

Palestine
An-Najah National University
Faculty of Information Technology
Computerized Information



دولة فلسطين
جامعة النجاح الوطنية
كلية تكنولوجيا المعلومات
قسم شبكات وامن المعلومات

Experiment#1 Public Key Cryptography

Manar Eyad Harb	11924470
Dyaa Al-dein Ashraf Tummazeh	11924899
Yara Mohammad Sholi	11924207

Supervisor:

Dr. Amjad Hawash

Abstract

The main topic of this experiment was learning the basics of public-key encryption, digital signature and public-key infrastructure certificates and how to use OPENSSL in this scope.

Introduction

public-key cryptography is a class of cryptographic protocols based on algorithms. This method of cryptography requires two separate keys, one that is private or secret, and one that is public(reference: digitalguardian.com), this pair of keys is used to encrypt and decrypt data to protect it from unauthorized use.

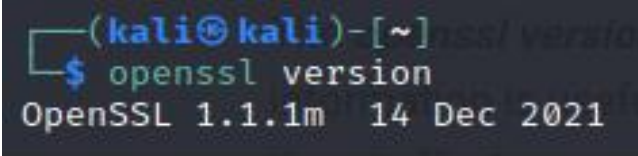
A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document, it's being used to solve the problem of tampering and impersonation in digital communications.

These two topics are very important in the security field and are being used always when there's a connection through a network, OPENSSL is such a tool that can be used to generate these things using the commands we are going to use in this experiment.

Procedures

I. Certificate Authority (CA) :

a. First, openssl tool is a must since all the work is going to be using it, checking if it's installed or not was done using the following command:



```
(kali@kali)-[~]
$ openssl version
OpenSSL 1.1.1m 14 Dec 2021
```

Figure1.a shows files in downloads

It returned the version of OPENSSL, which means that it's installed on the system.

b. Certificate authority gives digital certificates to users which then makes them certified, digital certificates are electronic credentials that bind the identity of the certificate owner to a pair of electronic encryption keys, that can be used to encrypt and sign information digitally.(reference:Wikipedia)

c. Commercial CAs charge money to issue certificates, examples on this type are :

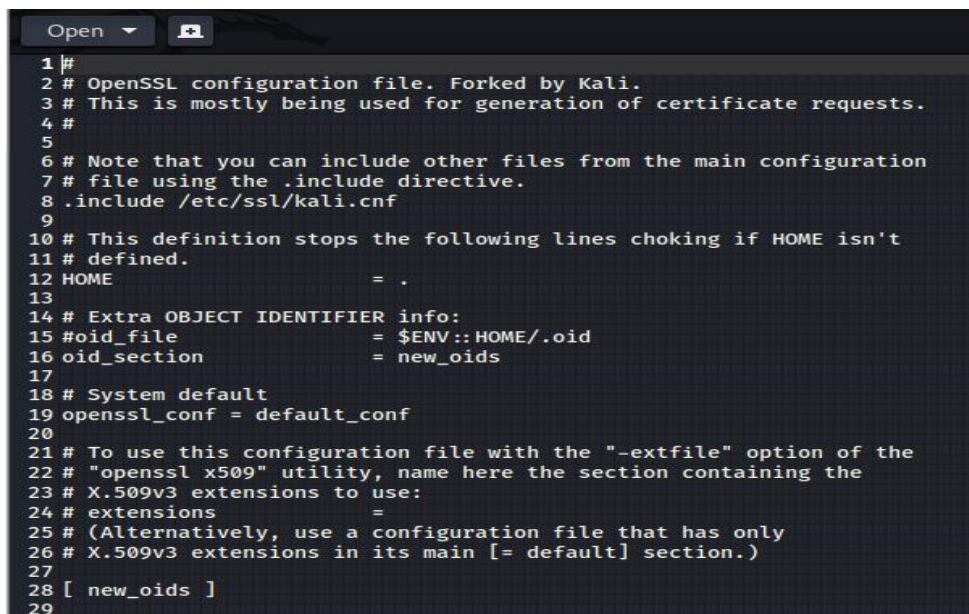
- 1- Amazon web service.
- 2- Cloudflare.
- 3- Google cloud platform.

(reference: Wikipedia)

d. Root CA is a CA that issues the root certificates that are used to sign other CA certificates. Root certificates are self-signed certificates.(reference: doubleoctupos.com), the root certificate is usually made trustworthy by some mechanism like a secure physical distribution. For example, Microsoft distributes root certificates to windows Desktop and windows phone 8, Apple distributes root certificates that are members of its own root program

e. Checking openssl configuration file was necessary, this was done by directing to the path where the configuration file is:

```
cd /usr/lib/ssl/
sudo gedit openssl.cnf
```

A screenshot of a terminal window with a dark background. At the top, there is a menu bar with 'Open' and a file icon. The terminal displays the contents of an OpenSSL configuration file, with line numbers 1 through 29 on the left. The text is as follows:

```
1 #  
2 # OpenSSL configuration file. Forked by Kali.  
3 # This is mostly being used for generation of certificate requests.  
4 #  
5  
6 # Note that you can include other files from the main configuration  
7 # file using the .include directive.  
8 .include /etc/ssl/kali.cnf  
9  
10 # This definition stops the following lines choking if HOME isn't  
11 # defined.  
12 HOME = .  
13  
14 # Extra OBJECT IDENTIFIER info:  
15 #oid_file = $ENV::HOME/.oid  
16 oid_section = new_oids  
17  
18 # System default  
19 openssl_conf = default_conf  
20  
21 # To use this configuration file with the "-extfile" option of the  
22 # "openssl x509" utility, name here the section containing the  
23 # X.509v3 extensions to use:  
24 # extensions =  
25 # (Alternatively, use a configuration file that has only  
26 # X.509v3 extensions in its main [= default] section.)  
27  
28 [ new_oids ]  
29
```

Figure1.e shows the result of previous commands

f. Creating a copy of openssl.cnf with name copy.cnf was done under the same directory of the original file.

g. After creating a copy, a sub-directory with the name demoCA was implemented.

