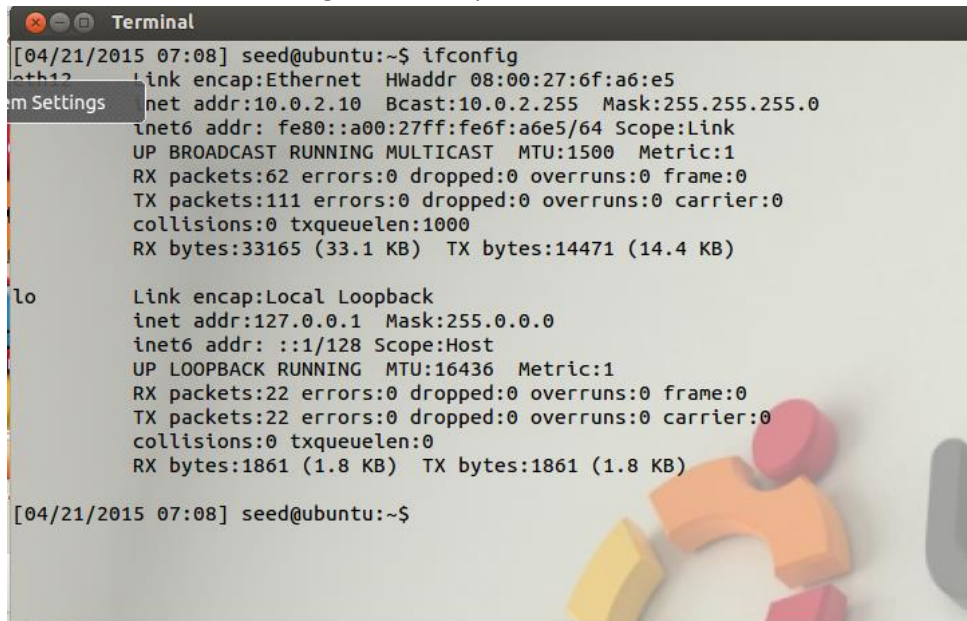


# Crypto Lab III – Public-Key Cryptography and PKI

-Bharath Darapu

## 2 Lab Environment

For this lab we will be using one VM to perform the tasks.



```
Terminal
[04/21/2015 07:08] seed@ubuntu:~$ ifconfig
eth1:  Link encap:Ethernet  HWaddr 08:00:27:6f:a6:e5
       inet addr:10.0.2.10  Bcast:10.0.2.255  Mask:255.255.0
       inet6 addr: fe80::a00:27ff:fe6f:a6e5/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:62 errors:0 dropped:0 overruns:0 frame:0
       TX packets:111 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:33165 (33.1 KB)  TX bytes:14471 (14.4 KB)

lo:    Link encap:Local Loopback
       inet addr:127.0.0.1  Mask:255.0.0.0
       inet6 addr: ::1/128 Scope:Host
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
       RX packets:22 errors:0 dropped:0 overruns:0 frame:0
       TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:1861 (1.8 KB)  TX bytes:1861 (1.8 KB)

[04/21/2015 07:08] seed@ubuntu:~$
```

VM\_1 (10.0.2.10)

### Installing OpenSSL:

OpenSSL is already downloaded onto our pre-built VMs. We need to configure it using the following commands:

```
$ cd /seed/openssl-1.0.1
```

```
$ ./config
```

```
$ make
```

```
$ make test
```

```
$ sudo make install
```

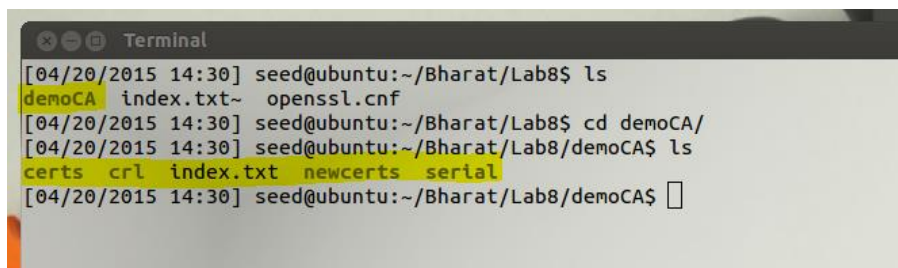
If we have /usr/local/ssl then our configuration ran successfully.

## 3 Lab Tasks

### 3.1 Task 1: Become a Certificate Authority (CA)

#### The Configuration File openssl.cnf:

In order to use OpenSSL to create certificates, we need a configuration file. We will copy the /etc/ssl/openssl.cnf to /Bharat/Lab8/openssl.cnf. Next we create the required subdirectories as per the CA\_Default section. After creating the sub directories our structure will look like:



```
Terminal
[04/20/2015 14:30] seed@ubuntu:~/Bharat/Lab8$ ls
demoCA  index.txt~  openssl.cnf
[04/20/2015 14:30] seed@ubuntu:~/Bharat/Lab8$ cd demoCA/
[04/20/2015 14:30] seed@ubuntu:~/Bharat/Lab8/demoCA$ ls
certs  crl  index.txt  newcerts  serial
[04/20/2015 14:30] seed@ubuntu:~/Bharat/Lab8/demoCA$
```

sub directories created as per CA\_Default section

Directories: certs,crl and newcerts.

Files: index.txt and serial

## Certificate Authority (CA):

Next we need to create a self-signed certificate for the CA. Which we can generate by using the following command:

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

It will ask for a few details and can be seen as follows:

```
Terminal
[04/20/2015 14:35] seed@ubuntu:~/Bharat/Lab8$ openssl req -new -x509 -keyout ca
.key -out ca.crt -config openssl.cnf
Generating a 1024 bit RSA private key
.....+++++
.....+++++
unable to write 'random state'
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]:US
State or Province Name (full name) [Some-State]:New York
Locality Name (eg, city) []:Syracuse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:PKILab
Organizational Unit Name (eg, section) []:Certificate Authority
Common Name (e.g. server FQDN or YOUR name) []:Lab8
Email Address []:bdarapu@syr.edu
[04/20/2015 14:37] seed@ubuntu:~/Bharat/Lab8$
```

generating root certificate.

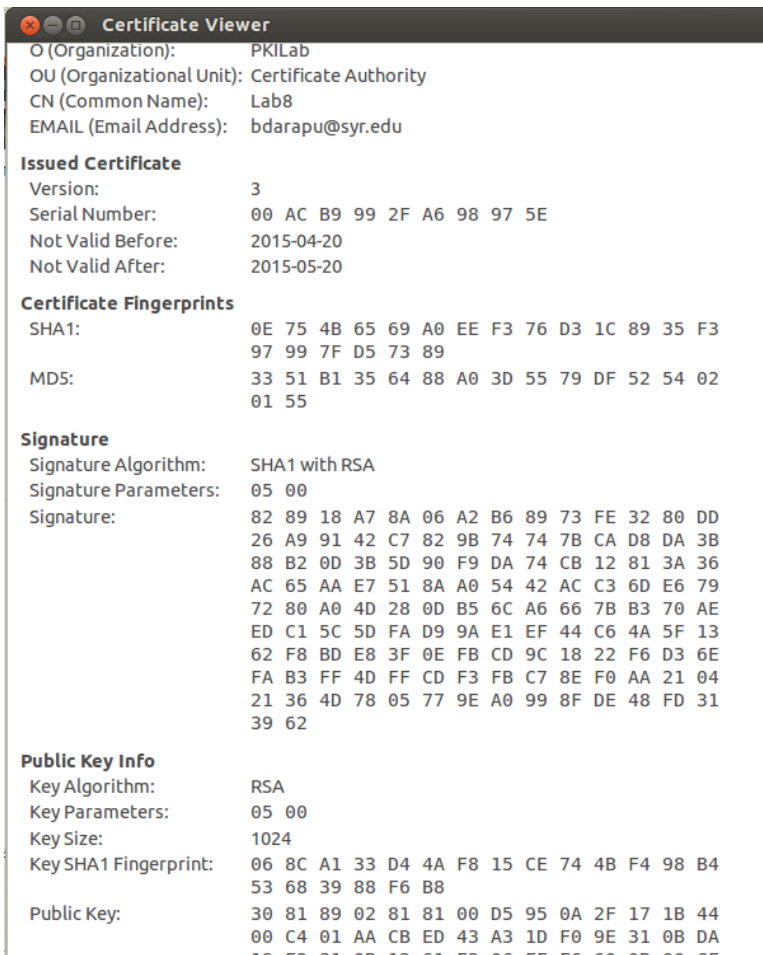
Here two new files are created with the following content:

```
Terminal
[04/20/2015 14:40] seed@ubuntu:~/Bharat/Lab8$ cat ca.crt
-----BEGIN CERTIFICATE-----
MIIC9jCCAl+gAwIBAgIJAKy5mS+mmJdeMA0GCSqGSIb3DQEBBQUAMIGTMQswCQYD
VQQGEwJVUzERMA8GA1UECAwITmV3IFlvcmsxETAPBgNVBACMFN5cmFjdXNlMQ8w
DQYDVQQKDAZQS0lMYWlxHjAcBgNVBASMFUNlcnRpZmljYXRlIEF1dGhvcml0eTEN
MA8GA1UEAwETGFI0DEEMBgGCSqGSIb3DQEJARYPYmRhcmFwdUBzeXIuZWRR1MB4X
DTE1MDQyMDIxMzcyNFoXDTE1MDUyMDIxMzcyNFowZmMxZmMxZmMxZmMxZmMxZmMx
DwYDVQQIDAh0ZXcgWW9yazERMA8GA1UEBwwIU3lyYWN1c2UxZDZANBgNVBAoMB1BL
SUXhYjEeMBwGA1UECwwVQ2VydGlnaWNhdGUGuGQXV0aG9yaXR5MQ0wCwYDVQQDDARM
YWI4MR4wHAYJKoZIhvcNAQkBFg9iZGFyYXN1c2UxZDZANBgNVBz8wDQYJKoZIhvcN
AQEBBQADgY0AMIGJAoGBANWVCi8XG0QAxAGqy+1Dox3wnjEL2hnygQsTYfIG//Zg
CwDP0HnzRd41Gt0fCwELF5ASHitexJqZE3ryF+eRrwtMH4RKYFY9dPUR30tp3Ere
0e0ixI20tXg5jTB1ibzNsVL+bDpeT/1W10gSGVtRyyMT7AeUxxXAoAB0u/TNbbmX
AgMBAAGjUDBOMB0GA1UdDgQWBBSdtWUC7gDzOXWlvRdHzv1k9PRI/jAfBgNVHSME
GDAWBSdtWUC7gDzOXWlvRdHzv1k9PRI/jAMBgNVHRMEBTADAQH/MA0GCSqGSIb3
DQEBBQUAA4GBAIBKJGKeKBqK2iXP+MoDdJqmRQseCm3R0e8rY2juIsg07XZD52nTL
EoE6NqxlqudRiQBQqzDbeZ5coCgTSgNtwymZnuzcK7twVxd+tma4e9ExkpfE2L4
veg/DvvNnBgi9tNu+rP/Tf/N8/vHjvCqIQQhNk14BXeeoJmP3kj9MTli
-----END CERTIFICATE-----
[04/20/2015 14:40] seed@ubuntu:~/Bharat/Lab8$
```

