



CS5460 – ASSIGNMENT 2



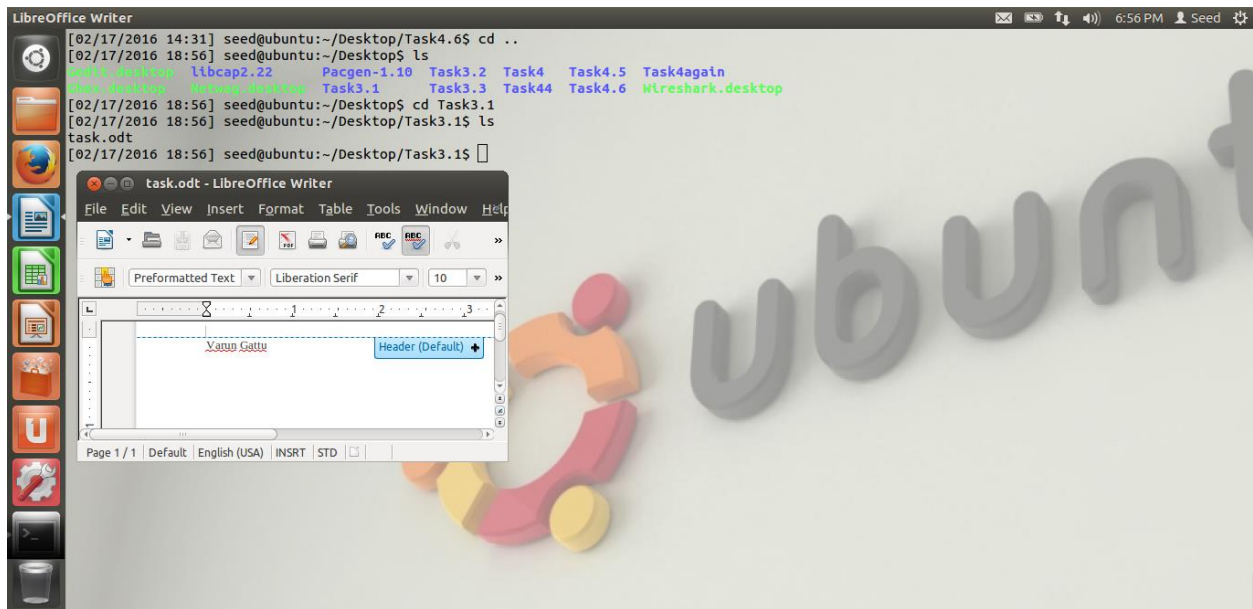
FEBRUARY 19, 2016

VARUN GATTU
A02092613

3. One-Way Hash Lab

3.1 Task 1: Creating Message Digest and MAC

To understand various one-way hash algorithms, I have created a file named task.odt with some data in it. The following screenshot shows it:



I have then used four different variants of the message digests. The digest types that I have used are:

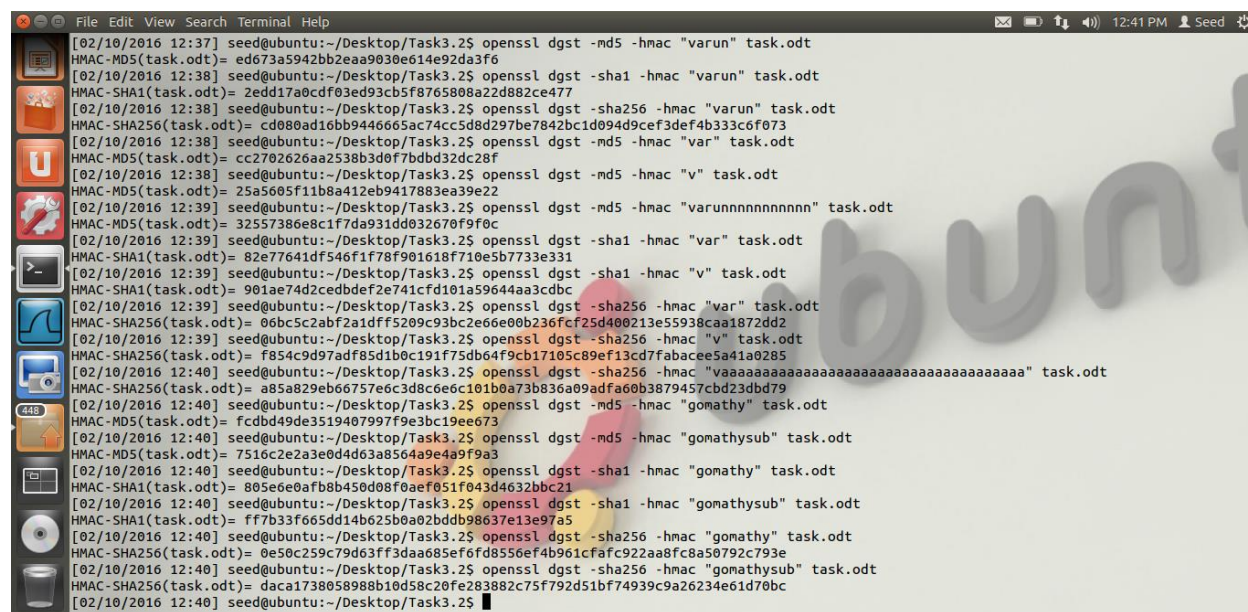
MD5, SHA. SHA1 and SHA256 as shown in the following screenshot.



I can see that MD5 creates the smallest hash of all hash functions because it produces a 128 bit hash value. The SHA hash creates a hash with 160 bits similar to SHA1. The only difference between SHA and SHA1 is that SHA1 has one extra bitwise rotation in it. Whereas SHA256 produces the biggest hash value of 256bits.

