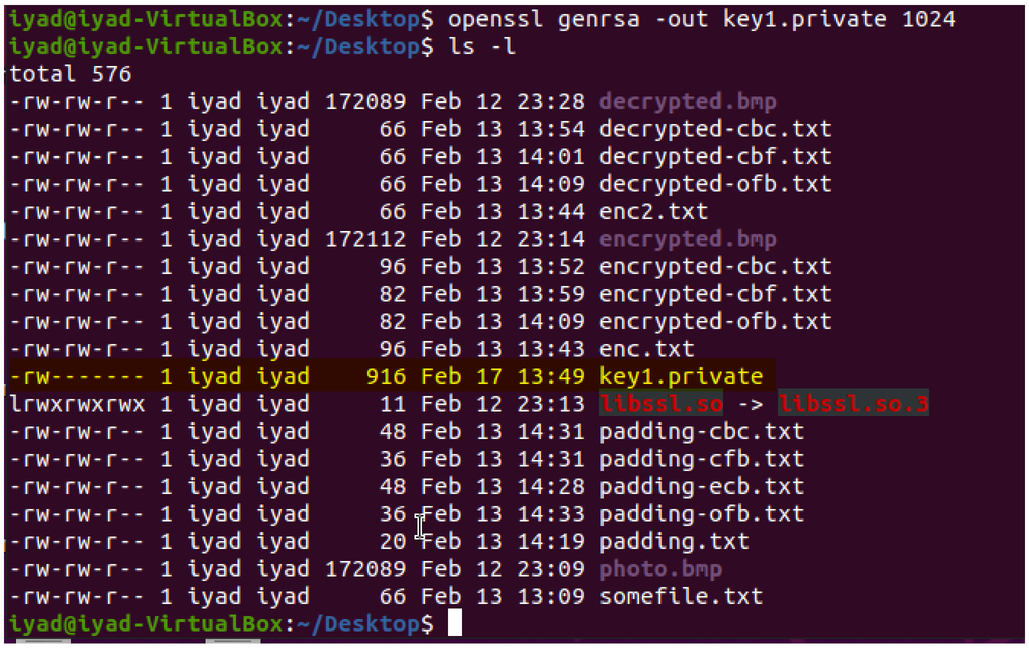
```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

Figure 25: Certificate pass.

3.4 Digital Signatures Creation

Question: What is the purpose of digital signatures?
Digital signatures create a virtual fingerprint that is unique to a person or entity and are used to identify users and protect information in digital messages or documents.

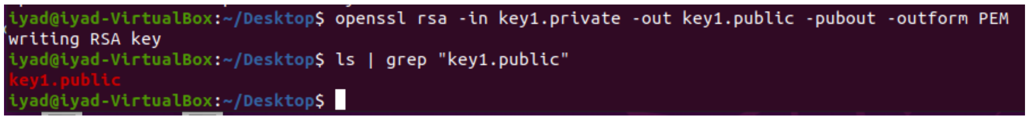
1. We generated an RSA private key using the following command:
   openssl genrsa -out key1.private 1024, as shown in Figure 26.