h. Redirecting to the new directory “demoCA”, and then creating a text file called index.txt.

i. A new file with the name serial was created under demoCA, and a value of 1000 was stored inside it.

j. After that, creation of sub-directories under demoCA was needed: certs,crl,newcerts.

k. Opening the copy.cnf file and checking the values that was assigned to the new files and directories we have just created:

```
37 #####
38 [ ca ]
39 default_ca = CA_default # The default ca section
40 #####
41 [ CA_default ]
42 #####
43 dir = ./demoCA # Where everything is kept
44 certs = $dir/certs # Where the issued certs are kept
45 crl_dir = $dir/crl # Where the issued crl are kept
46 database = $dir/index.txt # database index file.
47 #unique_subject = no # Set to 'no' to allow creation of
48 # several certs with same subject.
49 new_certs_dir = $dir/newcerts # default place for new certs.
50
51 certificate = $dir/cacert.pem # The CA certificate
52 serial = $dir/serial # The current serial number
53 crlnumber = $dir/crlnumber # the current crl number
54 # must be commented out to leave a V1 CRL
55 crl = $dir/crl.pem # The current CRL
56 private_key = $dir/private/cakey.pem # The private key
57
58 x509_extensions = usr_cert # The extensions to add to the cert
59
60 # Comment out the following two lines for the "traditional"
61 # (and highly broken) format.
62 name_opt = ca_default # Subject Name options
63 cert_opt = ca_default # Certificate field options
64
65 # Extension copying option: use with caution.
66 # copy_extensions = copy
67
68 # Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
69 # so this is commented out by default to leave a V1 CRL.
70 # crlnumber must also be commented out to leave a V1 CRL.
71 # crl_extensions = crl_ext
72
73 default_days = 365 # how long to certify for
74 # default_crl_days = 30 # how long before next CRL
```

Figure1.k shows the corresponding files in opensslcopy.cnf

I. Next step is generating a self-signed certificate for the CA using the following command:

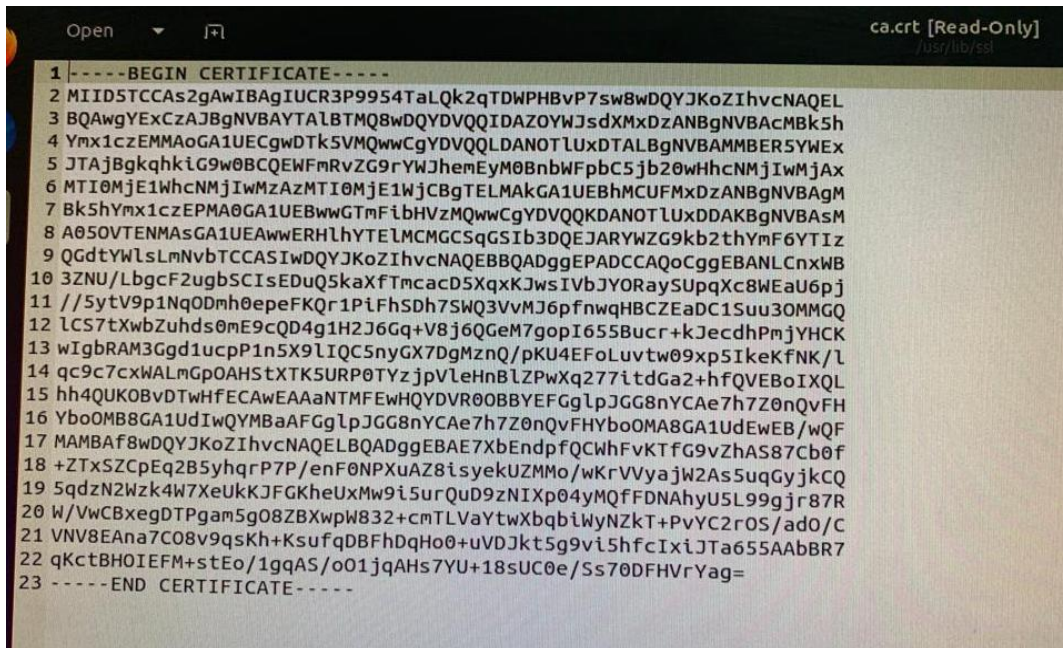
openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf

Some information was needed to be filled after executing the command:

```
student@linux: /usr/lib/ssl
student@linux:/usr/lib/ssl$ sudo openssl req -new -x509 -keyout ca.key -out ca.c
rt -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]:PS
State or Province Name (full name) [Some-State]:Nablus
Locality Name (eg, city) []:Nablus
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NNU
Organizational Unit Name (eg, section) []:NNU
Common Name (e.g. server FQDN or YOUR name) []:Dyaa
Email Address []:dodokabaza23@gmail.com
student@linux:/usr/lib/ssl$
```