generated ca.crt

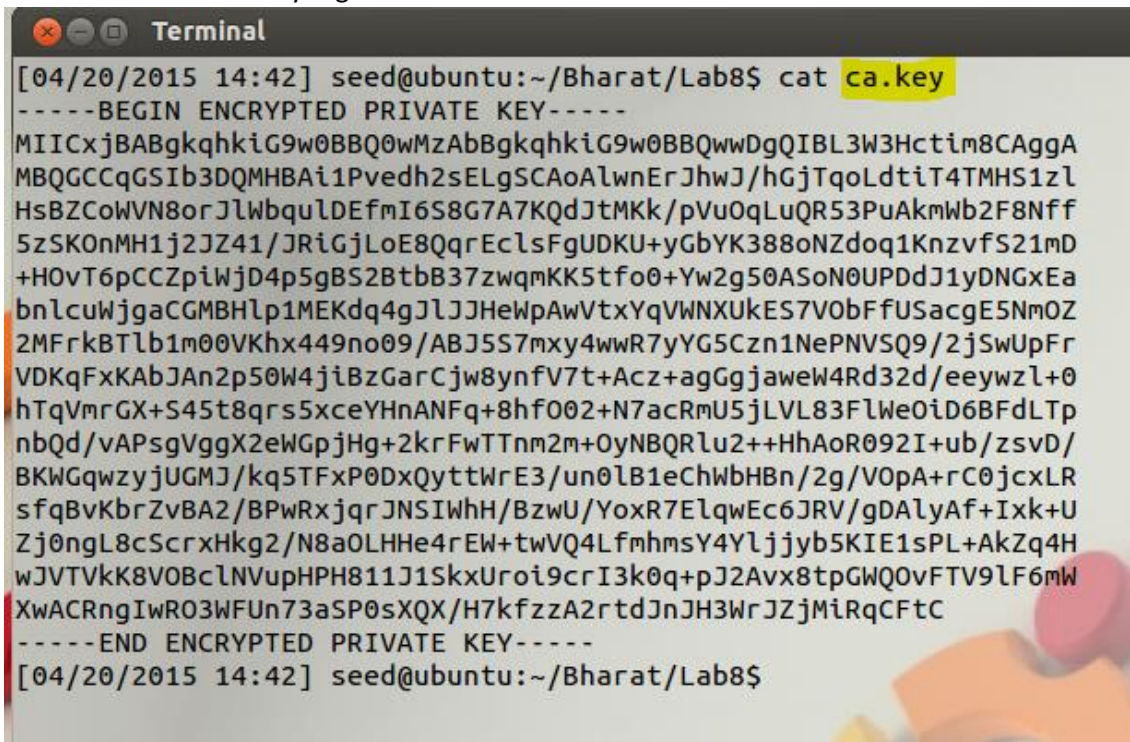
when we actually open the file with a certificate viewer we can see the following:





ca.crt opened with a certificate viewer

Another file called ca.key is generated with content as:



ca.key contents

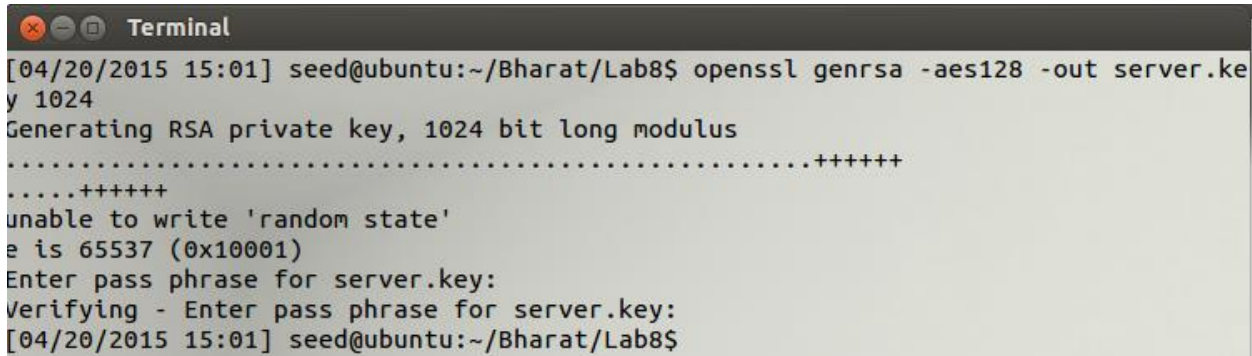
The file ca.key contains the CA's private key, while ca.crt contains the public-key among others.

### 3.2 Task 2: Create a Certificate for PKILabServer.com

#### Step 1: Generate public/private key pair:

we will use the following command:

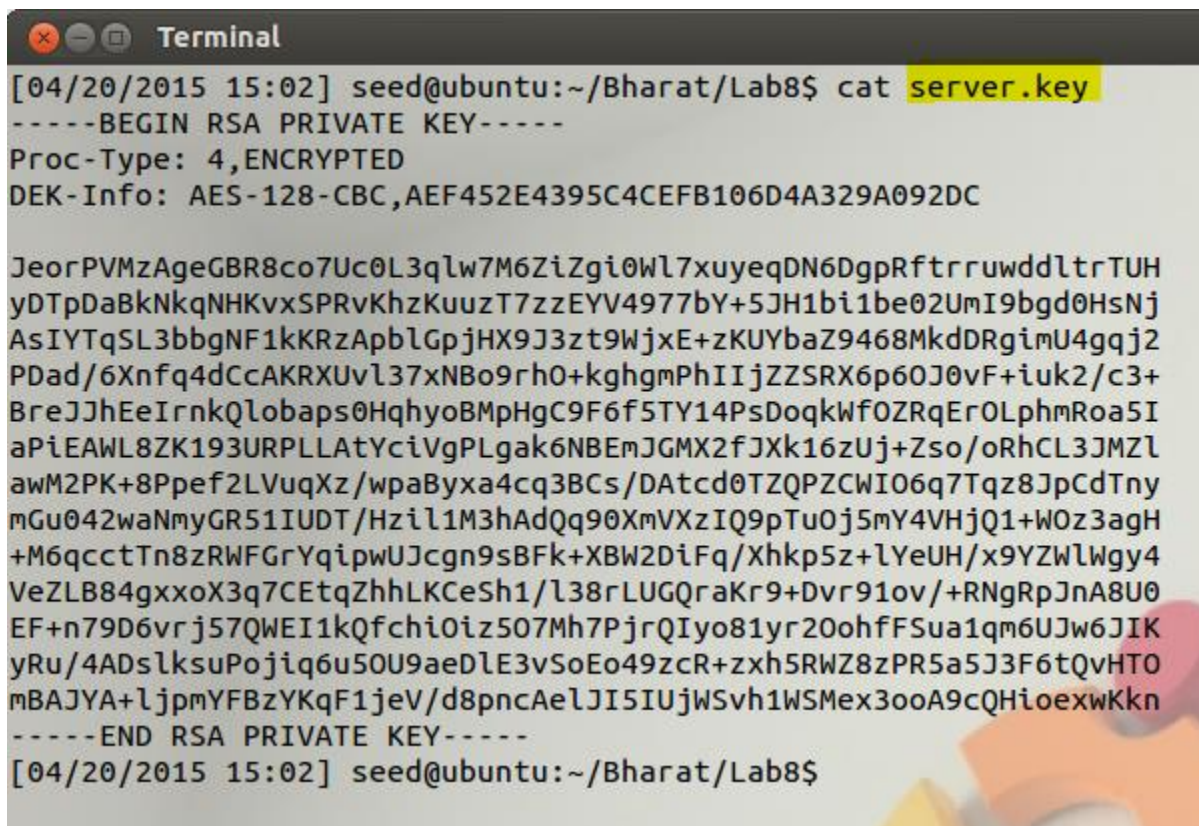
`'openssl genrsa -aes128 -out server.key 1024'`



```
Terminal
[04/20/2015 15:01] seed@ubuntu:~/Bharat/Lab8$ openssl genrsa -aes128 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
unable to write 'random state'
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[04/20/2015 15:01] seed@ubuntu:~/Bharat/Lab8$
```