3.2 Keyed Hash and HMAC

Here, I have generated keyed hash for the file named task.odt. I have used MD5, SHA1 and SHA256 hashing methods to create the keyed hash. I have used various lengths for the string to be included as shown below:



```
[02/10/2016 12:37] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -md5 -hmac "varun" task.odt
HMAC-MD5(task.odt)= ed673a5942bb2eaa9030e614e92da3f6
[02/10/2016 12:38] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha1 -hmac "varun" task.odt
HMAC-SHA1(task.odt)= 2ed17a0cdf03ed93cb5f8765808a22d882ce477
[02/10/2016 12:38] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha256 -hmac "varun" task.odt
HMAC-SHA256(task.odt)= cd088ad16bb9446665ac74cc5d8d297be7842bc1d094d9cef3def4b333c6f073
[02/10/2016 12:38] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -md5 -hmac "var" task.odt
HMAC-MD5(task.odt)= cc2702626aa2538b3d0f7bdbc32dc28f
[02/10/2016 12:38] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -md5 -hmac "v" task.odt
HMAC-MD5(task.odt)= 25a5605f11b8a412eb9417883ea39e22
[02/10/2016 12:39] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -md5 -hmac "varunnnnnnnnnnn" task.odt
HMAC-MD5(task.odt)= 32557386e8c1f7da931dd032670f9f0c
[02/10/2016 12:39] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha1 -hmac "var" task.odt
HMAC-SHA1(task.odt)= 82e77641df546f1f78f901618f710e5b7733e331
[02/10/2016 12:39] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha1 -hmac "v" task.odt
HMAC-SHA1(task.odt)= 901ae74d2cedbdef2e741cf101a59644aa3cdbc
[02/10/2016 12:39] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha256 -hmac "var" task.odt
HMAC-SHA256(task.odt)= 06bc5c2abf2a1dff5209c93bc2e66e00b236fcf25d400213e55938caa1872dd2
[02/10/2016 12:39] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha256 -hmac "v" task.odt
HMAC-SHA256(task.odt)= f854c9d97adf85d1b0c191f75db64f9cb17105c89ef13cd7fabacee5a41a0285
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha256 -hmac "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" task.odt
HMAC-SHA256(task.odt)= a85a829eb66757e6c3d8c6e6c101b0a73b836a09adfa60b3879457cbd23dbd79
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -md5 -hmac "gomathy" task.odt
HMAC-MD5(task.odt)= fcd8d49de3519407997f9e3bc19ee673
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -md5 -hmac "gomathysub" task.odt
HMAC-MD5(task.odt)= 7516c2e2a3e0d4d63a8564a9e4a9f9a3
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha1 -hmac "gomathy" task.odt
HMAC-SHA1(task.odt)= 805e6e0afb8b450d08f0aef051f043d4632bbc21
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha1 -hmac "gomathysub" task.odt
HMAC-SHA1(task.odt)= ff7b33ff665dd14b625b0a02bddb98637e13e97a5
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha256 -hmac "gomathy" task.odt
HMAC-SHA256(task.odt)= 0e50c259c79d63ff3daa685ef6fd8556ef4b961cfaf9c22aa8fc8a50792c793e
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$ openssl dgst -sha256 -hmac "gomathysub" task.odt
HMAC-SHA256(task.odt)= daca1738058988b10d58c20fe283882c75f792d51bf74939c9a26234e61d70bc
[02/10/2016 12:40] seed@ubuntu:~/Desktop/Task3.2$
```

I could see that the hash value is not effected with the string length for any possible length. Similar to the Task3.1, MD5, SHA1 and SHA256 had 128bit, 160bit and 256bit keys respectively.

3.3 The randomness of One-way Hash

I have created a file named task.odt with some random data in it. I have used the file to create MD5 and SHA1 hash values on it. Then, I have changed a part of the file and again create similar hash on the new files. I could notice that the hash value completely changes in both the algorithms because they are based on various XOR operations. The hash value was same when I tried the hash on similar file for n number of times. But, once the file is tampered, hash value changes completely. The screenshot below shows my results:



Task 4: Public Key Cryptography Lab

4.1 Task1: Become a Certificate Authority (CA):

I have created a folder which has the files needed for this task. I have copied the openssl.cnf configuration file from /usr/lib/ssl/openssl.cnf to my local directory. The base folder had a subfolder called demoCA. demoCA folder had files called index.txt which is an empty database to store the certificate data. It also had a file called serial with some random number in it which will be the name of the certificate. It also had another folder named newcerts which holds the newly created certificates. The folder structure is as follows:



I have then created a self-signed certificate authority. The following screenshot shows how it is done:

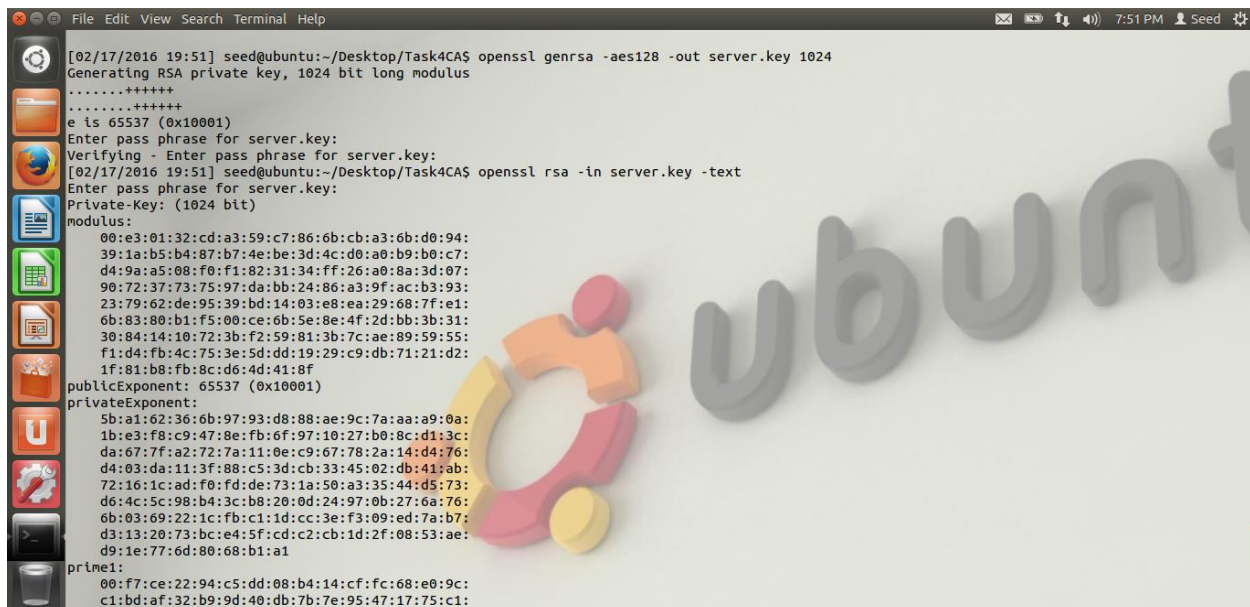


```
File Edit View Search Terminal Help
[02/17/2016 19:44] seed@ubuntu:~/Desktop/Task4CA$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM 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]:US
State or Province Name (full name) [Some-State]:US
Locality Name (eg, city) []:US
Organization Name (eg, company) [Internet Widgits Pty Ltd]:US
Organizational Unit Name (eg, section) []:US
Common Name (e.g. server FQDN or YOUR name) []:US
Email Address []:US
[02/17/2016 19:45] seed@ubuntu:~/Desktop/Task4CA$
```

4.2 Task 2: Create a Certificate for PKILabServer.com:

Step 1: Generate a public/private key pair:

I have created a public and private key pair with RSA algorithm. The following screenshots shows how this is achieved:



```
File Edit View Search Terminal Help
[02/17/2016 19:51] seed@ubuntu:~/Desktop/Task4CA$ openssl genrsa -aes128 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[02/17/2016 19:51] seed@ubuntu:~/Desktop/Task4CA$ openssl rsa -in server.key -text
Enter pass phrase for server.key:
Private-Key: (1024 bit)
modulus:
00:e3:01:32:cd:a3:59:c7:86:6b:cb:a3:6b:d0:94:
39:1a:b5:b4:87:b7:4e:be:3d:4c:d0:a0:b9:b0:c7:
d4:9a:a5:08:f0:f1:02:31:34:ff:26:a0:8a:3d:07:
90:72:37:73:75:97:da:bb:24:86:a3:9f:ac:b3:93:
23:79:62:de:95:39:bd:14:03:e8:ea:29:68:7f:e1:
6b:83:80:b1:f5:00:ce:6b:5e:8e:4f:2d:bb:3b:31:
30:84:14:10:72:3b:f2:59:81:3b:7c:ae:89:59:55:
f1:d4:fb:4c:75:3e:5d:dd:19:29:c9:db:71:21:d2:
1f:81:b8:fb:8c:d6:4d:41:8f
publicExponent: 65537 (0x10001)
privateExponent:
5b:a1:62:36:6b:97:93:d8:88:ae:9c:7a:aa:a9:0a:
1b:e3:f8:c9:47:8e:fb:6f:97:10:27:b0:8c:d1:3c:
da:67:7f:a2:72:7a:11:0e:c9:67:78:2a:14:d4:76:
d4:03:da:11:3f:88:c5:3d:cb:33:45:02:db:41:ab:
72:16:1c:ad:f0:fd:de:73:1a:50:a3:35:44:d5:73:
d6:4c:5c:98:b4:3c:b8:20:0d:24:97:0b:27:6a:76:
6b:03:69:22:1c:fb:c1:1d:cc:3e:f3:09:ed:7a:b7:
d3:13:20:73:bc:ae:45:f5:cd:c2:cb:1d:2f:08:53:ae:
d9:1e:77:6d:80:68:b1:a1
prime1:
00:f7:ce:22:94:c5:dd:08:b4:14:cf:fc:68:e0:9c:
c1:bd:af:32:b9:9d:40:db:7b:7e:95:47:17:75:c1:
```

```
File Edit View Search Terminal Help
exponent1:
68:09:d0:74:fb:b9:78:1c:fb:1b:f3:52:28:f3:47:
58:17:05:49:ee:9e:bb:47:56:f6:df:79:17:04:5a:
81:58:c9:1b:df:8e:e2:70:d9:56:ab:8c:7a:cb:13:
cd:29:ed:f7:34:58:68:09:16:51:64:db:9f:c9:7b:
6a:84:f5:39
exponent2:
45:f1:f8:60:0d:05:27:f4:92:fa:b8:1e:31:dd:16:
23:4a:30:5c:8b:9c:9b:69:a6:5c:d7:71:0b:d7:1a:
e3:36:0f:4f:8a:b9:0a:e3:f3:d8:c4:3b:96:a7:cb:
25:4b:15:fe:fa:8f:5b:b4:d5:85:eb:0c:76:e0:2c:
1e:2e:97:73
coefficient:
32:d2:e2:c3:78:d3:89:05:4e:03:99:e7:bf:44:a1:
d4:6c:dc:35:9d:6b:a1:ef:d7:fc:ee:46:ce:bc:be:
86:b0:34:83:bd:2c:f8:83:bb:55:fb:9a:25:51:fa:
d4:f8:9f:e5:15:4d:b4:88:2f:e8:eb:0f:c4:c2:fd:
b0:c4:fa:7b
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQDjATLNo1nHhmvLo2vQldkatbSHt06+PuzQoLmw9SapQjw8YIX
NP8moIo9B5ByN3N19q7JJa6yzyN5Yt6VOb0UA+jqKwh/4WuDLH1AM5rXoSP
Lbs7MTCEFBByO/JZgTt8roLZVfHU+0x1PL3dGSnJ23Eh0h+BuPuM1k1BjwIDAQAB
AoGAw6FlnmuXk9lIrp6gqkKG+P4yUeO+2+XECewjNE82md/onJ6EQ7JZ3gqFNR2
1APaET+IXT3LM0UC20GrchYcrfD93nMaUKM1RNVz1kxcmLQ8uCANJcLJ2p2awNp
Ihz7wR3MPvMJ7Xq30xMgc7zkX83Cyx0vCF0u2R53bYBosaECQDQ3ziKuxd0ItBTP
/GjgnMG9rzK5nUDbe36VRxd1xwJ6Mm6X+L5qyfz2Y0vcKCSFcyEMkpcFhdNPLsO
Gxq/wL4pAkeA6oL4eD7TYRg0r2DckscYwGfKqET2tXJXgSpMy45wAfgRuxCyr
UDAdqH/mt7r8ywt65kTlt4FvsL94f4o9wJaaAnQdPuSeBz7G/NSKPNHMBcFSe6e
u0dW9t95FwRagVjG9+04nDZVquMessTzSnt9zRYaAkHUNTbn8l7aoT10QJARFH4
YA0FJ/SS+rgeM0WI0owXIucm2mmXNdxC9ca4zYPT4q5CuPz2Mq7LqfLJUsV/vqP
w7TVhesMduAsHi6CwJAMtLlw3jTiQVOASnnv0Sh1GzCNZ1roe/X/0SGzry+hrA0
g70s+I07VfuaJVH61Pif5RVNTIgv60sPxML9sMT6ew==
-----END RSA PRIVATE KEY-----
[02/17/2016 19:51] seed@ubuntu:~/Desktop/Task4CA$
```

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

I have then created a Certificate signing request which will be sent to the Certificate Authority to generate a certificate for the public key.

```
File Edit View Search Terminal Help
[02/17/2016 19:57] seed@ubuntu:~/Desktop/Task4CA$ 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]:US
Locality Name (eg, city) []:US
Organization Name (eg, company) [Internet Widgits Pty Ltd]:US
Organizational Unit Name (eg, section) []:US
Common Name (e.g. server FQDN or YOUR name) []:US
Email Address []:US

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:dees
An optional company name []:dees
[02/17/2016 19:57] seed@ubuntu:~/Desktop/Task4CA$
```

Step 3: Generating Certificates:

I have now converted the CSR request into a certificate using our own certificate authority.



```
> -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: 4386 (0x1122)
    Validity
        Not Before: Feb 18 04:02:35 2016 GMT
        Not After : Feb 17 04:02:35 2017 GMT
    Subject:
        countryName           = US
        stateOrProvinceName   = US
        organizationName      = US
        organizationalUnitName = US
        commonName            = US
        emailAddress          = US
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        Netscape Comment:
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            EC:c1:59:2E:C4:D6:9D:17:57:79:6D:38:42:12:95:55:66:22:09:EC
        X509v3 Authority Key Identifier:
            keyId:DA:A6:67:42:8F:1D:D4:9C:F5:10:D2:CD:D2:0C:2C:3E:C7:AE:43:74
Certificate is to be certified until Feb 17 04:02:35 2017 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
[02/17/2016 20:02] seed@ubuntu:~/Desktop/Task4CA$
```

After successful creation of the certificate the folder structure looks like this:



```
[02/17/2016 20:04] seed@ubuntu:~/Desktop/Task4CA$ ls
ca.crt  ca.key  demoCA  openssl.cnf  server.crt  server.key
[02/17/2016 20:04] seed@ubuntu:~/Desktop/Task4CA$ cd demoCA
[02/17/2016 20:04] seed@ubuntu:~/Desktop/Task4CA/demoCA$ ls
index.txt  index.txt.attr  index.txt.old  newcerts  serial  serial.old
[02/17/2016 20:04] seed@ubuntu:~/Desktop/Task4CA/demoCA$ cd newcerts
[02/17/2016 20:05] seed@ubuntu:~/Desktop/Task4CA/demoCA/newcerts$ ls
1122.pem
[02/17/2016 20:05] seed@ubuntu:~/Desktop/Task4CA/demoCA/newcerts$
```


4.3 Task 3: Use PKI for Web Sites:

I have then added the PKILabServer.com address to the hosts file for it to be recognized by the system.