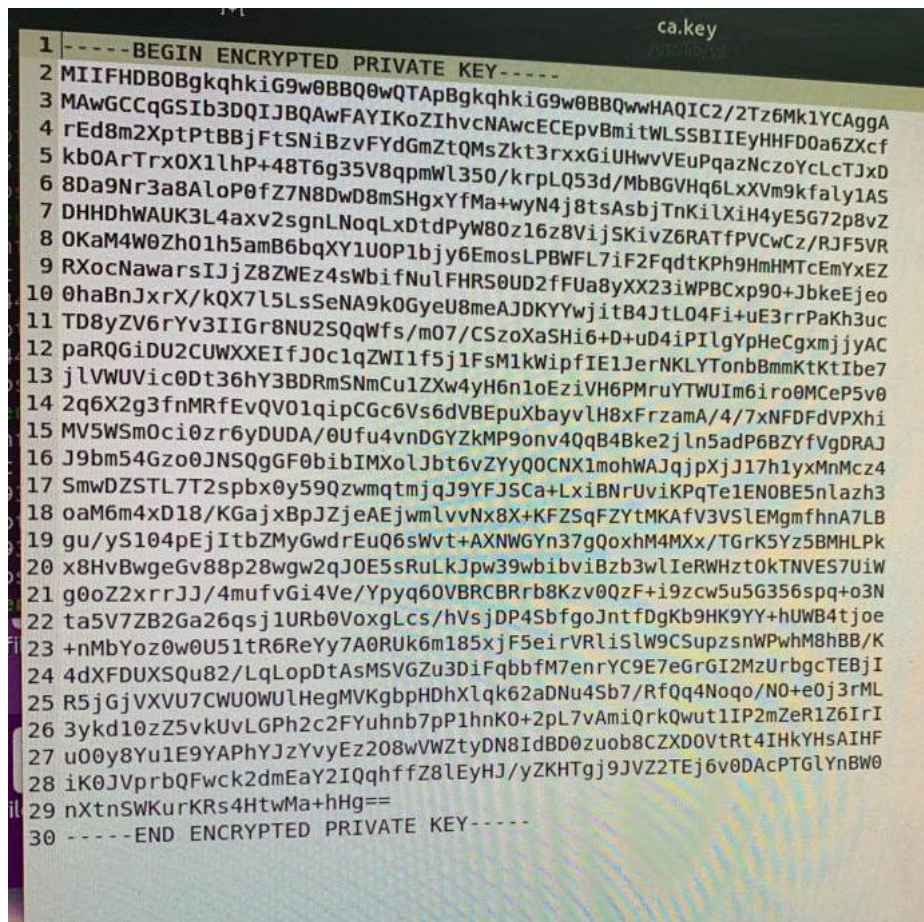
Figure1.l shows the result to the command and the info you need to fill it.

m. We have two files, ca.key that contains the private key and ca.crt that contains the public key.



```
1 -----BEGIN CERTIFICATE-----
2 MIIDSTCCAs2gAwIBAgIUCR3P9954TaLQk2qTDWPHBvP7sw8wDQYJKoZIhvcNAQEL
3 BQAwgYEXCzAjbGNVBAYTA1BTMQ8wDQYDVQQIDAZOYWJsZXMxXDAzANBgNVBACMBk5h
4 Ymx1czEMMAoGA1UECgwDTk5VMQwwCgYDVQQLDANOTLUXDTALBgNVBAMMBER5YWEx
5 JTAjBgkqhkiG9w0BCQEFMRvZG9rYWJhemEyM0BnbWFPbC5jb20wHhcNMjIwMjAx
6 MTI0MjE1WWhcNMjIwMjAxMjE1WjCBGTELMAKGA1UEBhMCUFRMxZANBgNVBAGM
7 Bk5hYmx1czEMMAoGA1UEBwwGTmFibHVzMjIwMjAxMjE1WjCBGTELMAKGA1UEBhMCUFRMxZANBgNVBAGM
8 A0S0VTENMASGA1UEAwVERHhYU01jQk5hYmx1czEMMAoGA1UEBhMCUFRMxZANBgNVBAGM
9 QGdtYWlsLnNvbTCCASiWdQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANLcNxBW
10 3ZNU/LbgCF2ugbSCISeDUQ5kaXFTmcacD5XqXKJwsIVbJYORaySUqXc8WEaU6pj
11 //5ytV9p1Nq0Dmh0epeFKQr1PiFhSDh7SWQ3VvMJ6pfnwqHBCZEaDC1Suu30MMGQ
12 lCS7tXwbZuhds0mE9cQD4g1H2J6Gq+V8j6QGeM7gopI655Bucr+kJecdHPmjYHCK
13 wIGBRAM3Ggd1ucpP1n5X9LIQC5nyGX7DgmznQ/pKU4EfoLuvtw09xp5IkeKfNK/L
14 qc9c7cxWALmGp0AHSTXTKSURP0TYzjpVleHnBLZPwXq277itdGa2+hfQVEBoIXQL
15 hh4QUK0BvDThwFECAwEAANtMFewHQYDVR0BBYEFgglpJGG8nYCAe7h7Z0nQvFH
16 YboOMB8GA1UdIwQYMBaAFGglpJGG8nYCAe7h7Z0nQvFHYboOMA8GA1UdEwEB/wQF
17 MAMBAf8wDQYJKoZIhvcNAQELBQADggEBAE7XbEndpfQCWhFvKTFG9vZhaS87Cb0F
18 +ZTxSZCPeEq2B5yhqrP7P/enF0NPXuAZ8isyekUZMMo/wKRvVyaJW2As5uqGyjkCQ
19 5qdzN2WzK4W7XeUkKJFGKheUxMw9iSurQuD9zNIXP04yMQfFDNAhyU5L99gjr87R
20 W/VwCBXegDTPgam5g08ZBXwpw832+cmTLVaYtwXbqbiWynZKT+PvYC2r0S/ad0/C
21 VNV8EAna7C08v9qsKh+KsufqDBFhDqHo0+uVDJkt5g9vi5hfcIxiJTa655AABBR7
22 qKctBHOIEFM+stEo/1gqAS/o01jqAHs7YU+18sUC0e/Ss70DFHVRyag=
23 -----END CERTIFICATE-----
```