*Generate public/private key pair*

The key is written to a server.key file.



```
Terminal
[04/20/2015 15:02] seed@ubuntu:~/Bharat/Lab8$ cat server.key
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,AEF452E4395C4CEFB106D4A329A092DC

JeorPVMzAgeGBR8co7Uc0L3qlw7M6ZiZgi0Wl7xuyeqDN6DgpRftrruwddltrTUH
yDTpDaBkNkqNHKvxSPRvKhZKuuzT7zzEYV4977bY+5JH1bi1be02UmI9bgd0HsNj
AsIYTqSL3bbgNF1kKRzApblGpjHX9J3zt9WjxE+zKUYbaZ9468MkdDRgimU4gqj2
PDad/6Xnfq4dCcAKRXUv137xNB09rh0+kghgmPhIIjZZSRX6p60J0vF+iuk2/c3+
BreJJhEeIrnkQlobaps0HqhyoBMpHgC9F6f5TY14PsDoqkWf0ZRqErOLphmRoa5I
aPiEAWL8ZK193URPLLatYciVgPLgak6NBEmJGMX2fJXk16zUj+Zso/orhCL3JMZL
awM2PK+8PpEf2LVuqXz/wpaByxa4cq3BCs/DAtcd0TZQPZCWI06q7Tqz8JpCdTny
mGu042waNmyGR51IUDT/Hzil1M3hAdQq90XmVXzIQ9pTu0j5mY4VHjQ1+W0z3agH
+M6qcctTn8zRWFGrYqipwUJcgn9sBFk+XBW2DiFq/Xhkp5z+lYeUH/x9YZWlWgy4
VeZLB84gxxoX3q7CEtqZhhLKCeSh1/l38rLUGQraKr9+Dvr91ov/+RNgRpJnA8U0
EF+n79D6vrj57QWEI1kQfchi0iz507Mh7PjrQIyo81yr20ohfFSua1qm6UJw6JIK
yRu/4ADslksuPojiq6u50U9aedlE3vSoEo49zcR+zxh5RWZ8zPR5a5J3F6tQvHTO
mBAJYA+lJpmYFBzYKqF1jeV/d8pncAelJI5IUjWSvh1WSMex30oA9cQHioexwKkn
-----END RSA PRIVATE KEY-----
[04/20/2015 15:02] seed@ubuntu:~/Bharat/Lab8$
```

*server.key file with the private key*

As we can see, this file is encoded. We can use the following command to see the contents:

`'openssl rsa -in server.key -text'`.

A pass-phrase will be asked to open this file. This is the one which we used during generation of the key-pair.



```

Terminal
[04/20/2015 15:03] seed@ubuntu:~/Bharat/Lab8$ openssl rsa -in server.key -text
Enter pass phrase for server.key:
Private-Key: (1024 bit)
modulus:
 00:b7:77:6b:56:81:01:c9:c5:cf:be:23:b4:e7:de:
 ad:65:9f:02:84:29:ff:f6:6b:69:94:d8:37:9b:3a:
 b5:7d:d3:a2:37:74:12:19:62:ef:05:62:8c:f3:44:
 f0:f0:1d:e7:e2:17:46:c5:8a:c7:51:0d:70:c3:54:
 20:96:56:b1:b0:53:3b:9d:07:91:a8:46:84:7c:39:
 e2:f4:6c:d2:ef:fb:0a:08:02:cb:48:db:06:23:d5:
 db:ec:58:64:7f:1b:a4:3a:72:bb:0c:85:31:95:ee:
 a5:df:19:a0:a8:ff:55:12:e4:bb:31:65:dd:6b:e7:
 59:10:7f:55:2d:f3:b8:94:21
publicExponent: 65537 (0x10001)
privateExponent:
 24:fc:72:2b:32:3e:c6:0a:96:e2:e2:17:ea:56:bd:
 e6:2a:b4:3b:28:ac:6d:65:22:17:db:ae:fa:20:4f:
 ce:5a:07:ed:dd:fc:78:19:b5:ce:04:1c:1e:3e:db:
 c3:cc:83:9e:d4:ca:73:b0:92:96:08:7b:9f:25:2a:
 9c:a4:45:95:cb:10:33:52:38:33:09:2e:e3:86:b7:
 78:d1:d4:61:fe:5f:b8:47:ce:2a:11:3b:1c:b8:86:
 ab:07:83:35:d1:98:b4:21:46:a2:76:fb:ac:d3:62:
 85:77:8c:10:71:23:4a:02:08:65:02:3e:84:9d:f7:
 b3:9e:54:71:6a:1c:08:71

```

*Server.key contents*

## Step 2: Generate a Certificate Signing Request (CSR):

Now that we have the key file. We can generate a Certificate Signing Request (CSR). And we will be using the following command:

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

```

Terminal
[04/20/2015 15:07] seed@ubuntu:~/Bharat/Lab8$ 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]:US
State or Province Name (full name) [Some-State]:New York
Locality Name (eg, city) []:Syracuse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Lab8 Demo
Organizational Unit Name (eg, section) []:Lab8
Common Name (e.g. server FQDN or YOUR name) []:PKILabServer.com
Email Address []:bdarapu@syr.edu

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:bharat
An optional company name []:bharat
[04/20/2015 15:08] seed@ubuntu:~/Bharat/Lab8$

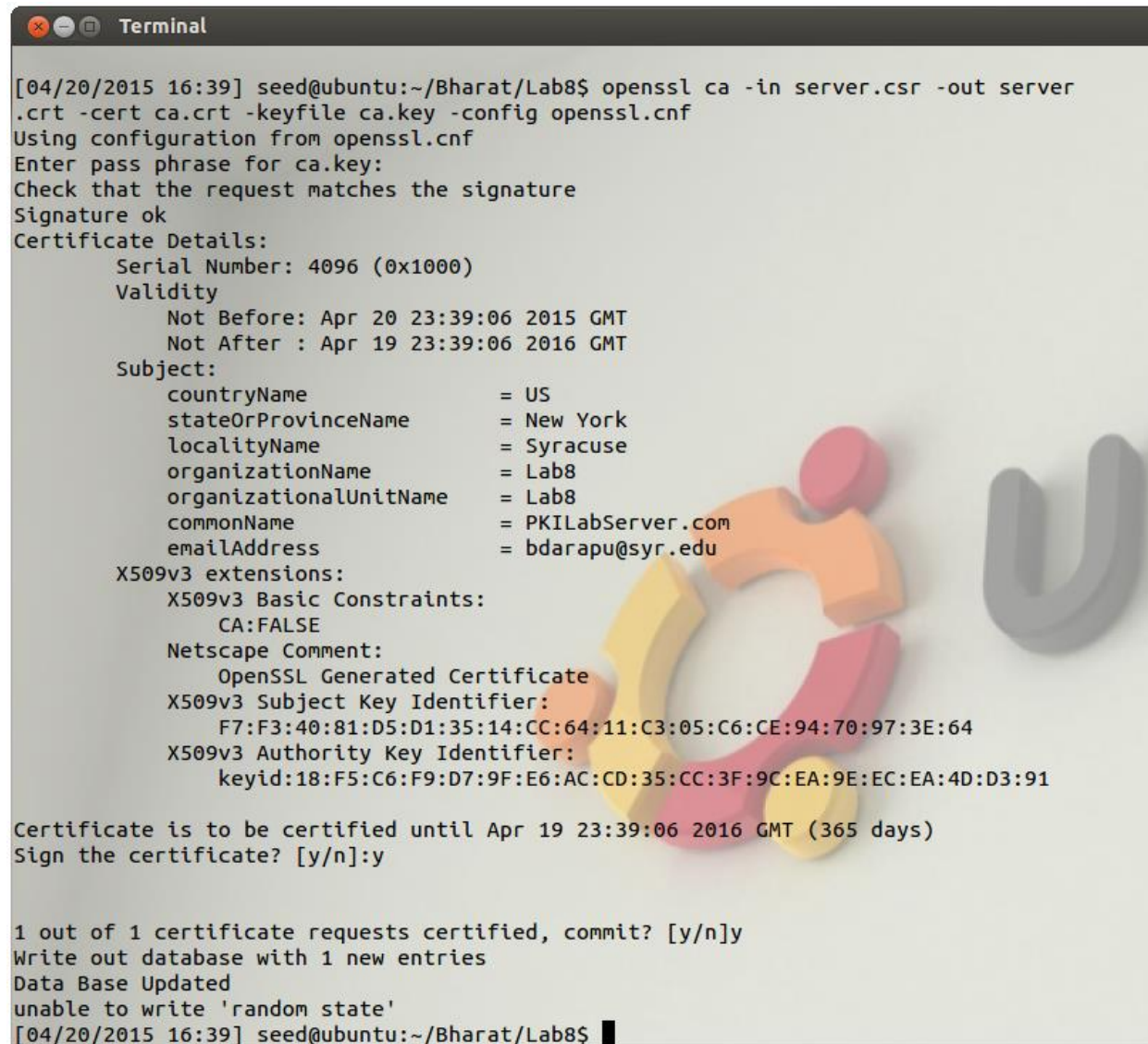
```

*CSR generated*

### Step 3: Generating Certificates:

We will be using the following command to turn the certificate signing request (server.csr) into an x509 certificate (server.crt):

```
'openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \ -config openssl.cnf'
```



```
[04/20/2015 16:39] seed@ubuntu:~/Bharat/Lab8$ 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: Apr 20 23:39:06 2015 GMT
    Not After : Apr 19 23:39:06 2016 GMT
  Subject:
    countryName           = US
    stateOrProvinceName   = New York
    localityName          = Syracuse
    organizationName      = Lab8
    organizationalUnitName = Lab8
    commonName            = PKILabServer.com
    emailAddress          = bdarapu@syr.edu
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
    Netscape Comment:
      OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
      F7:F3:40:81:D5:D1:35:14:CC:64:11:C3:05:C6:CE:94:70:97:3E:64
    X509v3 Authority Key Identifier:
      keyid:18:F5:C6:F9:D7:9F:E6:AC:CD:35:CC:3F:9C:EA:9E:EC:EA:4D:D3:91

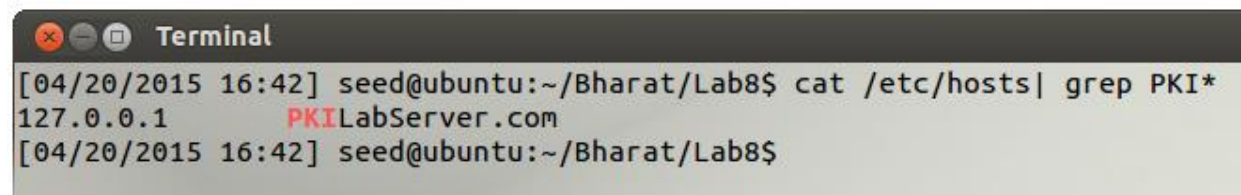
Certificate is to be certified until Apr 19 23:39:06 2016 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
unable to write 'random state'
[04/20/2015 16:39] seed@ubuntu:~/Bharat/Lab8$
```