```
iyad@iyad-VirtualBox:~/Desktop$ openssl genrsa -out key1.private 1024
iyad@iyad-VirtualBox:~/Desktop$ ls -l
total 576
-rw-rw-r-- 1 iyad iyad 172089 Feb 12 23:28 decrypted.bmp
-rw-rw-r-- 1 iyad iyad     66 Feb 13 13:54 decrypted-cbc.txt
-rw-rw-r-- 1 iyad iyad     66 Feb 13 14:01 decrypted-cbf.txt
-rw-rw-r-- 1 iyad iyad     66 Feb 13 14:09 decrypted-ofb.txt
-rw-rw-r-- 1 iyad iyad     66 Feb 13 13:44 enc2.txt
-rw-rw-r-- 1 iyad iyad 172112 Feb 12 23:14 encrypted.bmp
-rw-rw-r-- 1 iyad iyad     96 Feb 13 13:52 encrypted-cbc.txt
-rw-rw-r-- 1 iyad iyad     82 Feb 13 13:59 encrypted-cbf.txt
-rw-rw-r-- 1 iyad iyad     82 Feb 13 14:09 encrypted-ofb.txt
-rw-rw-r-- 1 iyad iyad     96 Feb 13 13:43 enc.txt
-rw------- 1 iyad iyad    916 Feb 17 13:49 key1.private
lrwxrwxrwx 1 iyad iyad     11 Feb 12 23:13 libssl.so -> libssl.so.3
-rw-rw-r-- 1 iyad iyad     48 Feb 13 14:31 padding-cbc.txt
-rw-rw-r-- 1 iyad iyad     36 Feb 13 14:31 padding-cfb.txt
-rw-rw-r-- 1 iyad iyad     48 Feb 13 14:28 padding-ecb.txt
-rw-rw-r-- 1 iyad iyad     36 Feb 13 14:33 padding-ofb.txt
-rw-rw-r-- 1 iyad iyad     20 Feb 13 14:19 padding.txt
-rw-rw-r-- 1 iyad iyad 172089 Feb 12 23:09 photo.bmp
-rw-rw-r-- 1 iyad iyad     66 Feb 13 13:09 somefile.txt
iyad@iyad-VirtualBox:~/Desktop$
```

Figure 26: Private key generation.

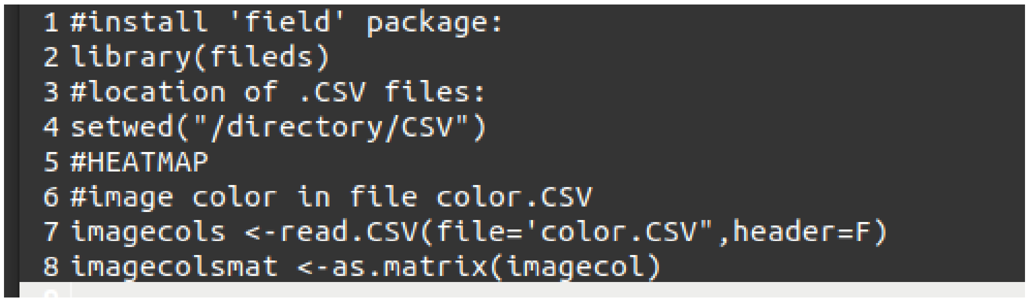
2. We generated an RSA public key using the following command (from the private key):
   openssl rsa -in key1.private -out key1.public -pubout -outform PEM, as shown in Figure 27.



```
iyad@iyad-VirtualBox:~/Desktop$ openssl rsa -in key1.private -out key1.public -pubout -outform PEM
writing RSA key
iyad@iyad-VirtualBox:~/Desktop$ ls | grep "key1.public"
key1.public
iyad@iyad-VirtualBox:~/Desktop$
```

Figure 27: Public key generation.

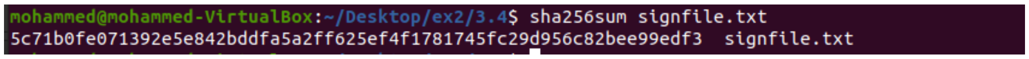
3. We created a file named **signfile.txt** and filled it with arbitrary text, as shown in Figure 28.



```
#install 'field' package:
library(fileds)
#location of .CSV files:
setwed("/directory/CSV")
#HEATMAP
#image color in file color.CSV
imagecols <-read.CSV(file='color.CSV",header=F)
imagecolsmat <-as.matrix(imagecol)
```

Figure 28: File creation and filling.

4. We generated a hash code using SHA256 for the file **signfile.txt**, as shown in Figure 29.

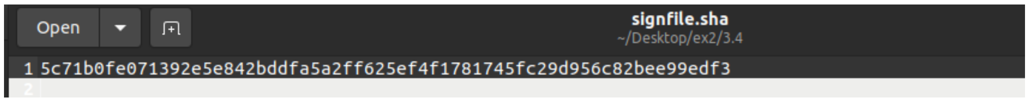


```
mohammed@mohammed-VirtualBox:~/Desktop/ex2/3.4$ sha256sum signfile.txt
5c71b0fe071392e5e842bddfa5a2ff625ef4f1781745fc29d956c82bee99edf3  signfile.txt
```

Figure 29: Hash code generation.

Question: How will you do that?
By using the dgst function in the openssl tool.

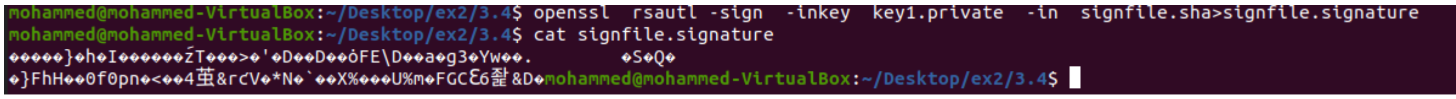
5. We saved the generated hash code in a file named **signfile.sha**, as shown in Figure 30.