Figure1.m.1 shows ca.crt file content



```
1 -----BEGIN ENCRYPTED PRIVATE KEY-----
2 MIIFHDB0BgkqhkiG9w0BBQ0wQTApgBgkqhkiG9w0BBQwwHAQIC2/2Tz6Mk1YCAggA
3 MAwGCCqGSIb3DQIJBQAwFAYIKoZIhvcNAwcECEpvBmitWLS5BIEYHFD0a6ZxcF
4 rEd8m2XptPtBBjFtSNiBzvFYdGmZtQMsZkt3rxxGiUHwvVEuPqazNczoYcLTJxD
5 kb0ArTrx0X1lhP+48T6g35V8qpmWl350/krpLQ53d/MbBGVHq6LxXVm9kfaly1AS
6 8Da9Nr3a8AloP0fZ7N8DwD8mSHgxYfMa+wyn4j8tsAsbjTnKiLXiH4YE5G72p8vZ
7 DHHDHWAUK3L4axv2sgnLNoqLxDtdPyW80z16z8VijSKivZ6RATfPVCwCz/RJF5VR
8 OKaM4W0Zh01h5amB6bqXYLU0P1bjy6EmosLPBwFL7iF2FqdtKPh9HmHMTcEmYxEZ
9 RXocNawarsIJjZ8ZWEz4sWbifNulFHR50UD2fFua8yXX23iWPBCxp90+JbkeEjeo
10 0haBnJxR/xkQX7L5LsSeNA9k0GyeU8meAJDKYyWjitB4JtL04Fi+uE3rrPaKh3uc
11 TD8yZV6rYv3IIGr8NU2SQWfs/m07/CSzoXaSHi6+D+uD4PiIgYpHeCgxmjjyAC
12 parQGiDU2CUWXXEIfJ0c1qZWI1f5j1fSmlkWiPfiE1JerNKLYTonbBmmKtKiIbe7
13 jLVWUvic0Dt36hY3BDRmSnmCu1ZXw4yH6n1oEziVH6PMruYTWUIm6iro0MCP5v0
14 2q6X2g3fnMRfEvQV01qipCGc6Vs6dVBEpuXbayvLH8xFrzama/4/7xNFDfVpXhi
15 MV5WSm0ci0zr6yDUDA/0ufu4vnDGYZKMP9onv4QqB4Bke2jln5adP6BZYfVgDRAJ
16 J9bm54Gzo0JNSQGGF0bibIMXolJbt6vZyYQ0CNX1mohWAJqjpXj17hlyxMnMcZ4
17 SmwDZSTL7T2spbx0y59QzwmqtmjQJ9YFJSCa+LxiBNrUviKPqTe1EN0BE5nlazh3
18 oaM6m4xD18/KGajxBpJZjeAEjwmlvvNx8X+KFZSqFZYtMKAfV3VSLemgmfnA7LB
19 gu/yS104pEjItbZMyGwdrEuQ6sWvt+AXNWGYn37gQoxhM4MXx/TGRK5Yz5BMHLpk
20 x8HvBwgeGv88p28wgw2qJ0E5sRuLkJPw39wbibviBzb3wLiErWHztOkTNVES7UiW
21 g0oZ2xrrJJ/4mufvGi4Ve/Ypyq60VBRBCBRb8Kzv0QzF+i9zcw5u5G356spq+o3N
22 ta5V7ZB2Ga26qsj1URb0VoxgLcs/hVsJDP4SbfgoJntfDgKb9HK9YY+uHwB4tjoe
23 +nMbYoz0w0U51tR6ReYy7A0RUk6m185xjF5eirVRLiSLW9CSupzsnWPwhM8hBB/K
24 4dXFDUXSQu82/LqLopDtAsMSVGZu3DiFqbbfM7enrYC9E7eGrGI2MzUrbgcTEBjI
25 R5jGjVXVU7CWUOWULHegMVKGbPHDhXlqk62aDNU4Sb7/RfQq4Noqo/N0+e0j3rML
26 3ykd10zZ5vkUvLGPh2c2FYuhnb7pP1hnK0+2pL7vAmiQrkQwut1IP2mZeR1Z6IrI
27 u00y8Yu1E9YAPHYJzYvYez208wVWZtyDN8IdBD0zuob8CXD0VtRt4IHkYHsAIHF
28 iK0JVprbQFwck2dmEaY2IQqhffZ8LEyHJ/yZKHTgj9JVZ2TEj6v0DacPTGLynBW0
29 nXtnSWKurKRs4HtwMa+hHg==
30 -----END ENCRYPTED PRIVATE KEY-----
```

Figure1.m.2 shows ca.key file content

In file ca.crt the file contains the public key, which is available to the public domain as its part of the SSL certificate and is made known to our server.

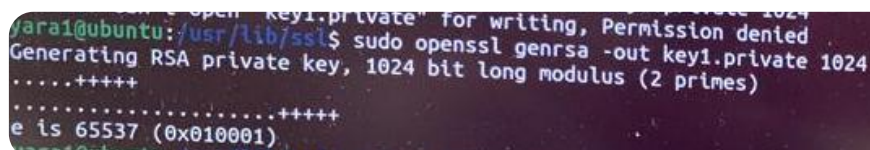
In file ca.key the file contains the private key which must correspond to the generated CSR, and it needs to match the certificate created from the CSR.

II. Giving Certificate for Clients:

II.i. Key-Pair Generation By the Client:

We assumed that a company called MyCompany.com wanted to get a digital certificate from us as a CA, to do that there should be a generation for a public-private key pair as a first step, and this was done with the following command:

openssl genrsa -aes128 -out server.key 1024



```

yara1@ubuntu:~$ openssl genrsa -out key1.private 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
yara1@ubuntu:~$
```

Figure2.i.1 shows the output when generate a key

As it appears the used algorithm was RSA, with 1024 bit long key, AES was use in the previous command because SSL uses symmetric cryptography with the session key after the initial handshake is done([reference: tutorialsteacher.com](http://tutorialsteacher.com)).

Now, when running the following command:

openssl rsa -in server.key -text

The output will look like:


```

student@linux:~$ sudo openssl rsa -in server.key -text
Enter pass phrase for server.key:
RSA Private-Key: (1024 bit, 2 primes)
modulus:
 00:d6:9b:30:de:54:63:6b:78:11:67:bf:4e:41:d5:
 57:c6:5c:3d:0c:08:93:2b:73:1d:76:9f:f3:d3:0f:
 b3:3d:fa:a9:79:4a:d3:88:36:91:d6:d0:dd:dd:1a:
 04:a7:04:8f:f8:66:ae:ea:7b:92:76:25:6e:21:da:
 28:3b:85:29:0d:4c:ce:2a:d2:2d:06:8e:10:69:15:
 88:de:29:6f:90:27:ff:71:70:82:37:c2:b9:c7:f9:
 fe:e1:3c:9e:b9:6c:af:1e:c2:b9:60:76:2c:e5:ca:
 2d:b1:ad:f4:2d:18:45:0a:ff:bc:06:45:bf:e1:07:
 eb:32:e4:d9:7d:98:9e:77:41
publicExponent: 65537 (0x10001)
privateExponent:
 4b:9b:57:47:2c:f1:ed:dd:da:3b:f3:e1:2d:3e:6d:
 73:1b:f2:01:f8:4f:69:22:60:41:f7:ae:5e:52:5f:
 12:b9:e9:d5:2a:b0:85:af:bc:07:b3:84:46:ae:30:
 ef:6a:a3:12:3f:92:e6:57:6c:cd:24:f8:bd:02:6f:
 e4:30:50:ba:92:54:e8:d1:8f:65:3b:fb:c9:8b:b1:
 f5:ea:75:1e:38:ed:52:aa:c9:d4:dd:58:d2:70:96:
 1a:39:d3:81:7a:a2:47:f2:9f:c1:88:cf:cf:90:40:
 d5:66:bb:91:ad:e1:b5:9a:eb:e0:64:41:66:6c:e1:
 07:3a:8a:ad:b9:5f:cb:01
prime1:
 00:f9:b1:57:64:56:c2:17:a4:c1:36:b0:73:3a:e4:
 ff:e7:40:8a:93:66:a1:e3:18:30:4c:50:84:7b:7f:
 d0:8d:bf:39:c4:91:af:d2:4c:e5:84:cc:cf:10:0c:
 24:9c:1a:f9:f5:62:cd:03:8d:69:94:98:10:f0:c3:
 bb:f4:ce:c8:51
prime2:
 00:dc:06:f5:a9:81:f8:a1:79:21:df:51:08:3c:af:
 90:ed:c2:b5:96:e8:35:ad:19:04:c6:74:aa:36:70:
 dd:84:02:ca:d3:c8:22:ca:82:df:6a:fe:79:e3:89:
 51:a2:b3:44:f4:d2:81:b6:d1:6e:9c:52:47:17:22:
 3a:18:1f:f3:f1
exponent1:
 00:9c:a7:5b:b9:f3:08:90:c6:e8:05:c4:cc:76:ad:
 c7:b3:b1:75:7f:a7:0b:78:2b:eb:d9:65:46:c9:28:
 d6:92:a7:df:b9:68:e1:d3:62:35:39:8d:39:77:ca:
 f6:89:0e:0f:b6:99:87:20:6c:6f:4a:2f:e1:a8:fc:
 54:86:f1:d9:11
exponent2:
 3a:c6:b5:2b:51:e7:fc:22:a2:b4:c8:2d:be:20:5c:
 43:d7:66:b0:e8:59:26:63:6f:8e:20:2e:34:1a:d4:
 61:e6:69:0e:48:01:5b:3d:b8:b7:19:41:b0:51:c7:
 78:02:11:a2:a0:f5:3b:c4:18:9b:33:b3:7f:ef:89:
 bb:56:15:d1
coefficient:
 00:f4:99:e7:7f:99:3f:53:f9:cf:fe:07:e1:0b:f6:
 52:0c:e7:72:05:85:3f:0e:0c:d3:fd:cf:45:c8:72:
 82:47:33:35:4b:cc:47:60:cc:2e:fe:79:22:95:3c:
 2f:96:7a:b5:76:f8:c4:be:fd:02:fd:97:8a:38:03:
 4c:2e:f4:6d:95
writing RSA key
-----BEGIN RSA PRIVATE KEY-----

```

Figure2.i.2 shows the output the previous command

```

bb:56:15:d1
coefficient:
 00:f4:99:e7:7f:99:3f:53:f9:cf:fe:07:e1:0b:f6:
 52:0c:e7:72:05:85:3f:0e:0c:d3:fd:cf:45:c8:72:
 82:47:33:35:4b:cc:47:60:cc:2e:fe:79:22:95:3c:
 2f:96:7a:b5:76:f8:c4:be:fd:02:fd:97:8a:38:03:
 4c:2e:f4:6d:95
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDwmzDeVGNreBFnv05B1VfGXD0MCJMrcx12n/PTb7M9+qL5StOI
NpHW0N3dGgSnBI/4Zq7qe5J2JW4h2ig7hskNTM4q0i0GjhBpFYjeKW+QJ/9xcII3
wrnH+f7hPJ65bk8ewrlgd1zly12xrfQtGEUK/7wGRb/hB+sy5Nl9mJ53QQIDAQAB
AoGASStXRYzx7d3a0/PhLT5tcxvyAfhPaSJgQfeuxLJfErnp1Sqwha+8B70ERq4w
72qJEj+S5ldszST4vQJv5DBQuPJUGNGPZTv7yYux9ep1HjttUqrJ1N1Y0nCWGjnT
gXqIR/KfwYjPz5BA1Wa7ka3htZrr4GRBZmzhBzqKrb1fywECQD5sVdkVsIXpME2
sHM65P/nQIqTZqHjGDBMUIR7f9CNvznEka/STOWEzM8QDCScGvn1Ys0DjWmUmBDw
w7v0zshRAKEA3Ab1qYH4oXkh31EIPK+Q7cK1lug1rRkExnSqNnDdhALK08gtYoLf
av5544LRorNE9NKBttFunFJHFyI6GB/z8QJBAJynW7nzCJDG6AXEzHatx70xdX+n
C3gr69llRsko1pKn37Lo4dNlNTmNOXfK9okOD7aZhyBsb0ov4aj8VIBx2RECQDrG
tStR5/wlorTILb4gXEPXZrDoWSZjb44gLjQa1GHnaQSIaVs9uLcZQbBRx3gCEaKg
9TveGJszs3/v1btWfDECQD0med/mt9T+c/+B+EL9lIM53IFhT80DNP9z0XicoJH
MzVLZEdgzC7+eSKVPC+WerV2+MS+/QL9L4o4A0wu9G2V
-----END RSA PRIVATE KEY-----
student@linux:~$

```

Figure2.i.3 shows the output the previous command

II.ii. 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.