*Server.crt, certificate generated*

### 3.3 Task 3: Use PKI for Web Sites

First we make PKILabServer.com as our domain name and map the domain name to local host by editing the /etc/hosts file.



```
[04/20/2015 16:42] seed@ubuntu:~/Bharat/Lab8$ cat /etc/hosts | grep PKI*
127.0.0.1      PKILabServer.com
[04/20/2015 16:42] seed@ubuntu:~/Bharat/Lab8$
```

*host added in VM*

Next we combine both the certificate and the key into a single file (to load later) using the following commands:

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

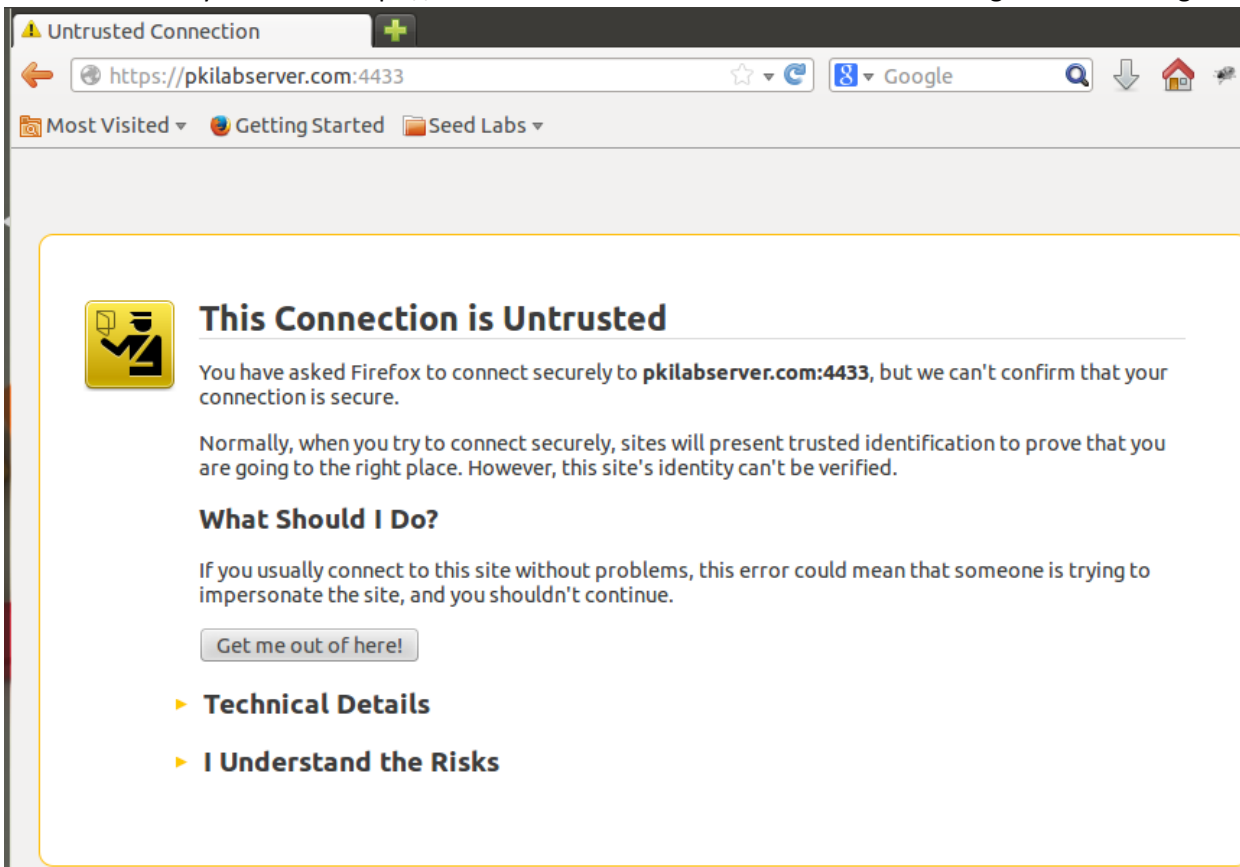
Further we launch a webserver with the certificate generated in the previous task using the following command:  
'`openssl s_server -cert server.pem -www`'

The result can be seen as follows:

```
Terminal
[04/20/2015 16:44] seed@ubuntu:~/Bharat/Lab8$ cp server.key server.pem
[04/20/2015 16:44] seed@ubuntu:~/Bharat/Lab8$ cat server.crt >> server.pem
[04/20/2015 16:45] seed@ubuntu:~/Bharat/Lab8$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
ACCEPT
ACCEPT
```

*PKILabServer.com webservice started*

Now when we try to access 'https://PKILabServer.com:4433' from the browser we get the following message:



*untrusted connection message in firefox.*

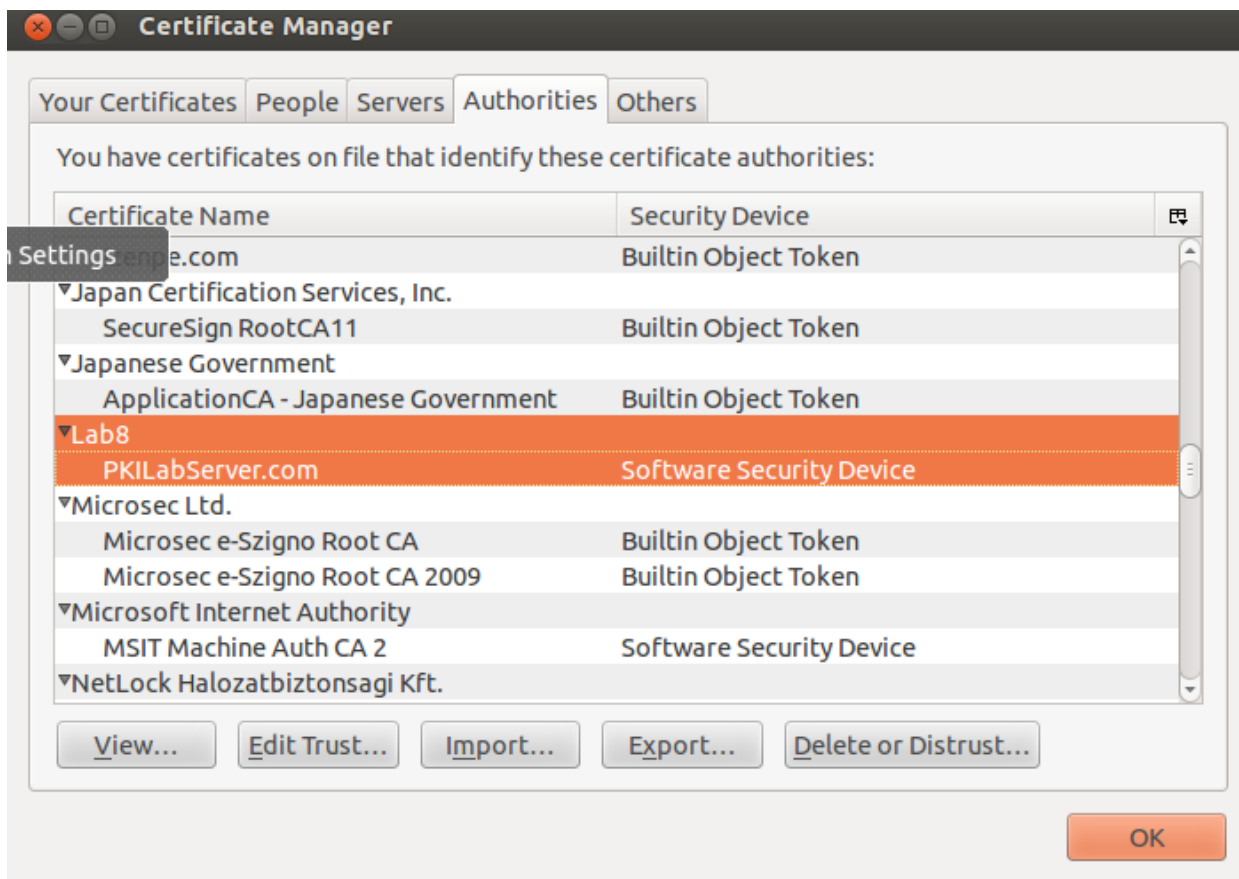
This happens since there is no valid certificate issued previously for that server.