```
5c71b0fe071392e5e842bddfa5a2ff625ef4f1781745fc29d956c82bee99edf3
```

Figure 30: Saving hash.

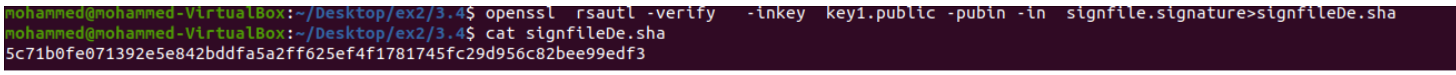
6. We encrypted the hash code using RSA private key using the following command:
openssl pkeyutl -encrypt -inkey key1.private -in signfile.sha ¿ signfile.signature , as shown in Figure 31.



```
mohammed@mohammed-VirtualBox:~/Desktop/ex2/3.4$ openssl  rsautl -sign  -inkey  key1.private  -in  signfile.sha>signfile.signature
mohammed@mohammed-VirtualBox:~/Desktop/ex2/3.4$ cat signfile.signature
�����}�h�I������źT���>�'�D��D��ȯFE\D��a�g3�Yw��.       �S�Q�
�}FhH��0f0pn�<��4茧&rȼV�*N�`��X%���U%m�FGCƐ6잝&D�mohammed@mohammed-VirtualBox:~/Desktop/ex2/3.4$ █
```

Figure 31: Encrypting hash code.

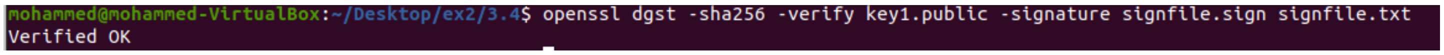
7. We decrypted the encrypted hash code using the RSA public key, as shown in Figure 32.



```
mohammed@mohammed-VirtualBox:~/Desktop/ex2/3.4$ openssl  rsautl -verify   -inkey  key1.public -pubin -in  signfile.signature>signfileDe.sha
mohammed@mohammed-VirtualBox:~/Desktop/ex2/3.4$ cat signfileDe.sha
5c71b0fe071392e5e842bddfa5a2ff625ef4f1781745fc29d956c82bee99edf3
```

Figure 32: Decrypting hash code.

8. We used the hash code to confirm the integrity of the file **signfile.txt**, as shown in Figure 33.

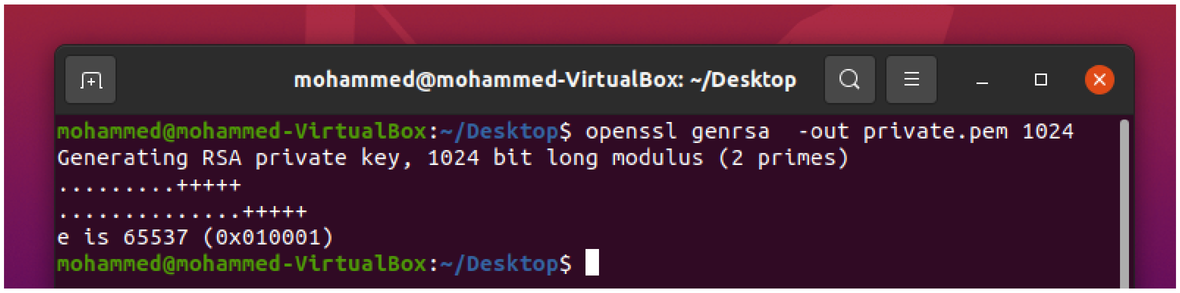


```
mohammed@mohammed-VirtualBox:~/Desktop/ex2/3.4$ openssl dgst -sha256 -verify key1.public -signature signfile.sign signfile.txt
Verified OK
```

Figure 33: Hash code integrity check.

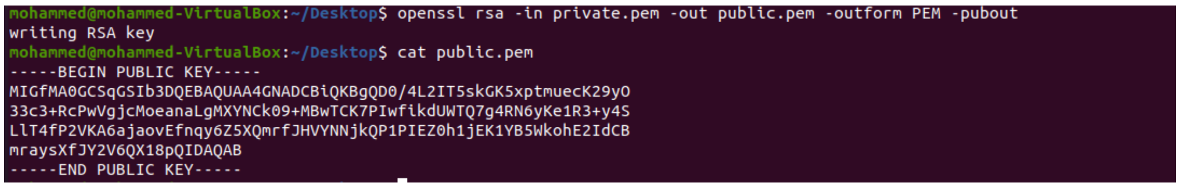
3.5 To DO

1. We prepared a file named **message.txt** that contains a 16-byte message.
2. We prepared a 1024-bit RSA public/private key pair, as shown in Figures 34 & 35.