Use this command to start a new certificate using company private key (server.key):

```
openssl req -new -key server.key -out server.csr -config  
openssl.cnf
```

And fill some information:

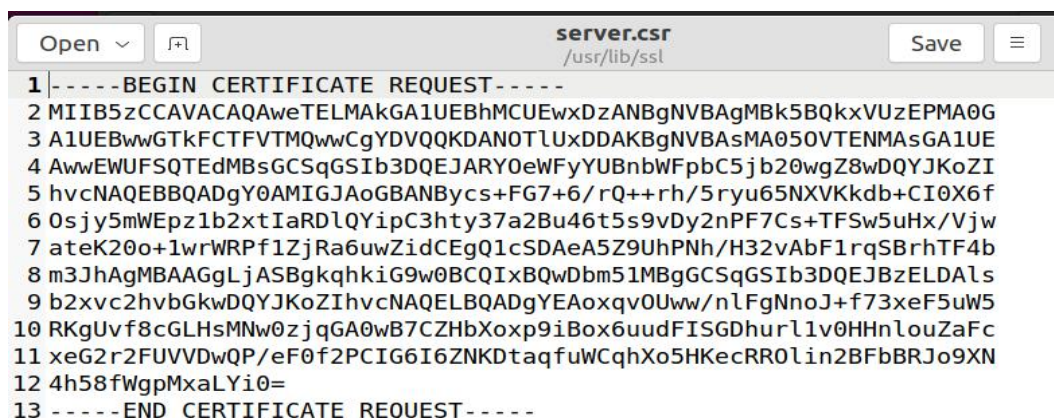
```
yara1@ubuntu:/usr/lib/ssl$ sudo openssl req -new -key server.key -out server.csr -config openssl.cnf
[sudo] password for yara1:
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is 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]:PS
State or Province Name (full name) [Some-State]:Nablus
Locality Name (eg, city) []:Nablus
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NNU
Organizational Unit Name (eg, section) []:NNU
Common Name (e.g. server FQDN or YOUR name) []:YARA
Email Address []:YARA@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:yara12345
An optional company name []:company
yara1@ubuntu:/usr/lib/ssl$ sudo gedit server.csr

** (gedit:114792): WARNING **: 09:55:17.885: Set document metadata failed: Setting attribute metadata::gedit-position
not supported
yara1@ubuntu:/usr/lib/ssl$
```

Figure2.2 shows the result of previous command.

The content of server.csr file:



```
1 -----BEGIN CERTIFICATE REQUEST-----
2 MIIB5zCCAACAQAweTElMAkGA1UEBhMCUExDzANBgNVBAgMBk5BQkxVUzEPMA0G
3 A1UEBwwGTkFCTFVtMQwwCgYDVQQKDANOTLUXDDAKBgNVBAsMA050VTENMAsgA1UE
4 AwWEWUFSQTEDMBsGCSqGSIb3DQEJARY0eWFyYUBnbWFPbC5jb20wgZ8wDQYJKoZI
5 hvCNAQEBAQADgY0AMIGJAoGBANBycs+FG7+6/rQ++rh/5ryu65NXVKkdb+CI0X6f
6 0sJy5mWEpz1b2xtIaRDLQYipC3hty37a2Bu46t5s9vDy2nPF7Cs+TFSw5uHx/Vjw
7 ateK20o+1wrWRPflZjRa6uwZidCEgQ1cSDAeA5Z9UHPNh/H32vAbF1rqSBrhTF4b
8 m3JhAgMBAAGgljASBgkqhkiG9w0BCQIxMQwDbm51MBGCSqGSIb3DQEJJBZELDAlS
9 b2xvc2hvbGkwDQYJKoZIhvcNAQELBQADgYEAoxqv0Uww/nlFgNnoJ+f73xeF5uW5
10 RKgUvf8cGLHsMNw0zjqGA0wB7CZHbXoxp9iBox6uudFISGDhurl1v0HHnlouZaFc
11 xeG2r2FUVVDwQP/eF0f2PCIG6I6ZNKDtaqfuWCqhXo5HKecRR0lin2BFbBRJo9XN
12 4h58fwgPMxALYi0=
13 -----END CERTIFICATE REQUEST-----
```

Figure2.2 shows the content of file server.csr.

II.iii. Generating Certificates:

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

In this step we need two files, private key file (ca.key) and certificate request file. Now, change the following line in openssl.cnf file: **policy=policy-match to policy-anything**. The purpose of this modification is to allow to all corresponding fields that match certain fields in CA's certificate in the client certificate to be signed.

To generate a certificate of the client using this command:
openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf

Then show the content in to files index.txt and serial file.

The content of index.txt file:

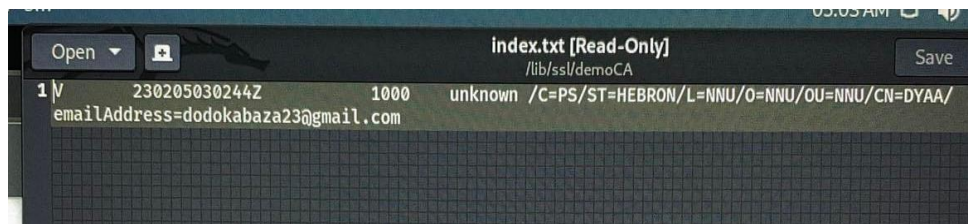
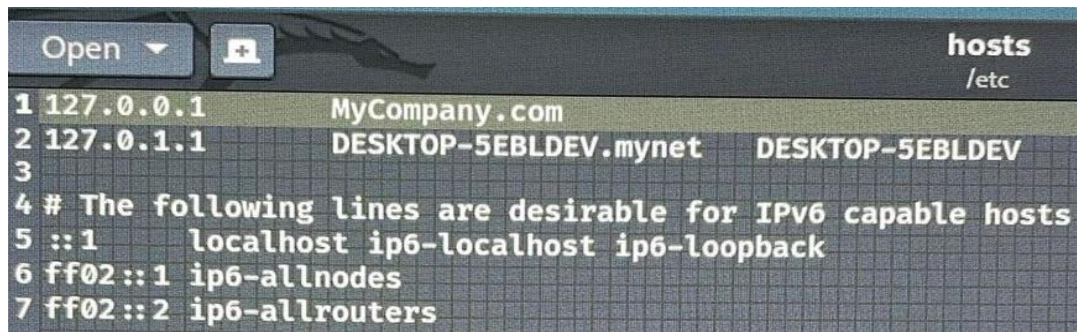


Figure2.3.1 The content of index file.