Now we have to manually add our CA's certificate to firefox browser by going to:

*Edit -> Preference -> Advanced -> View Certificates*

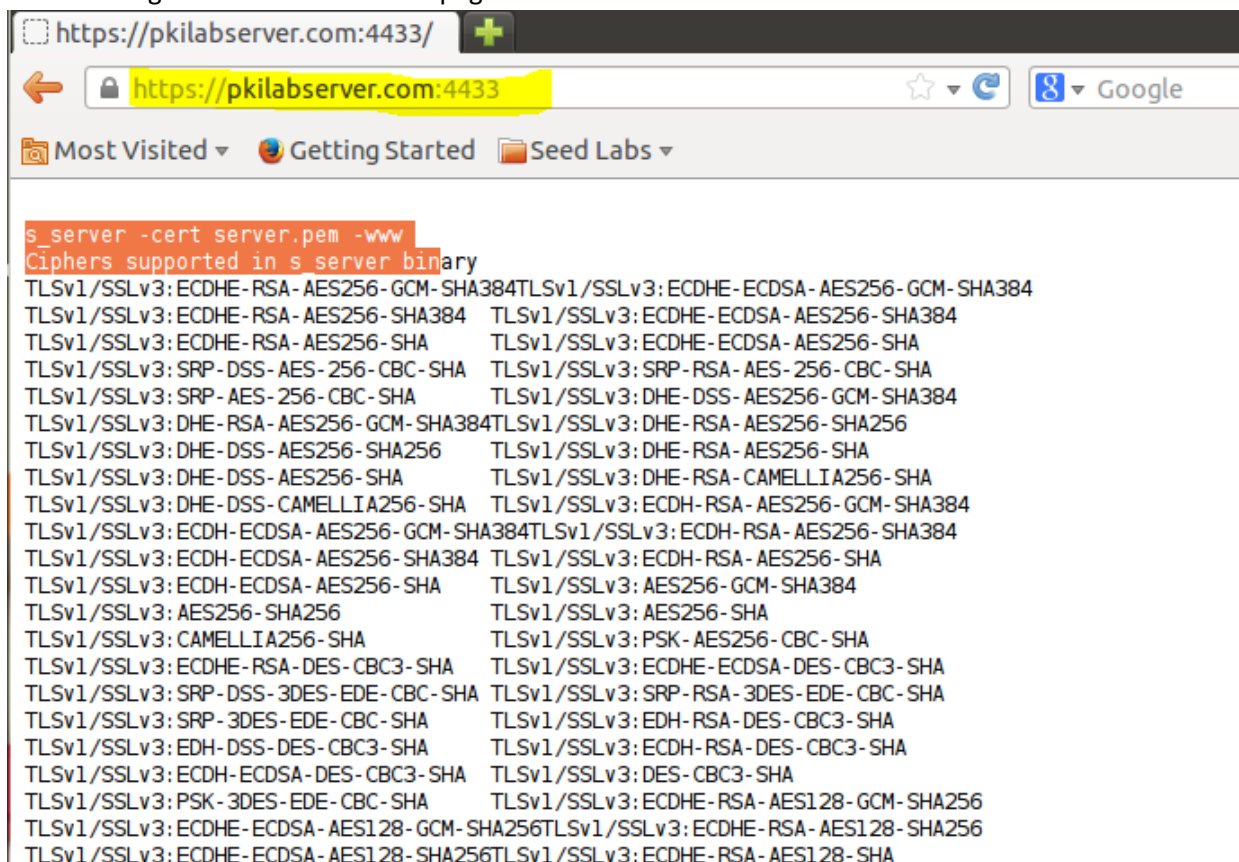
And upload the certificate which we created in the steps above.





certificate uploaded and added

After loading the certificate the webpage can be seen as follows:



webpage after certificate load

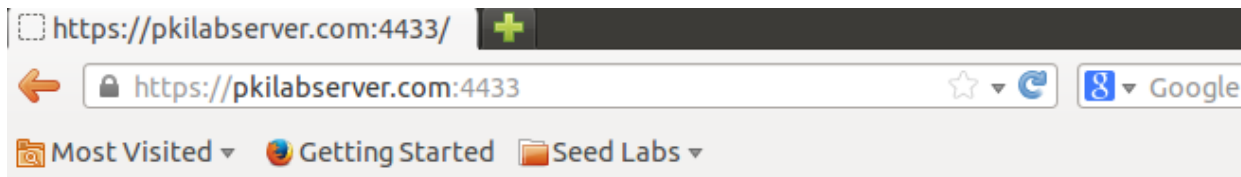


Next we modify one byte of server.pem and try to restart the server

```
Terminal
[04/20/2015 17:08] seed@ubuntu:~/Bharat/Lab8$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
```

editing server.pem file and restarting the server

Next when we try to reload the url.



```
s_server -cert server.pem -www
Ciphers supported in s_server binary
TLSv1/SSLv3: ECDHE-RSA-AES256-GCM-SHA384 TLSv1/SSLv3: ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1/SSLv3: ECDHE-RSA-AES256-SHA384 TLSv1/SSLv3: ECDHE-ECDSA-AES256-SHA384
TLSv1/SSLv3: ECDHE-RSA-AES256-SHA TLSv1/SSLv3: ECDHE-ECDSA-AES256-SHA
TLSv1/SSLv3: SRP-DSS-AES-256-CBC-SHA TLSv1/SSLv3: SRP-RSA-AES-256-CBC-SHA
TLSv1/SSLv3: SRP-AES-256-CBC-SHA TLSv1/SSLv3: DHE-DSS-AES256-GCM-SHA384
TLSv1/SSLv3: DHE-RSA-AES256-GCM-SHA384 TLSv1/SSLv3: DHE-RSA-AES256-SHA256
TLSv1/SSLv3: DHE-DSS-AES256-SHA256 TLSv1/SSLv3: DHE-RSA-AES256-SHA
TLSv1/SSLv3: DHE-DSS-AES256-SHA TLSv1/SSLv3: DHE-RSA-CAMELLIA256-SHA
TLSv1/SSLv3: DHE-DSS-CAMELLIA256-SHA TLSv1/SSLv3: ECDH-RSA-AES256-GCM-SHA384
TLSv1/SSLv3: ECDH-ECDSA-AES256-GCM-SHA384 TLSv1/SSLv3: ECDH-RSA-AES256-SHA384
TLSv1/SSLv3: ECDH-ECDSA-AES256-SHA384 TLSv1/SSLv3: ECDH-RSA-AES256-SHA
TLSv1/SSLv3: ECDH-ECDSA-AES256-SHA TLSv1/SSLv3: AES256-GCM-SHA384
TLSv1/SSLv3: AES256-SHA256 TLSv1/SSLv3: AES256-SHA
TLSv1/SSLv3: CAMELLIA256-SHA TLSv1/SSLv3: PSK-AES256-CBC-SHA
TLSv1/SSLv3: ECDHE-RSA-DES-CBC3-SHA TLSv1/SSLv3: ECDHE-ECDSA-DES-CBC3-SHA
TLSv1/SSLv3: SRP-DSS-3DES-EDE-CBC-SHA TLSv1/SSLv3: SRP-RSA-3DES-EDE-CBC-SHA
TLSv1/SSLv3: SRP-3DES-EDE-CBC-SHA TLSv1/SSLv3: EDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3: EDH-DSS-DES-CBC3-SHA TLSv1/SSLv3: ECDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3: ECDH-ECDSA-DES-CBC3-SHA TLSv1/SSLv3: DES-CBC3-SHA
TLSv1/SSLv3: PSK-3DES-EDE-CBC-SHA TLSv1/SSLv3: ECDHE-RSA-AES128-GCM-SHA256
TLSv1/SSLv3: ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1/SSLv3: ECDHE-RSA-AES128-SHA256
TLSv1/SSLv3: ECDHE-ECDSA-AES128-SHA256 TLSv1/SSLv3: ECDHE-RSA-AES128-SHA
TLSv1/SSLv3: ECDHE-ECDSA-AES128-SHA TLSv1/SSLv3: SRP-DSS-AES-128-CBC-SHA
TLSv1/SSLv3: SRP-RSA-AES-128-CBC-SHA TLSv1/SSLv3: SRP-AES-128-CBC-SHA
TLSv1/SSLv3: DHE-DSS-AES128-GCM-SHA256 TLSv1/SSLv3: DHE-RSA-AES128-GCM-SHA256
TLSv1/SSLv3: DHE-RSA-AES128-SHA256 TLSv1/SSLv3: DHE-DSS-AES128-SHA256
TLSv1/SSLv3: DHE-RSA-AES128-SHA TLSv1/SSLv3: DHE-DSS-AES128-SHA
TLSv1/SSLv3: DHE-RSA-SEED-SHA TLSv1/SSLv3: DHE-DSS-SEED-SHA
TLSv1/SSLv3: DHE-RSA-CAMELLIA128-SHA TLSv1/SSLv3: DHE-DSS-CAMELLIA128-SHA
TLSv1/SSLv3: ECDH-RSA-AES128-GCM-SHA256 TLSv1/SSLv3: ECDH-ECDSA-AES128-GCM-SHA256
TLSv1/SSLv3: ECDH-RSA-AES128-SHA256 TLSv1/SSLv3: ECDH-ECDSA-AES128-SHA256
TLSv1/SSLv3: ECDH-RSA-AES128-SHA TLSv1/SSLv3: ECDH-ECDSA-AES128-SHA
```