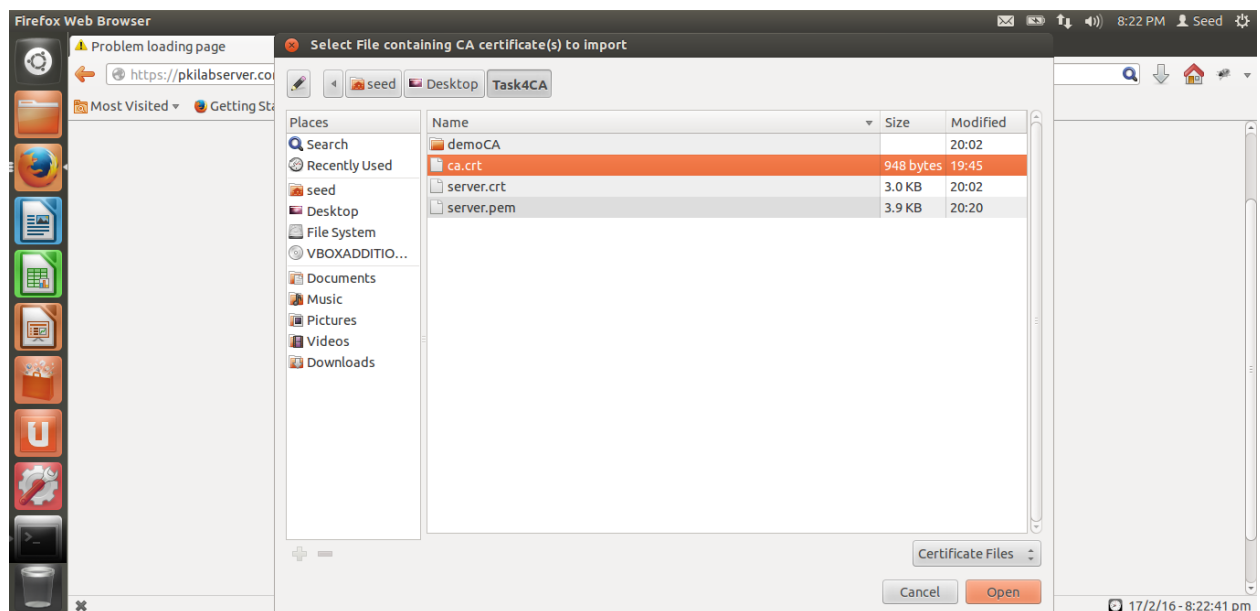
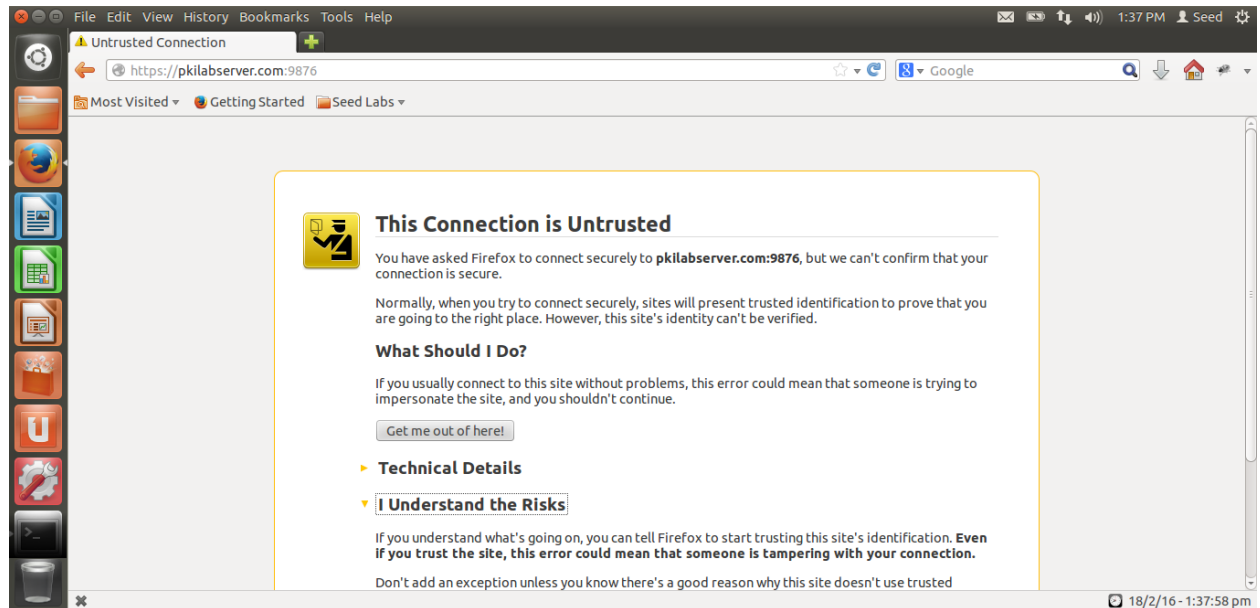
```
File Edit View Search Terminal Help
127.0.0.1 localhost
127.0.1.1 ubuntu
# The following lines are for SEED labs
127.0.0.1 www.OriginalPhpbb3.com
127.0.0.1 www.CSRFLabCollabtive.com
127.0.0.1 www.CSRFLabAttacker.com
127.0.0.1 www.SQLLabCollabtive.com
127.0.0.1 www.XSSLabCollabtive.com
127.0.0.1 www.SOPLab.com
127.0.0.1 www.SOPLabAttacker.com
127.0.0.1 www.SOPLabCollabtive.com
127.0.0.1 www.OriginalPhpMyAdmin.com
127.0.0.1 www.CSRFLabElgg.com
127.0.0.1 www.XSSLabElgg.com
127.0.0.1 www.SeedLabElgg.com
127.0.0.1 www.WTLabElgg.com
127.0.0.1 www.wtmobilestore.com
127.0.0.1 www.wtshoestore.com
127.0.0.1 www.wtelectronicstore.com
127.0.0.1 www.wtcamerastore.com
127.0.0.1 www.wtlabadserver.com
127.0.0.1 PKILabServer.com
# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
"/etc/hosts" 38L, 922C
1,1 Top
```

I have then combined the secret key and certificate into a single file and launched a server with the new certificate. Once the server starts running, we can access the website through a web browser.