The index.txt file contains some information that will be incorporated into the certificate request. But in serial it still the same because it does not change in any part in certificate generation.

III. Public Key Infrastructure (PKI) for Domains:

a. An entry for the domain MyCompany.com was needed in our Linux machine, we needed to map an IP for this domain, the IP that mapped was 127.0.0.1 and this was done by directing to the hosts file under the, etc file and modifying it so when a request reaches 127.0.0.1 it opens MyComany.com.



```
Open [icon] hosts /etc
1 127.0.0.1 MyCompany.com
2 127.0.1.1 DESKTOP-5EBLDEV.mynet DESKTOP-5EBLDEV
3
4 # The following lines are desirable for IPv6 capable hosts
5 ::1 localhost ip6-localhost ip6-loopback
6 ff02::1 ip6-allnodes
7 ff02::2 ip6-allrouters
```

Figure3.a shows the ip address of Mycompany.com

b. The secret key and certificate was combined into one file name server.pem using the following commands after directing to the files path :

```
cp server.key server.pem
cat server.crt >> server.pem
```

The first command copies the content of server.key and pastes it into server.pem, the second command takes the content of server.crt and appends it to the server.pem file, cp doesn't work here,because cp overwrites the file, whereas cat appends to the file. The reason of combining these 2 files is to create a certification for the website which consists of a secret key and a certificate file.

c. The web server started using the following command:

```
Openssl s_server -cert server.pem -www
```

A passphrase was required several times for security purposes.

d. The default used port number is 4433. Lunch the Firefox and use the following URL:

<https://mycompany.com:4433/>

It will look like this because the certificate does not import yet.

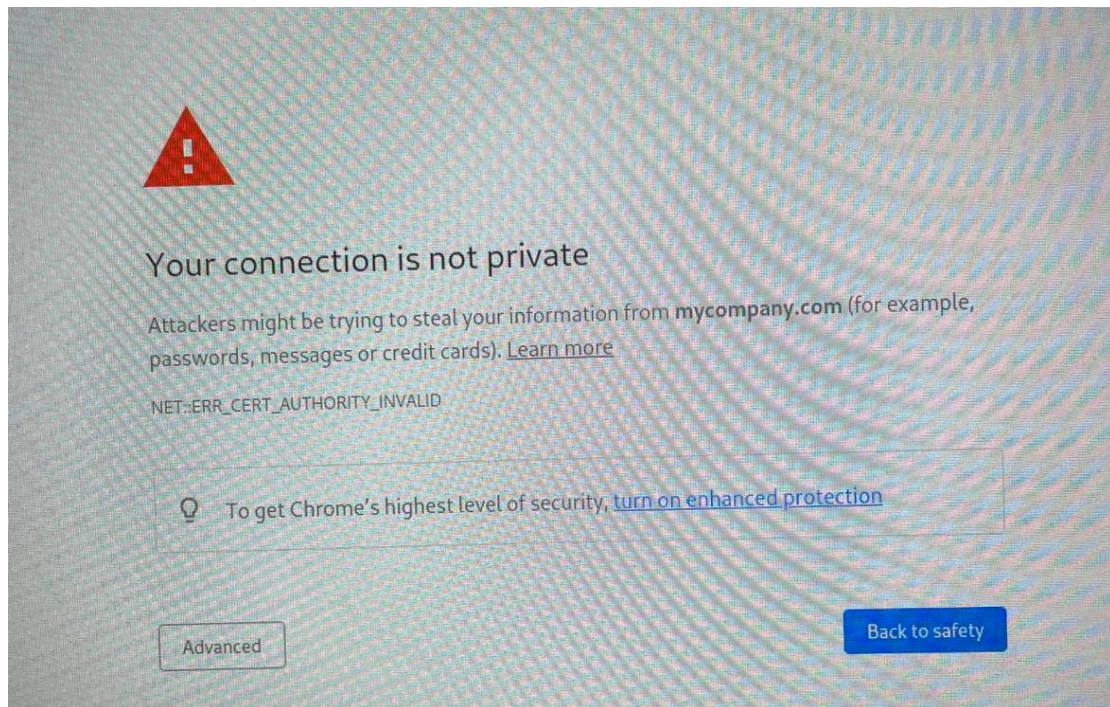


Figure3.d shows when you went to connect before importing a certificate.

e. Usually, a CA requests browsing companies (such as Mozilla for Firefox) to include its own certificates. However, here you will have to manually load your certificate (ca.cert) file into Firefox. By following this steps:

-First, in the browser address bar, chrome://settings/.

-Then, Navigate to bottom of the page and click Advanced and choose Privacy and Security, click Security then click on Manage certificates and a certificate dialog will open, click the Trusted Root Certification Authority tab, finally, choose Import.

-Now, the certificate import Wizard appears. Click Next. Browse to the certificate. Make sure you show All Files to find the certificate. After finding the certificate, click Next (click Next until the import is complete).

-Finally, in the certificate, confirm the certificate information.

f. Now use the URL https://mycompany.com:443/ again in your Firefox. It should open because the certificate was imploded.

g. Modify a single byte of the file server.pem and restart the server. Reload the URL.

When modifying the bit the page will not open again as happened in figure3.d, because changing one bit in the certificate file will change the whole file and it affected the authority of the file.

h. Now, restore the original server.pem file and restart the server again. Reload the URL.

When restoring the original file it will work because the certificate that use in the first is back.