url loaded

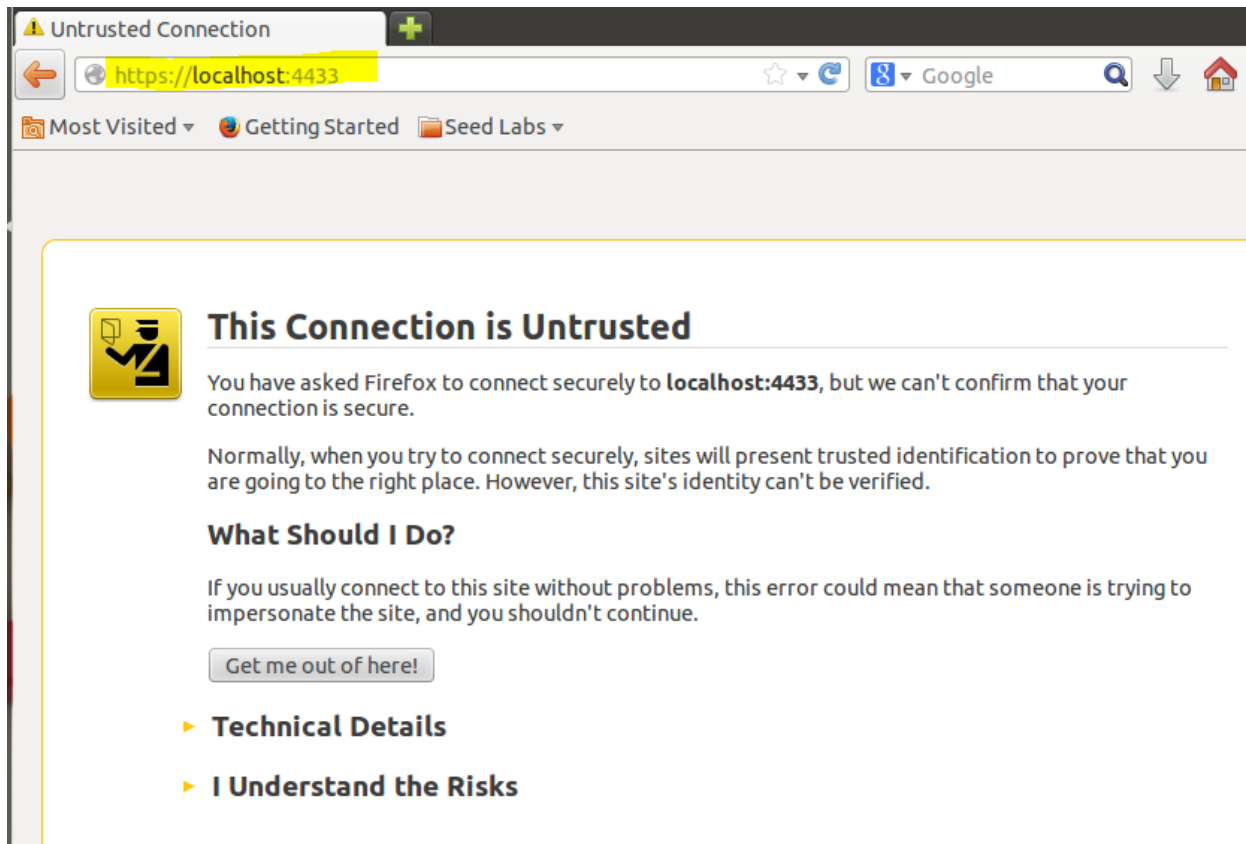
However when we change something in human readable format or the modulus or the key identifier nothing changes. The webpage loads as before.

When we change a byte in the last certificate section we get the following error:

```
Terminal
[04/20/2015 17:28] seed@ubuntu:~/Bharat/Lab8$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
unable to load certificate
3073882312:error:0D07209B:asn1 encoding routines:ASN1_get_object:too long:asn1_lib.c:142:
3073882312:error:0D068066:asn1 encoding routines:ASN1_CHECK_TLEN:bad object header:tasn_d
c.c:1324:
3073882312:error:0D07803A:asn1 encoding routines:ASN1_ITEM_EX_D2I:nested asn1 error:tasn_
ec.c:388:Type=X509
3073882312:error:0906700D:PEM routines:PEM_ASN1_read_bio:ASN1 lib:pem_oth.c:83:
[04/20/2015 17:28] seed@ubuntu:~/Bharat/Lab8$
```

*certificate section modified*

Next since PKILabServer.com is registered with 127.0.0.1 which is localhost, let see what happens if we hit <https://localhost:4433>.



*localhost hit*

As we can see, it is not taking the previous certificate as the previous certificate is valid only for PKILabServer.com.

### 3.4 Task 4: Establishing a TLS/SSL connection with server

We will be using the example files given in the lab description.

For this task we will be using the certificates which we generated in the previous section. With valid certificates and no code changes we can see that the communication between the server and the client is successful.



```
Terminal
[04/20/2015 20:36] seed@ubuntu:~/Bharat/Lab8/task4$ ./serv
Enter PEM pass phrase:
Connection from 100007f, port c886
SSL connection using AES256-GCM-SHA384
Client certificate:
  subject: /C=US/ST=New York/L=Syracuse/O=SEED Labs Inc./OU=PKI Lab/CN=PKILabClient.
com/emailAddress=admin@PKILabClient.com
  issuer: /C=US/ST=New York/L=Syracuse/O=SEED Labs Inc./OU=PKI Lab/CN=PKILabCA.com/e
mailAddress=admin@PKILabCA.com
Got 12 chars:'Hello World!'
[04/20/2015 20:37] seed@ubuntu:~/Bharat/Lab8/task4$
```

```
Terminal
[04/20/2015 20:37] seed@ubuntu:~/Bharat/Lab8/task4$ ./cli
Enter PEM pass phrase:
SSL connection using AES256-GCM-SHA384
Server certificate:
  subject: /C=US/ST=New York/L=Syracuse/O=SEED Labs Inc./OU=PKI Lab/CN=PK
ILabServer.com/emailAddress=admin@PKILabServer.com
  issuer: /C=US/ST=New York/L=Syracuse/O=SEED Labs Inc./OU=PKI Lab/CN=PKI
LabCA.com/emailAddress=admin@PKILabCA.com
Got 11 chars:'I hear you.'
[04/20/2015 20:37] seed@ubuntu:~/Bharat/Lab8/task4$
```

*transfer successful*

#### **Date Validation:**

Now let us change the date and see where the error comes:

Using the following command we stopped the time sync feature:

*'sudo service vboxadd-service stop'.*

Using the following command we changed the system date:

*'sudo date --set="1 May 2030"'*

We can notice the date change in the terminal before the \$ prompt. When we try to load the server we get the following error:

```
Terminal
[05/01/2030 00:00] seed@ubuntu:~/Bharat/Lab8/task4$ ./serv
Enter PEM pass phrase:
Connection from 100007f, port ca86
SSL connection using (NONE)
Client does not have certificate.
Got 0 chars:''
[05/01/2030 00:00] seed@ubuntu:~/Bharat/Lab8/task4$
```

```
Terminal
[05/01/2030 00:00] seed@ubuntu:~/Bharat/Lab8/task4$ ./cli
Enter PEM pass phrase:
3076351624:error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate v
erify failed:s3_clnt.c:1166:
[05/01/2030 00:00] seed@ubuntu:~/Bharat/Lab8/task4$
```

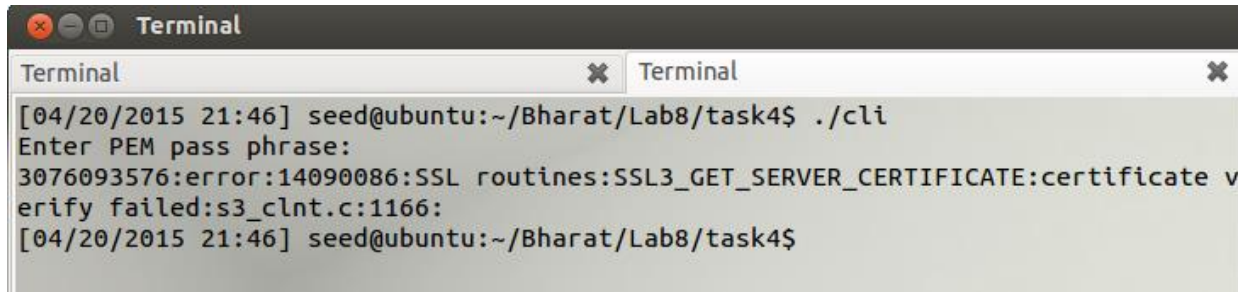
*certificate verify failed due to invalid certificate.*

Alternatively we can use the following lines of code to get and check the duration of the certificate:

```
ASN1_TIME *not_before = X509_get_notBefore(server_cert);
ASN1_TIME *not_after = X509_get_notAfter(server_cert);
```

### Whether the server certificate is signed by an authorized CA

we can do this by loading a certificate that was not signed by us in the previous task and check. This way we are not holding the private key for the certificate.



```
Terminal
[04/20/2015 21:46] seed@ubuntu:~/Bharat/Lab8/task4$ ./cli
Enter PEM pass phrase:
3076093576:error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate v
erify failed:s3_clnt.c:1166:
[04/20/2015 21:46] seed@ubuntu:~/Bharat/Lab8/task4$
```

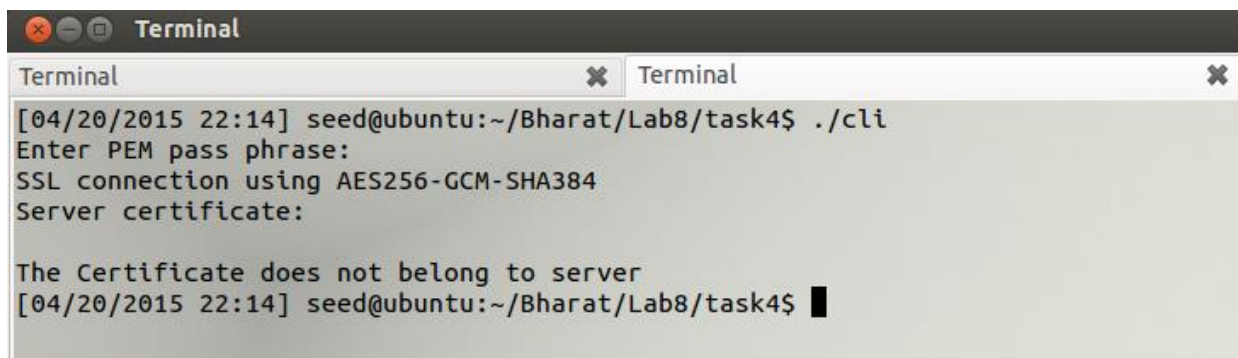
*Invalid ca.cert loaded*

### Whether the certificate belongs to the server

The following code can be used to check if the certificate belongs to the server or not. If not then the program will exit.

```
char serverName [100];
strcpy(serverName, "PKILabServer.com");
char nameFromCertificate [100];
X509_NAME_get_text_by_NID(X509_get_subject_name(server_cert), NID_commonName, nameFromCertificate, 100);
if(strcmp(serverName, nameFromCertificate) != 0) {
    printf("The Certificate does not belong to server");
    X509_free (server_cert);
    exit(0);
}
```

If we use a different server certificate than 'PKILabServer.com' then we get the following:



```
Terminal
[04/20/2015 22:14] seed@ubuntu:~/Bharat/Lab8/task4$ ./cli
Enter PEM pass phrase:
SSL connection using AES256-GCM-SHA384
Server certificate:

The Certificate does not belong to server
[04/20/2015 22:14] seed@ubuntu:~/Bharat/Lab8/task4$ █
```

*Certificate not from the expected server.*

### Whether the server is indeed the machine that the client wants to talk to

Now for this task we will have two servers:

1. Valid server with original certificate + valid private key.
2. Fraudulent server with original certificate + invalid private key (client\_key was used).