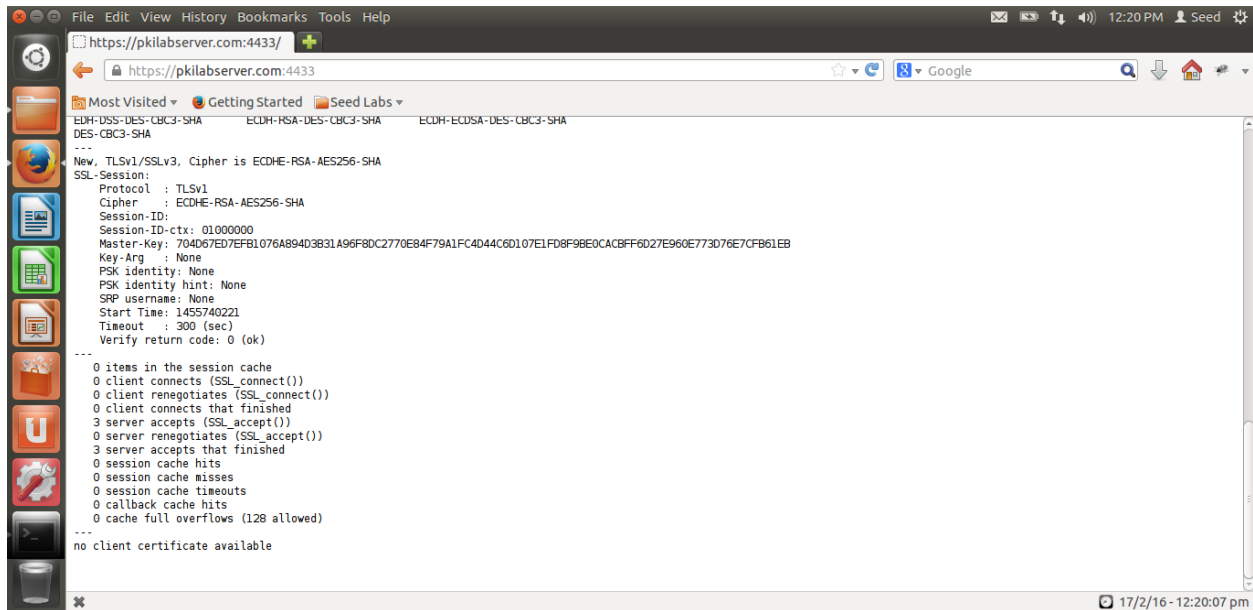
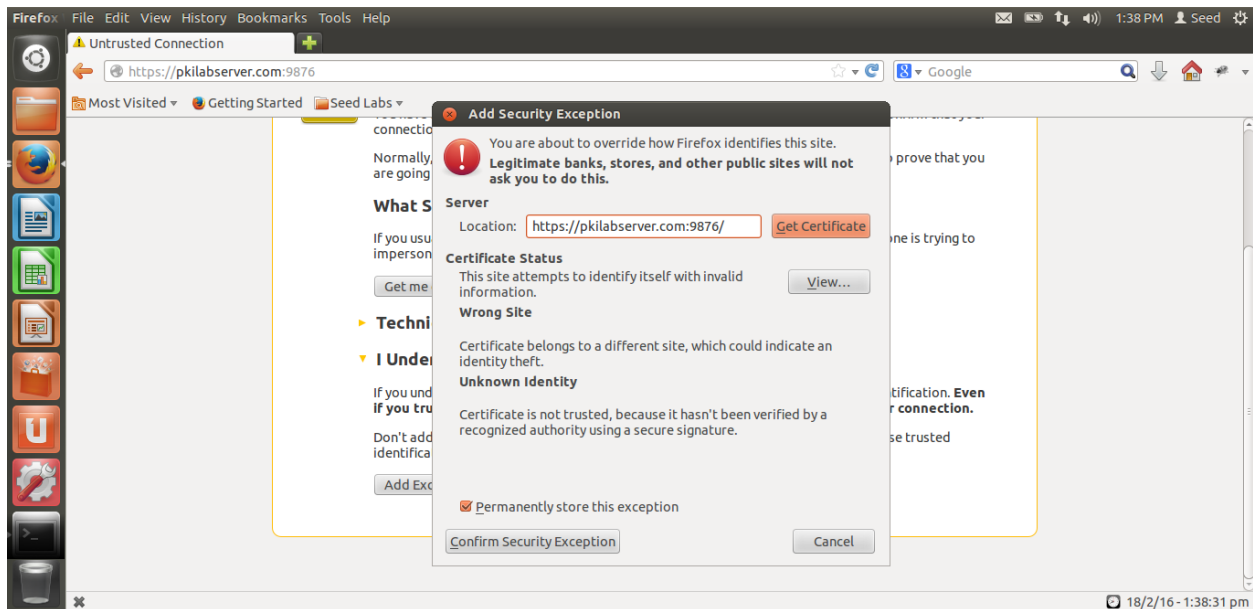


```
File Edit View Search Terminal Help
[02/17/2016 20:20] seed@ubuntu:~/Desktop/Task4CA$ cp server.key server.pem
[02/17/2016 20:20] seed@ubuntu:~/Desktop/Task4CA$ cat server.crt >> server.pem
[02/17/2016 20:20] seed@ubuntu:~/Desktop/Task4CA$ openssl s_server -cert server.pem -www -accept 5678
Enter pass phrase for server.pem:
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
```


Mozilla firefox will not allow to access the page because the certificate authority is not global and is not imported to the firefox. We do it in the following steps:



After importing and refreshing the browser, we will now be able to access the site.



Now, if we modify the server.pem and start the server again to check the website, it will not be displayed. The browser throws an error message saying “secure connection failed” and doesn’t allow to access the website.

If we use localhost instead of pkilabserver, we still get a warning stating the connection is not trusted because of the CA and hosts file data.

4.4 Task 4: Establishing a TLS/SSL connection with server

I have downloaded the files into a folder and extracted them. I have then copied the server.key and server.crt from the previous folder to the current one.

The code to check whether certificate is signed by authorized CA or not is here:

```
if (SSL_CTX_use_certificate_file(ctx, CERTF, SSL_FILETYPE_PEM) <= 0) {  
    ERR_print_errors_fp(stderr);  
    exit(-2);  
}
```

The code to check whether the certificate belongs to the server or not is here:

```
if (SSL_CTX_use_PrivateKey_file(ctx, KEYF, SSL_FILETYPE_PEM) <= 0) {  
    ERR_print_errors_fp(stderr);  
    exit(-3);  
}
```

The code to check whether the client is speaking to the server is here:

```
if (!SSL_CTX_check_private_key(ctx)) {  
    printf("Private key does not match the certificate public keyn");  
    exit(-4);  
}
```

The piece of code that verifies the client certificate is shown here:

```
if (SSL_CTX_use_certificate_file(ctx, CERTF, SSL_FILETYPE_PEM) <= 0) {  
    ERR_print_errors_fp(stderr);  
    exit(3);  
}
```

This part of the code is removed because it is unwanted.