```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl genrsa  -out private.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.........+++++
..............+++++
e is 65537 (0x010001)
mohammed@mohammed-VirtualBox:~/Desktop$
```

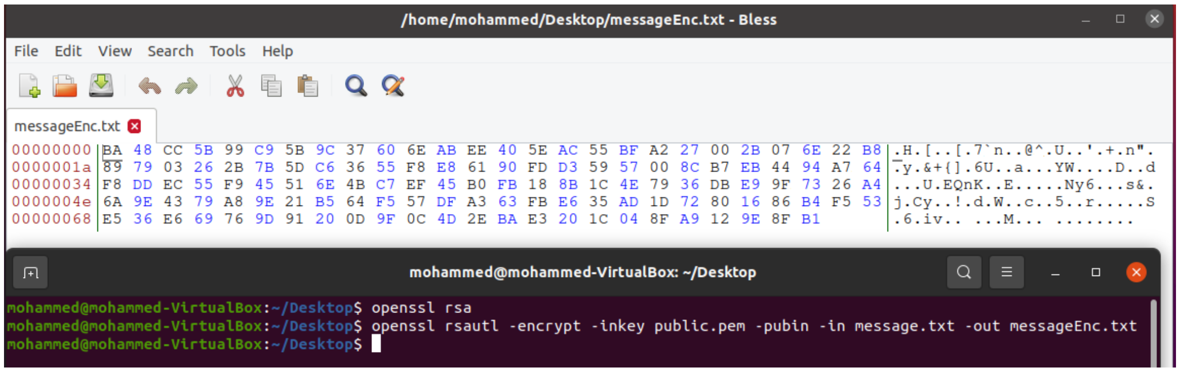
Figure 34: Private key generation.



```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl rsa -in private.pem -out public.pem -outform PEM -pubout
writing RSA key
mohammed@mohammed-VirtualBox:~/Desktop$ cat public.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQD0/4L2IT5skGK5xptmuecK29yO
33c3+RcPwVgjcMoeanaLgMXYNCk09+MBwTCK7PIwfikdUWTQ7g4RN6yKe1R3+y4S
LlT4fP2VKA6ajaovEfnqy6Z5XQmrfJHVYNNjkQP1PIEZ0h1jEK1YB5WkohE2IdCB
mraysXfJY2V6QX18pQIDAQAB
-----END PUBLIC KEY-----
```

Figure 35: Public key generation.

3. We encrypted the message using the public key and saved the output in **message-enc.txt**, as shown in Figure 36.



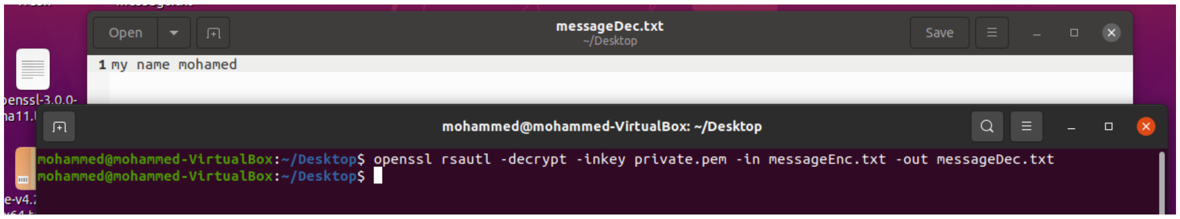
/home/mohammed/Desktop/messageEnc.txt - Bless
File Edit View Search Tools Help
messageEnc.txt

```
00000000 BA 48 CC 5B 99 C9 5B 9C 37 60 6E AB EE 40 5E AC 55 BF A2 27 00 2B 07 6E 22 B8 .H.[..[.7`n..@^.U..'.+.n".
0000001a 89 79 03 26 2B 7B 5D C6 36 55 F8 E8 61 90 FD D3 59 57 00 8C B7 EB 44 94 A7 64 .y.&+{].6U..a...YW....D..d
00000034 F8 DD EC 55 F9 45 51 6E 4B C7 EF 45 B0 FB 18 8B 1C 4E 79 36 DB E9 9F 73 26 A4 ...U.EQnK..E.....Ny6...s&.
0000004e 6A 9E 43 79 A8 9E 21 B5 64 F5 57 DF A3 63 FB E6 35 AD 1D 72 80 16 86 B4 F5 53 j.Cy..!.d.W..c..5..r.....S
00000068 E5 36 E6 69 76 9D 91 20 0D 9F 0C 4D 2E BA E3 20 1C 04 8F A9 12 9E 8F B1       .6.iv.. ...M... ........
```