The valid server was able to connect without any trouble. However when we try to connect a fake server which has a valid certificate but not a valid private key we get the following error:



```
Terminal
[04/20/2015 23:01] seed@ubuntu:~/Bharat/Lab8/task4$ ./serv_fake
Enter PEM pass phrase:
3075856008:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:evp_enc.c:539:
3075856008:error:0906A065:PEM routines:PEM_do_header:bad decrypt:pem_lib.c:476:
3075856008:error:140B0009:SSL routines:SSL_CTX_use_PrivateKey_file:PEM lib:ssl_rsa.c:669:
[04/20/2015 23:02] seed@ubuntu:~/Bharat/Lab8/task4$
```

*fake\_server used to connect*

As we can see the server went for a decryption with the private key it had but failed. So we were unable to login.

If we do not need to verify the client certificate, this can be done by modifying the following line on serv.cpp:

```
Terminal
[05/01/2030 00:09] seed@ubuntu:~/Bharat/Lab8/task4$ cat serv.cpp | grep SSL_CTX_
set_verify
// SSL_CTX_set_verify(ctx,SSL_VERIFY_PEER,NULL); /* whether verify the certificate */
SSL_CTX_set_verify(ctx,SSL_VERIFY_NONE,NULL);
[05/01/2030 00:10] seed@ubuntu:~/Bharat/Lab8/task4$
```

*disable client certificate verification*

**What part of the code is responsible for the key exchange, i.e. for both sides to agree upon a secret key?**

After both the server and client have validated their respective certificates, a secure TCP tunnel is formed. And the data is transferred via that. The data transfer code can be seen as follows:

This happens after a valid TCP connection is established and SSL negotiation starts:

```
ssl = SSL_new (ctx);          CHK_NULL(ssl);
SSL_set_fd (ssl, sd);
err = SSL_connect (ssl);      CHK_SSL(err);
```

Once the certificates are verified the data transfer begins using the following code:

**Client side:**

```
err = SSL_write (ssl, "Hello World!", strlen("Hello World!")); CHK_SSL(err);
err = SSL_read (ssl, buf, sizeof(buf) - 1);      CHK_SSL(err);
```

The SSL\_write method is used to write the key to a buffer on the channel and the SSL\_read method is used to read the data from the channel.

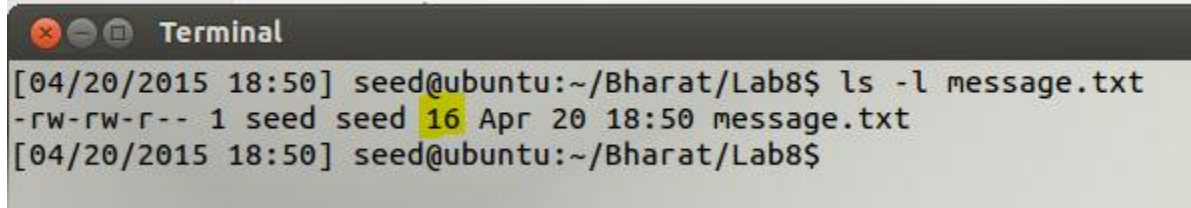
**Server side:**

```
err = SSL_read (ssl, buf, sizeof(buf) - 1);      CHK_SSL(err);
err = SSL_write (ssl, "I hear you.", strlen("I hear you.")); CHK_SSL(err);
```

The same applies to the server side as well. The SSL\_read method is used to read data from the channel and SSL\_write method is used to write data to it.

### 3.5 Task 5: Performance Comparison: RSA versus AES

For this task we will use a 16-byte file message.txt as follows:

A terminal window titled "Terminal" with a dark background. It shows a command prompt session where the user runs 'ls -l message.txt'. The output shows the file permissions as '-rw-rw-r--', the owner as 'seed', the group as 'seed', the size as '16', and the date and time as 'Apr 20 18:50'. The file name is 'message.txt'. The prompt is 'seed@ubuntu:~/Bharat/Lab8\$'.

16 byte message.txt file

We will be using the following code (shell script) to see how the execution happens:

```
#!/bin/bash
echo "Generate 1024-bit public/private key pair"
openssl genrsa -out privateKey.pem 1024
openssl rsa -in privateKey.pem -out publicKey.pem -outform PEM -pubout

echo "RSA Encryption"
LOOP=1
time while (( $LOOP <= 500 ))
do
openssl rsautl -encrypt -inkey publicKey.pem -pubin -in message.txt -out message_enc.txt
LOOP=$(( LOOP+1 ))
done

echo "RSA Decryption"
LOOP=1
time while (( $LOOP <= 500 ))
do
openssl rsautl -decrypt -inkey privateKey.pem -in message_enc.txt -out message_decry.txt
LOOP=$(( LOOP+1 ))
done

echo "AES Encryption"
LOOP=1
time while (( $LOOP <= 500 ))
do
openssl enc -aes-128-cbc -e -in message.txt -out message_enc.txt -K 00112233445566778889aabbccddeeff -iv
102030405060708
LOOP=$(( LOOP+1 ))
done

echo "AES Decryption"
LOOP=1
time while (( $LOOP <= 500 ))
do
openssl enc -aes-128-cbc -d -in message_enc.txt -out message.txt -K 00112233445566778889aabbccddeeff -iv 102030405060708
LOOP=$(( LOOP+1 ))
done
Done
```



The output can be seen as follows:

```
Terminal
[04/20/2015 19:31] seed@ubuntu:~/Bharat/Lab8$ ./RSAvsAES.sh
Generate 1024-bit public/private key pair
Generating RSA private key, 1024 bit long modulus
..++++++
.+++++
unable to write 'random state'
e is 65537 (0x10001)
writing RSA key

real    0m3.932s
user    0m0.424s
sys     0m0.284s
RSA Encryption:

real    0m5.426s
user    0m2.664s
sys     0m1.204s
AES Encryption

real    0m4.020s
user    0m0.648s
sys     0m0.408s
AES Decryption

real    0m3.709s
user    0m0.336s
sys     0m0.256s
[04/20/2015 19:31] seed@ubuntu:~/Bharat/Lab8$
```

Performance comparison using custom script.

Now lets run speed benchmark provided by openssl and see the performance:

```
Terminal
[04/20/2015 19:18] seed@ubuntu:~/Bharat/Lab8$ openssl speed rsa
Doing 512 bit private rsa's for 10s: 59024 512 bit private RSA's in 9.88s
Doing 512 bit public rsa's for 10s: 657404 512 bit public RSA's in 9.94s
Doing 1024 bit private rsa's for 10s: 10893 1024 bit private RSA's in 9.92s
Doing 1024 bit public rsa's for 10s: 204553 1024 bit public RSA's in 9.93s
Doing 2048 bit private rsa's for 10s: 1650 2048 bit private RSA's in 9.91s
Doing 2048 bit public rsa's for 10s: 54133 2048 bit public RSA's in 9.93s
Doing 4096 bit private rsa's for 10s: 220 4096 bit private RSA's in 9.93s
Doing 4096 bit public rsa's for 10s: 13156 4096 bit public RSA's in 9.93s
OpenSSL 1.0.1 14 Mar 2012
built on: Thu Mar 19 15:18:31 UTC 2015
options:bn(64,32) rc4(8x,mmx) des(ptr,risc1,16,long) aes(partial) blowfish(idx)
compiler: cc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_
DLFCN_H -DL_ENDIAN -DTERMIO -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -W
format-security -Werror=format-security -D_FORTIFY_SOURCE=2 -Wl,-Bsymbolic-functions -Wl,-
z,relro -Wa,--noexecstack -Wall -DOPENSSL_NO_TLS1_2_CLIENT -DOPENSSL_BN_ASM_PART_WORDS -DO
PENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA
512_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM -DVPAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM

          sign    verify    sign/s verify/s
rsa 512 bits 0.000167s 0.000015s  5974.1  66137.2
rsa 1024 bits 0.000911s 0.000049s  1098.1  20599.5
rsa 2048 bits 0.006006s 0.000183s   166.5   5451.5
rsa 4096 bits 0.045136s 0.000755s    22.2   1324.9
[04/20/2015 19:19] seed@ubuntu:~/Bharat/Lab8$
```