The next screenshot shows the client code responsible for key exchange with the server:

```
/* Get server's certificate (note: beware of dynamic allocation) - opt */  
  
server_cert = SSL_get_peer_certificate (ssl);      CHK_NULL(server_cert);  
printf ("Server certificate:\n");  
  
str = X509_NAME_oneline (X509_get_subject_name (server_cert),0,0);  
CHK_NULL(str);  
printf ("\t subject: %s\n", str);  
OPENSSL_free (str);  
  
str = X509_NAME_oneline (X509_get_issuer_name (server_cert),0,0);  
CHK_NULL(str);  
printf ("\t issuer: %s\n", str);  
OPENSSL_free (str);
```

Similarly, the below screenshot shows the server side code that is responsible for key exchange:

```
client_cert = SSL_get_peer_certificate (ssl);  
if (client_cert != NULL) {  
    printf ("Client certificate:\n");  
  
    str = X509_NAME_oneline (X509_get_subject_name (client_cert), 0, 0);  
    CHK_NULL(str);  
    printf ("\t subject: %s\n", str);  
    OPENSSL_free (str);  
  
    str = X509_NAME_oneline (X509_get_issuer_name (client_cert), 0, 0);  
    CHK_NULL(str);  
    printf ("\t issuer: %s\n", str);  
    OPENSSL_free (str);
```

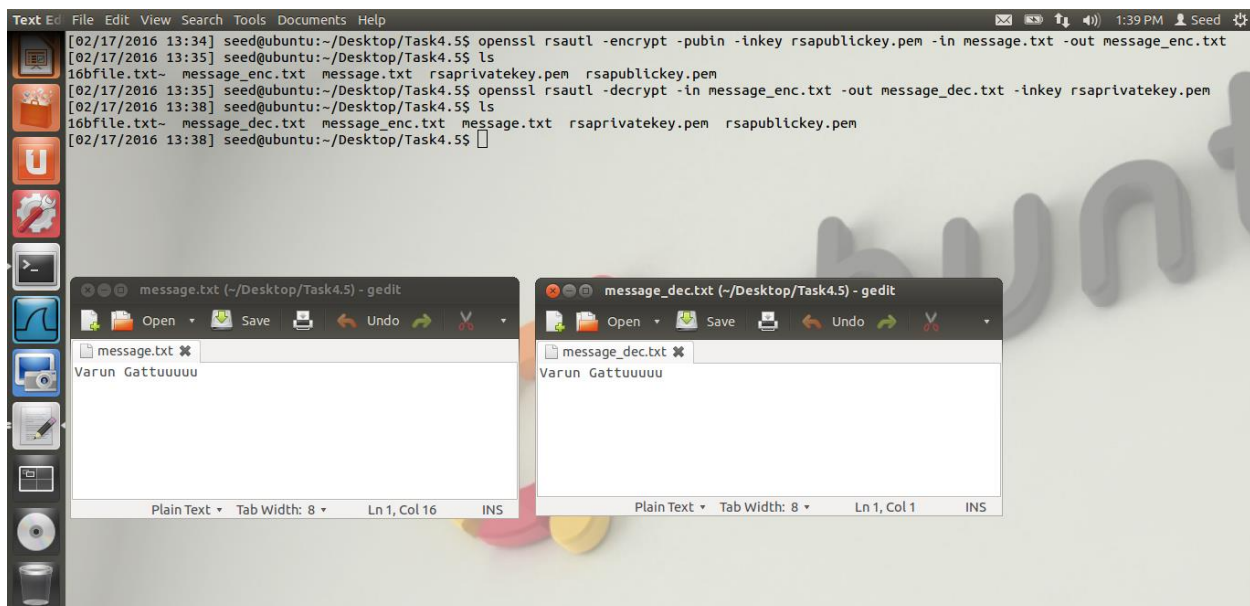

4.5 Task 5: Performance Comparison: RSA vs AES

For this task, I have created a file named message.txt which is of 16bytes. I have then generated a public and private key pair with RSA like the following:



```
[02/17/2016 12:28] seed@ubuntu:~/Desktop/Task44$ cd ..
[02/17/2016 13:22] seed@ubuntu:~/Desktop$ mkdir Task4.5
[02/17/2016 13:22] seed@ubuntu:~/Desktop$ vi 16bfile.txt
[02/17/2016 13:23] seed@ubuntu:~/Desktop$ cd Task4.5
[02/17/2016 13:29] seed@ubuntu:~/Desktop/Task4.5$ ls
16bfile.txt  16bfile.txt~
[02/17/2016 13:29] seed@ubuntu:~/Desktop/Task4.5$ ls
16bfile.txt  16bfile.txt~
[02/17/2016 13:29] seed@ubuntu:~/Desktop/Task4.5$ openssl genrsa -out rsaprivatekey.pem 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e 65537 (0x10001)
[02/17/2016 13:29] seed@ubuntu:~/Desktop/Task4.5$ ls
16bfile.txt  16bfile.txt~  rsaprivatekey.pem
[02/17/2016 13:29] seed@ubuntu:~/Desktop/Task4.5$ openssl rsa -in rsaprivatekey.pem -pubout
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDAPfFF66EpAtkBY8x1CRWVWMB
CGfH9A8x7VwEULPvRQ180Xo0CnhjEP0b0QPBDR+MZ5PT32rzrG54n7ESPpLp3Le
tfPhryuRvqbJ0dyhk0Z7ZUXNGMGKCLXia3hy1ZaqA/w7AmjoWuWdh+3zAPFRyCdr
xC/sSyh+kEDludCnJwIDAQAB
-----END PUBLIC KEY-----
[02/17/2016 13:30] seed@ubuntu:~/Desktop/Task4.5$
```

Now, I have encrypted the message.txt with the private key to create message_enc.txt. This is now decrypted using the public key. Now, the file is encrypted with 128 bit AES to create message_aes_enc.txt.