```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl rsa
mohammed@mohammed-VirtualBox:~/Desktop$ openssl rsautl -encrypt -inkey public.pem -pubin -in message.txt -out messageEnc.txt
mohammed@mohammed-VirtualBox:~/Desktop$
```

Figure 36: Message encryption.

4. We decrypted the message using the private key, as shown in Figure 37.

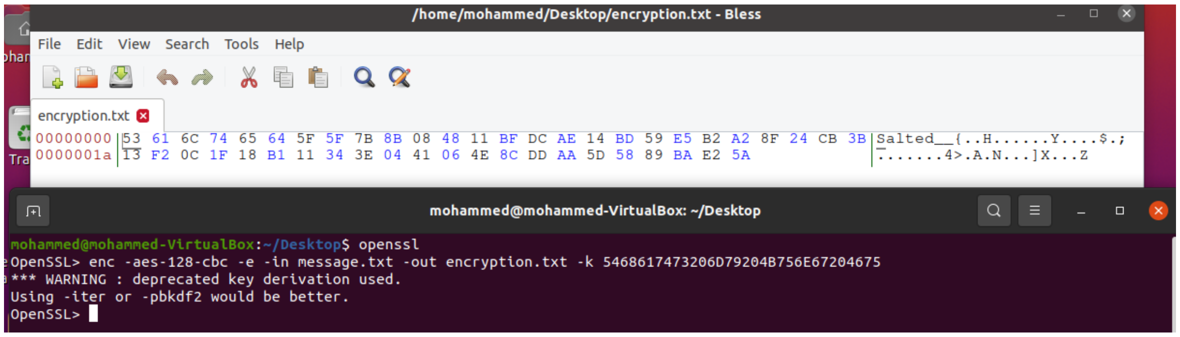


```
1 my name mohamed
```

```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl rsautl -decrypt -inkey private.pem -in messageEnc.txt -out messageDec.txt
mohammed@mohammed-VirtualBox:~/Desktop$
```

Figure 37: Message decryption.

5. We encrypted the message using 128-bit AES key, as shown in Figure 38.



/home/mohammed/Desktop/encryption.txt - Bless
File Edit View Search Tools Help
encryption.txt

```
00000000 53 61 6C 74 65 64 5F 5F 7B 8B 08 48 11 BF DC AE 14 BD 59 E5 B2 A2 8F 24 CB 3B Salted__{..H......Y....$.;
0000001a 13 F2 0C 1F 18 B1 11 34 3E 04 41 06 4E 8C DD AA 5D 58 89 BA E2 5A             .......4>.A.N...]X...Z
```

mohammed@mohammed-VirtualBox: ~/Desktop

```
mohammed@mohammed-VirtualBox:~/Desktop$ openssl
OpenSSL> enc -aes-128-cbc -e -in message.txt -out encryption.txt -k 5468617473206D79204B756E67204675
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
OpenSSL>
```

Figure 38: Message AES encryption.

6. We have compared the time spent for each operation and concluded that RSA is much slower than AES, because RSA is more computationally intensive than AES.

4 Conclusion

In brief, public key cryptography uses a pair of keys to encrypt and decrypt data to protect it against unauthorized access or use. Network users receive a public and private key pair from certification authorities. Public key cryptography is a highly important topic in cryptography science, and it revolutionized the cryptography world by using it in creating digital signatures, and we also noticed how it was used to gain certificates that help make the web as we know it a more secure place in a world full of hackers and attackers.

5 References

https://help.utk.edu/kb/index.php?func=showe=1960.
https://www.digitalrealty.com/blog/5-sdn-benefits-for-data-centers?fbclid=IwAR3_$OX9tmOmwWpTf9gkivkKf7gplKEW$
$text = Enter$.
https://digitalguardian.com/blog/what-public-key-cryptography.
https://en.wikipedia.org/wiki/Public-key$_c ryptography$.
https://gist.github.com/gwpl/2c7636f0b200cbfbe82cc9d4f6338585.