Rsa speed benchmark.

```

[04/20/2015 19:21] seed@ubuntu:~/Bharat/Lab8$ openssl speed aes
Doing aes-128 cbc for 3s on 16 size blocks: 13956663 aes-128 cbc's in 2.95s
Doing aes-128 cbc for 3s on 64 size blocks: 3684786 aes-128 cbc's in 2.94s
Doing aes-128 cbc for 3s on 256 size blocks: 951319 aes-128 cbc's in 2.96s
Doing aes-128 cbc for 3s on 1024 size blocks: 491079 aes-128 cbc's in 2.98s
Doing aes-128 cbc for 3s on 8192 size blocks: 64495 aes-128 cbc's in 2.97s
Doing aes-192 cbc for 3s on 16 size blocks: 12284402 aes-192 cbc's in 2.97s
Doing aes-192 cbc for 3s on 64 size blocks: 3551879 aes-192 cbc's in 2.98s
Doing aes-192 cbc for 3s on 256 size blocks: 872432 aes-192 cbc's in 2.97s
Doing aes-192 cbc for 3s on 1024 size blocks: 411869 aes-192 cbc's in 2.96s
Doing aes-192 cbc for 3s on 8192 size blocks: 54822 aes-192 cbc's in 2.98s
Doing aes-256 cbc for 3s on 16 size blocks: 11070171 aes-256 cbc's in 2.97s
Doing aes-256 cbc for 3s on 64 size blocks: 2869504 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 256 size blocks: 718209 aes-256 cbc's in 2.97s
Doing aes-256 cbc for 3s on 1024 size blocks: 369602 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 8192 size blocks: 40022 aes-256 cbc's in 2.97s
OpenSSL 1.0.1 14 Mar 2012
built on: Thu Mar 19 15:18:31 UTC 2015
options:bn(64,32) rc4(8x,mmx) des(ptr,risc1,16,long) aes(partial) blowfish(idx)
compiler: cc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_
DLFCN_H -DL_ENDIAN -DTERMIO -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -W
format-security -Werror=format-security -D_FORTIFY_SOURCE=2 -Wl,-Bsymbolic-functions -Wl,-
z,relro -Wa,--noexecstack -Wall -DOPENSSL_NO_TLS1_2_CLIENT -DOPENSSL_BN_ASM_PART_WORDS -DO
PENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA
512_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM -DVPAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM
The 'numbers' are in 1000s of bytes per second processed.
type           16 bytes      64 bytes      256 bytes      1024 bytes      8192 bytes
aes-128 cbc      75697.16k    80213.03k    82276.24k    168746.61k    177893.28k
aes-192 cbc      66178.60k    76281.97k    75199.53k    142484.41k    150705.31k
aes-256 cbc      59637.28k    61626.93k    61906.23k    127004.18k    110390.65k
[04/20/2015 19:22] seed@ubuntu:~/Bharat/Lab8$

```

AES speed benchmark.

#### Custom shell script:

```

RSA Encryption:          3.93s
RSA Decryption:          5.42s
AES Encryption           4.02s
AES Decryption           3.70s

```

#### Speed Benchmark:

```

1024 bit private rsa's: 10893, 1024 bit private RSA's in 9.92s
1024 bit public rsa's: 204553, 1024 bit public RSA's in 9.93s
aes-128 cbc, 1024 size blocks: 491079 aes-128 cbc's in 2.98s

```

As per both the shell script and the speed benchmark **RSA is taking much greater time** to encrypt and decrypt than AES.

### 3.6 Task 6: Create Digital Signature

We will use the following example.txt file to generate the digest.

Next using the following command we will generate the public and private key pair:

```
'openssl genrsa -out private.pem 1024'
```

```
'openssl rsa -in private.pem -out public.pem -outform PEM -pubout'
```



```
Terminal
[04/20/2015 19:52] seed@ubuntu:~/Bharat/Lab8$ openssl genrsa -out private.pem 1024
Generating RSA private key, 1024 bit long modulus
.+++++
.....+++++
unable to write 'random state'
e is 65537 (0x10001)
[04/20/2015 19:52] seed@ubuntu:~/Bharat/Lab8$ openssl rsa -in private.pem -out public.pem -
outform PEM -pubout
writing RSA key
[04/20/2015 19:52] seed@ubuntu:~/Bharat/Lab8$ cat example.txt
*****
*****
**
**
**      This is the plain text we will be using for encryption.      **
**
**              Demo for Lab8.              **
**
**                                -Bharat                                **
*****
*****
[04/20/2015 19:52] seed@ubuntu:~/Bharat/Lab8$
```

Generate the key pair and the contents of the example.txt file

Next we generate the hash using the following command:

```
openssl dgst -sha256 < example.txt > example.hash
```

Next we sign the SHA 256 hash of example.txt:

```
openssl rsautl -sign -inkey private.pem -keyform PEM -in example.hash > example.sha256
```

Next we verify the digital signature in example.sha256 :

```
openssl rsautl -verify -inkey public.pem -keyform PEM -pubin -in example.sha256 > example.sha256.verified
```

```
Terminal
[04/20/2015 19:54] seed@ubuntu:~/Bharat/Lab8$ openssl dgst -sha256 < example.txt > example.
hash
[04/20/2015 19:54] seed@ubuntu:~/Bharat/Lab8$ openssl rsautl -sign -inkey private.pem -keyf
orm PEM -in example.hash > example.sha256
[04/20/2015 19:54] seed@ubuntu:~/Bharat/Lab8$ openssl rsautl -verify -inkey public.pem -key
form PEM -pubin -in example.sha256 > example.sha256.verified
[04/20/2015 19:54] seed@ubuntu:~/Bharat/Lab8$
```

Generate the signature of the hash of example.txt

Next we slightly modify example.txt and save it as example2.txt

```
Terminal
[04/20/2015 19:55] seed@ubuntu:~/Bharat/Lab8$ cat example2.txt
-----
--
--      This is the plain text we will be using for encryption.      --
--
--              Demo for Lab8.              --
--
--                                -Bharat                                --
-----
[04/20/2015 19:56] seed@ubuntu:~/Bharat/Lab8$
```

modified example.txt saved as example2.txt



Next we generate the hash using the following command:

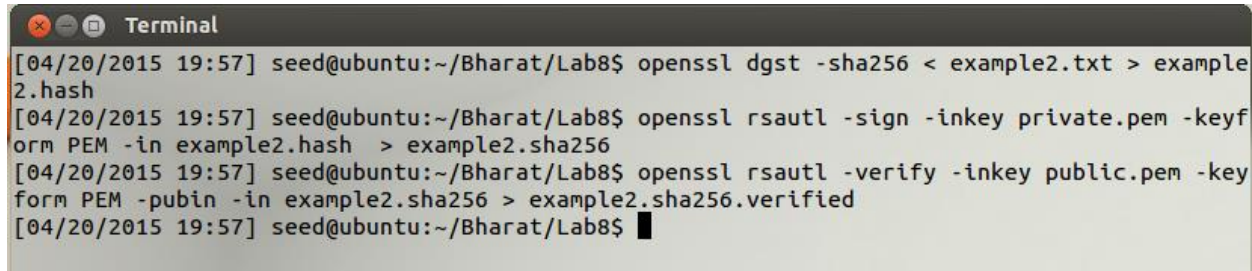
```
openssl dgst -sha256 < example2.txt > example2.hash
```

Next we sign the SHA 256 hash of example2.txt:

```
openssl rsautl -sign -inkey private.pem -keyform PEM -in example2.hash > example2.sha256
```

Next we verify the digital signature in example2.sha256 :

```
openssl rsautl -verify -inkey public.pem -keyform PEM -pubin -in example2.sha256 > example2.sha256.verified
```

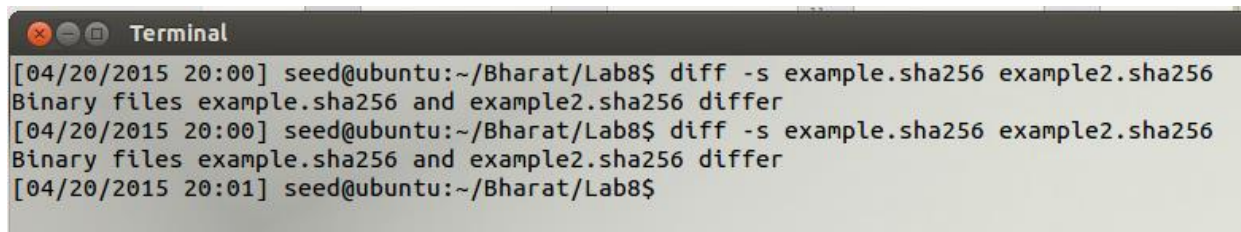


```
Terminal
[04/20/2015 19:57] seed@ubuntu:~/Bharat/Lab8$ openssl dgst -sha256 < example2.txt > example2.hash
[04/20/2015 19:57] seed@ubuntu:~/Bharat/Lab8$ openssl rsautl -sign -inkey private.pem -keyform PEM -in example2.hash > example2.sha256
[04/20/2015 19:57] seed@ubuntu:~/Bharat/Lab8$ openssl rsautl -verify -inkey public.pem -keyform PEM -pubin -in example2.sha256 > example2.sha256.verified
[04/20/2015 19:57] seed@ubuntu:~/Bharat/Lab8$
```

Generate the signature of the hash of example2.txt

Now that we have the hashes of both example.txt and example2.txt if we compare the hashes:

```
'diff -s example.sha256 example2.sha256'
```



```
Terminal
[04/20/2015 20:00] seed@ubuntu:~/Bharat/Lab8$ diff -s example.sha256 example2.sha256
Binary files example.sha256 and example2.sha256 differ
[04/20/2015 20:00] seed@ubuntu:~/Bharat/Lab8$ diff -s example.sha256 example2.sha256
Binary files example.sha256 and example2.sha256 differ
[04/20/2015 20:01] seed@ubuntu:~/Bharat/Lab8$
```

Comparison of hashes

Also, if we compare the signatures of the two hashes:

```
'diff -s example.sha256.verified example2.sha256.verified'
```



```
Terminal
[04/20/2015 19:59] seed@ubuntu:~/Bharat/Lab8$ diff example.sha256.verified example2.sha256.verified
1c1
< (stdin)= cf2d098988ecd3928d811cfe74bd206b0c4617e0d8e28fd84fd808a6711dc0c
---
> (stdin)= 4f08b4e1c0afbea35af18d006d0ad2f0e2c4d9005bb0527ea3a260221553ce13
[04/20/2015 19:59] seed@ubuntu:~/Bharat/Lab8$
```

comparison of signatures

A small change in file has completely changed the digital signature. Thus replicating the same signature is very difficult protecting our file making it very difficult to attack using standard dictionary attacks.