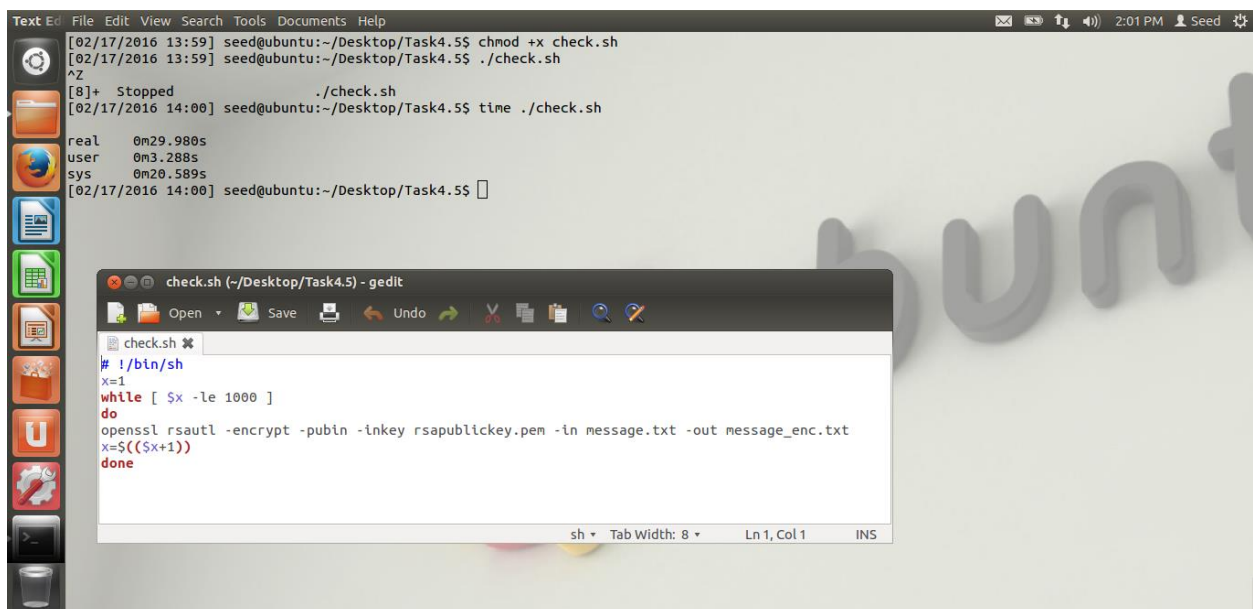
```
[02/17/2016 13:34] seed@ubuntu:~/Desktop/Task4.5$ openssl rsautl -encrypt -pubin -inkey rsapublickey.pem -in message.txt -out message_enc.txt
[02/17/2016 13:35] seed@ubuntu:~/Desktop/Task4.5$ ls
16bfile.txt  message_enc.txt  message.txt  rsaprivatekey.pem  rsapublickey.pem
[02/17/2016 13:35] seed@ubuntu:~/Desktop/Task4.5$ openssl rsautl -decrypt -in message_enc.txt -out message_dec.txt -inkey rsaprivatekey.pem
[02/17/2016 13:38] seed@ubuntu:~/Desktop/Task4.5$ ls
16bfile.txt  message_dec.txt  message_enc.txt  message.txt  rsaprivatekey.pem  rsapublickey.pem
[02/17/2016 13:38] seed@ubuntu:~/Desktop/Task4.5$
```

The image also shows two Gedit windows. The first window, titled 'message.txt (~/Desktop/Task4.5) - gedit', contains the text 'Varun Gattuuuuu'. The second window, titled 'message_dec.txt (~/Desktop/Task4.5) - gedit', also contains the text 'Varun Gattuuuuu', demonstrating successful decryption.



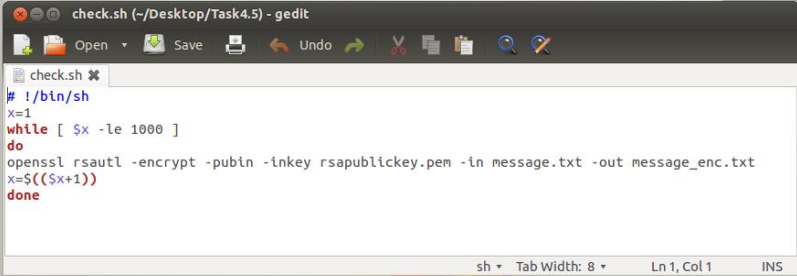
```
File Edit View Search Terminal Help
[02/17/2016 20:40] seed@ubuntu:~/Desktop/Task4.5$ openssl enc -aes-128-cbc -in message.txt -out message_aes_enc.txt -k dees
[02/17/2016 20:40] seed@ubuntu:~/Desktop/Task4.5$
```

To compare the time taken for both these processes, I have written a script which executes the commands 1000 times to calculate the total time as follows:



```
Text Ed File Edit View Search Tools Documents Help
[02/17/2016 13:59] seed@ubuntu:~/Desktop/Task4.5$ chmod +x check.sh
[02/17/2016 13:59] seed@ubuntu:~/Desktop/Task4.5$ ./check.sh
^Z
[8]+ Stopped ./check.sh
[02/17/2016 14:00] seed@ubuntu:~/Desktop/Task4.5$ time ./check.sh

real    0m29.980s
user    0m3.288s
sys     0m20.589s
[02/17/2016 14:00] seed@ubuntu:~/Desktop/Task4.5$
```

```
check.sh (~/.Desktop/Task4.5) - gedit
# !/bin/sh
x=1
while [ $x -le 1000 ]
do
openssl rsautl -encrypt -pubin -inkey rsapublickey.pem -in message.txt -out message_enc.txt
x=$((x+1))
done
```

```
Text Ed File Edit View Search Tools Documents Help
[02/17/2016 14:11] seed@ubuntu:~/Desktop/Task4.5$ vi check.sh
[02/17/2016 14:12] seed@ubuntu:~/Desktop/Task4.5$ time ./check.sh

real    0m39.849s
user    0m3.224s
sys     0m17.945s
[02/17/2016 14:12] seed@ubuntu:~/Desktop/Task4.5$

check.sh (~/Desktop/Task4.5) - gedit
# !/bin/sh
x=1
while [ $x -le 1000 ]
do
openssl enc -aes-128-cbc -in message.txt -out message_aes_enc.$x.txt -k dees
x=$((x+1))
done
```