IV. Digital Signatures Creation:

The purpose of using digital signatures is to ensure the integrity of messages and to authenticate the sender.

Now, to create a digital signature we will start with creating the private key by using this command:

openssl genrsa -out key1.private 1024

Then, generate an RSA public key using the following command:

openssl rsa -in key1.private -out key1.public -pubout -outform PEM

After generating the two keys, create a file named signfile.txt and fill it with arbitrary text. Then generate a hash code using SHA256 in a file named signfile.sha and save it into signfile.signature by using this command:

A terminal window showing the execution of two commands. The first command is 'sudo openssl dgst -sha256 signfile.txt > signfile.sha' and the second is 'sudo cat signfile.sha'. The output of the second command is 'SHA256(signfile.txt)= 49f88944022bafac579c97fa1cf7dc9d3206091536928374870957db35777a31'.

```
yara1@ubuntu: /usr/lib/ssl$ sudo openssl dgst -sha256 signfile.txt > signfile.sha
yara1@ubuntu: /usr/lib/ssl$ sudo cat signfile.sha
SHA256(signfile.txt)= 49f88944022bafac579c97fa1cf7dc9d3206091536928374870957db35777a31
yara1@ubuntu: /usr/lib/ssl$
```

Figure4.a shows the hash code and the command that are used.

For more security, we will encrypt the hash code using RSA private key using the following command:

openssl pkeyutl -encrypt -inkey key1.private -in signfile.sha > signfile.signature

To ensure if the encrypted hash code was change or not or your public key is correct or not, we will decrypt the encrypted hash code (signfile.signature):

**openssl pkeyutl -decrypt -pubin key1.public -in
signfile.signature > signfile1.txt**

Finally, we will use the hash code to confirm the integrity of the file signfile.txt by using this command:

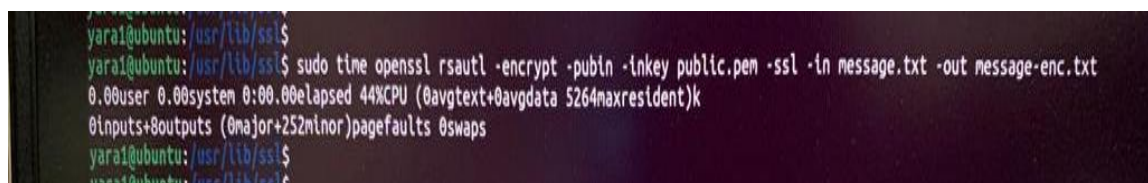
V. To Do :

Finally, in this step, we just will encrypt a file in two ways one using symmetric encryption and the second one using public-key encryption, and compare between the time it takes.

- a. In the first create a file named message.txt that contains a 16-byte message.
- b. Then, prepare a 1024-bit RSA public/private key pair by using this command:

**openssl genrsa -out privateKey.pem 1024
openssl rsa -in privateKey.pem -out public.pem -pubout -
outform PEM**

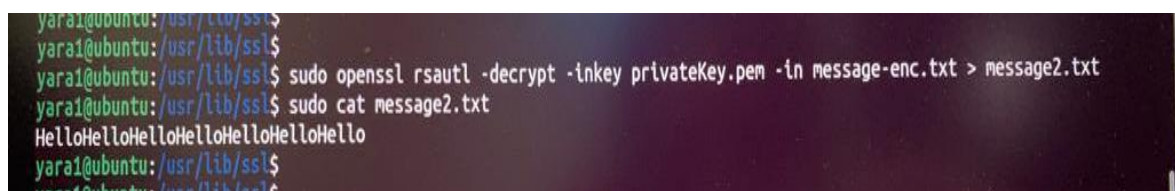
- c. Encrypt the message using the public key and save the output in message-enc.txt using this command:



```
yara1@ubuntu: /usr/lib/ssl$  
yara1@ubuntu: /usr/lib/ssl$ sudo time openssl rsautl -encrypt -pubin -inkey public.pem -ssl -in message.txt -out message-enc.txt  
0.00user 0.00system 0:00.00elapsed 44%CPU (0avgtext+0avgdata 5264maxresident)k  
0inputs+8outputs (0major+252minor)pagefaults 0swaps  
yara1@ubuntu: /usr/lib/ssl$  
yara1@ubuntu: /usr/lib/ssl$
```

Figure5.c shows the command and the time that are used.

- d. Decrypt the message using the private key using this command:



```
yara1@ubuntu: /usr/lib/ssl$  
yara1@ubuntu: /usr/lib/ssl$  
yara1@ubuntu: /usr/lib/ssl$ sudo openssl rsautl -decrypt -inkey privateKey.pem -in message-enc.txt > message2.txt  
yara1@ubuntu: /usr/lib/ssl$ sudo cat message2.txt  
HelloHelloHelloHelloHelloHello  
yara1@ubuntu: /usr/lib/ssl$  
yara1@ubuntu: /usr/lib/ssl$
```

Figure5.d shows the command for decryption.

- e. Encrypt the message using 128.bit AES key using this command:


```
yara1@ubuntu:/usr/lib/ssl$  
yara1@ubuntu:/usr/lib/ssl$ sudo time openssl enc -aes128 -in message.txt -out enc-message.txt  
enter aes-128-cbc encryption password:  
Verifying - enter aes-128-cbc encryption password:  
*** WARNING : deprecated key derivation used.  
Using -iter or -pbkdf2 would be better.  
0.00user 0.00system 0:07.26elapsed 0%CPU (0avgtext+0avgdata 5160maxresident)k  
0inputs+8outputs (0major+255minor)pagefaults 0swaps  
yara1@ubuntu:/usr/lib/ssl$
```

Figure 5.e shows the command for encryption using AES.

The time taken in public-key encryption is smaller than that taken in AES-128, and using public-key encryption is more secure than using AES-128 encryption because public-key used different keys in encryption and decryption operation but in AES-128 encryption use the same key in two operations.

Conclusion

In the end, we use openssl to learn about public-key encryption and its examples, how much bit can be affected on certificate, and using public-key encryption is better than symmetric encryption.