This shows that AES encryption takes a little longer than RSA to be computed. Now, I have tried to run the speed command for RSA1024 and AES to calculate their times to run the commands. The output is like the following:

```
File Edit View Search Terminal Help
[02/17/2016 14:23] seed@ubuntu:~/Desktop/Task4.5$ openssl speed aes
Doing aes-128 cbc for 3s on 16 size blocks: 3391739 aes-128 cbc's in 2.72s
Doing aes-128 cbc for 3s on 64 size blocks: 935363 aes-128 cbc's in 2.69s
Doing aes-128 cbc for 3s on 256 size blocks: 248586 aes-128 cbc's in 2.77s
Doing aes-128 cbc for 3s on 1024 size blocks: 58928 aes-128 cbc's in 2.81s
Doing aes-128 cbc for 3s on 8192 size blocks: 7909 aes-128 cbc's in 2.81s
Doing aes-192 cbc for 3s on 16 size blocks: 2664952 aes-192 cbc's in 2.76s
Doing aes-192 cbc for 3s on 64 size blocks: 758488 aes-192 cbc's in 2.71s
Doing aes-192 cbc for 3s on 256 size blocks: 203441 aes-192 cbc's in 2.82s
Doing aes-192 cbc for 3s on 1024 size blocks: 51683 aes-192 cbc's in 2.81s
Doing aes-192 cbc for 3s on 8192 size blocks: 6471 aes-192 cbc's in 2.81s
Doing aes-256 cbc for 3s on 16 size blocks: 2341159 aes-256 cbc's in 2.77s
Doing aes-256 cbc for 3s on 64 size blocks: 690454 aes-256 cbc's in 2.81s
Doing aes-256 cbc for 3s on 256 size blocks: 173311 aes-256 cbc's in 2.67s
Doing aes-256 cbc for 3s on 1024 size blocks: 40360 aes-256 cbc's in 2.79s
Doing aes-256 cbc for 3s on 8192 size blocks: 5646 aes-256 cbc's in 2.80s
OpenSSL 1.0.1 14 Mar 2012
built on: Tue Jun 4 07:25:48 UTC 2013
options:bn(64,32) rc4(8x,mxm) des(ptr,rlsc1,16,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 -Wformat-security -Werror=format-security -D_FORTIFY_SOURCE=2 -Wl,-Bsymbolic-functions -Wl,-z,relro -Wa,-noex
ecstack -Wall -DOPENSSL_NO_TLS1_2_CLIENT -DOPENSSL_MAX_TLS1_2_CIPHER_LENGTH=50 -DOPENSSL_BN_ASM_PART_WORDS -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM
MONT -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_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 19951.41k 22253.99k 22974.01k 21474.12k 23057.13k
aes-192 cbc 15449.00k 17912.63k 18468.40k 18833.95k 18864.92k
aes-256 cbc 13522.94k 15725.64k 16617.08k 14813.13k 16518.58k
[02/17/2016 14:24] seed@ubuntu:~/Desktop/Task4.5$
```

```
File Edit View Search Terminal Help
[02/17/2016 14:22] seed@ubuntu:~/Desktop/Task4.5$ openssl speed rsa1024
Doing 1024 bit private rsa's for 10s: 7469 1024 bit private RSA's in 9.57s
Doing 1024 bit public rsa's for 10s: 136243 1024 bit public RSA's in 9.71s
OpenSSL 1.0.1 14 Mar 2012
built on: Tue Jun  4 07:25:48 UTC 2013
options:bn(64,32) rc4(8x,mmx) des(ptr,risc1,16,long) aes(partial) blowfish(idx)
compiler: cc -fPIC -DOPENSSL_PIC -DLIB -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DL_ENDIAN -DTERMIO -g -O2 -fstack-protector -
-param-ssp-buffer-size=4 -Wformat -Wformat-security -Werror=format-security -D_FORTIFY_SOURCE=2 -Wl,-Bsymbolic-functions -Wl,-z,relro -Wa,-noex
ecstack -Wall -DOPENSSL_NO_TLS1_2_CLIENT -DOPENSSL_MAX_TLS1_2_CIPHER_LENGTH=50 -DOPENSSL_BN_ASM_PART_WORDS -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM
MONT -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM -DVPAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM
sign verify sign/s verify/s
rsa 1024 bits 0.001281s 0.000071s 780.5 14031.2
[02/17/2016 14:22] seed@ubuntu:~/Desktop/Task4.5$
```

This shows that RSA is slower than AES to execute the commands.

4.6 Task 6: Create Digital Signature:

I have created a file named example.txt with some random text in it. I have also created a public and private key pair with RSA for the same.

```
File Edit View Search Terminal Help
[02/17/2016 14:24] seed@ubuntu:~/Desktop/Task4.5$ cd ..
[02/17/2016 14:24] seed@ubuntu:~/Desktop$ mkdir Task4.6
[02/17/2016 14:24] seed@ubuntu:~/Desktop$ cd Task4.6
[02/17/2016 14:24] seed@ubuntu:~/Desktop/Task4.6$ vi example.txt
[02/17/2016 14:25] seed@ubuntu:~/Desktop/Task4.6$ openssl genrsa -out rsaprivatekey.pem 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
[02/17/2016 14:26] seed@ubuntu:~/Desktop/Task4.6$ openssl rsa -in rsaprivatekey.pem -pubout
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCsGSIb3DQEBAQUAA4GNADCBiQKBgQDLZ0LID6n1cFy57cLh8k88cVr4
8eepmw7ReQMxz1NCvr+fxqWAd451cIsGDrSdhFfnfynEOYXMSNcm+rD7t41ZB+wb
VguxKzThe45UqtAEqBaBj3Jv7OoYJzc6Yd+/t2JESrjCNTVs2vgOKmA/HyhW9uS
13U0aZV5IjbW13LQIDAQAB
-----END PUBLIC KEY-----
[02/17/2016 14:26] seed@ubuntu:~/Desktop/Task4.6$ ls
example.txt  rsaprivatekey.pem
[02/17/2016 14:26] seed@ubuntu:~/Desktop/Task4.6$ openssl rsa -in rsaprivatekey.pem -out rsapublickey.txt -pubout
writing RSA key
[02/17/2016 14:27] seed@ubuntu:~/Desktop/Task4.6$ ls
example.txt  rsaprivatekey.pem  rsapublickey.txt
[02/17/2016 14:27] seed@ubuntu:~/Desktop/Task4.6$
```


I then signed the example.txt with SHA256 hash to create example.sha256. I have verified the digital signature as below:

A terminal window on an Ubuntu desktop. The terminal shows three commands and their outputs: 1. 'openssl dgst -sha256 -sign rsaprivatekey.pem -out example.sha256 example.txt' is executed. 2. 'openssl dgst -sha256 -verify rsapublickey.txt -signature example.sha256 example.txt' is executed, resulting in 'Verified OK'. 3. A third identical verification command is executed, also resulting in 'Verified OK'. The desktop background features the Ubuntu logo and the word 'ubuntu' in large letters.

```
[02/17/2016 14:30] seed@ubuntu:~/Desktop/Task4.6$ openssl dgst -sha256 -sign rsaprivatekey.pem -out example.sha256 example.txt
[02/17/2016 14:30] seed@ubuntu:~/Desktop/Task4.6$ openssl dgst -sha256 -verify rsapublickey.txt -signature example.sha256 example.txt
Verified OK
[02/17/2016 14:30] seed@ubuntu:~/Desktop/Task4.6$
```

Now, I tried to modify the example.txt and then verify the signature. It doesn't verify the signature because of different hash values for both of them as below:

A terminal window on an Ubuntu desktop. The terminal shows four commands: 1. 'openssl dgst -sha256 -sign rsaprivatekey.pem -out example.sha256 example.txt' is executed. 2. 'openssl dgst -sha256 -verify rsapublickey.txt -signature example.sha256 example.txt' is executed, resulting in 'Verified OK'. 3. 'vi example.txt' is executed to modify the file. 4. 'openssl dgst -sha256 -verify rsapublickey.txt -signature example.sha256 example.txt' is executed again, resulting in 'Verification Failure'. The desktop background features the Ubuntu logo and the word 'ubuntu' in large letters.

```
[02/17/2016 14:30] seed@ubuntu:~/Desktop/Task4.6$ openssl dgst -sha256 -sign rsaprivatekey.pem -out example.sha256 example.txt
[02/17/2016 14:30] seed@ubuntu:~/Desktop/Task4.6$ openssl dgst -sha256 -verify rsapublickey.txt -signature example.sha256 example.txt
Verified OK
[02/17/2016 14:30] seed@ubuntu:~/Desktop/Task4.6$ vi example.txt
[02/17/2016 14:31] seed@ubuntu:~/Desktop/Task4.6$ openssl dgst -sha256 -verify rsapublickey.txt -signature example.sha256 example.txt
Verification Failure
[02/17/2016 14:31] seed@ubuntu:~/Desktop/Task4.6$
```