OPENSSL AND ITS APPLICATIONS

PROJECT REPORT

PRESENTED BY:

RAHUL RANJAN SINHA(56)
GAURAV KUMAR (2)

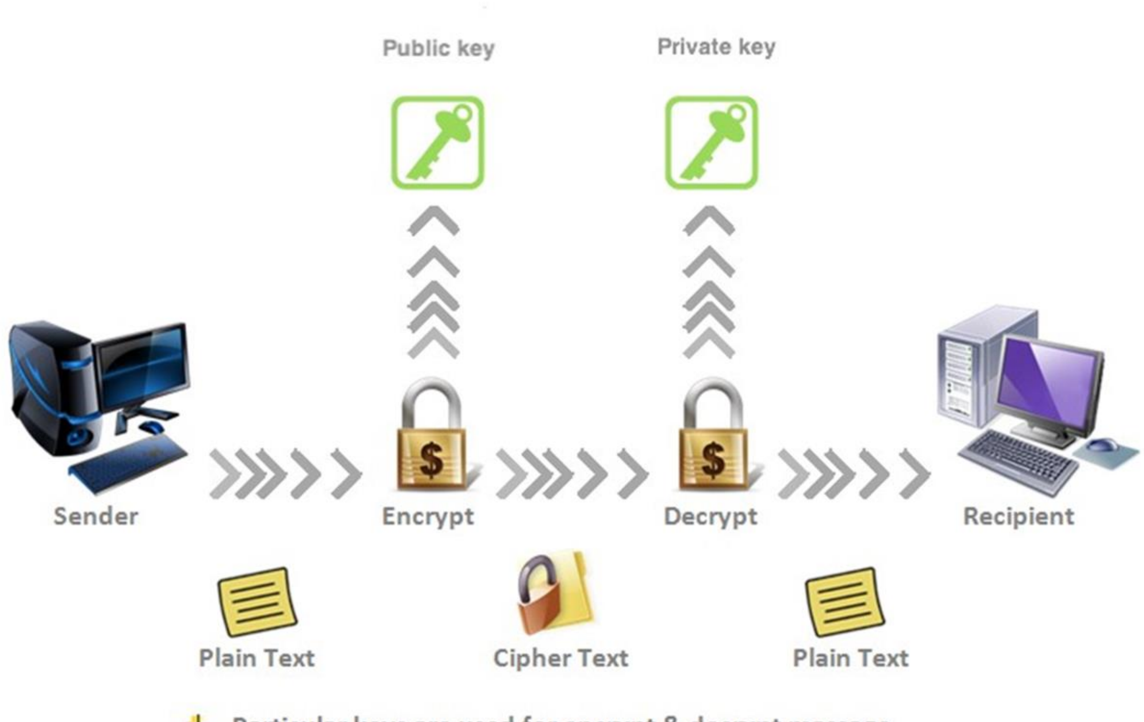
OPENSSL

OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used in internet web servers, serving a majority of all web sites.

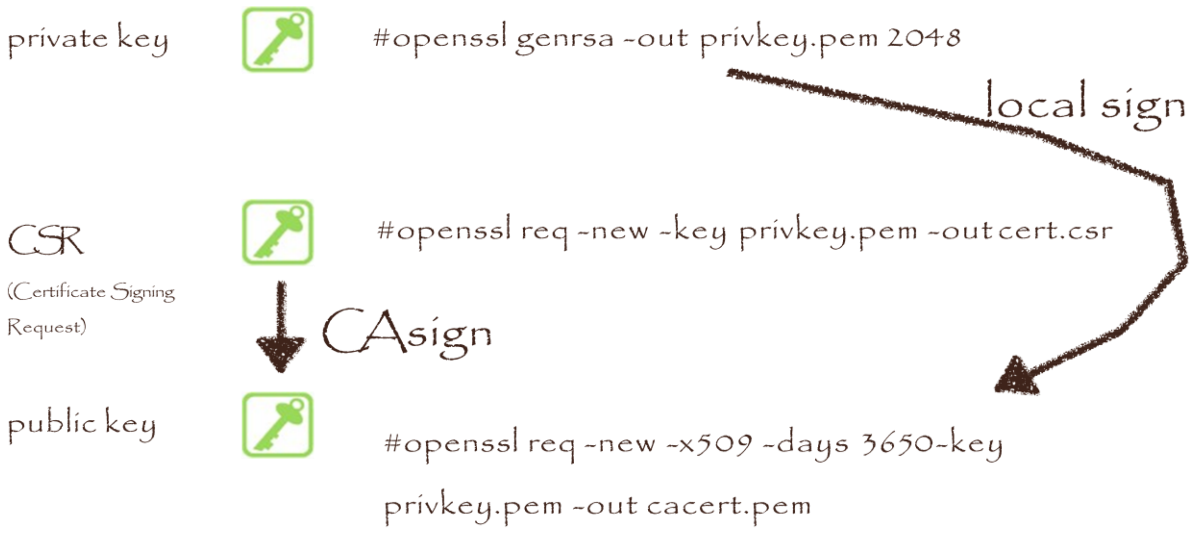
OpenSSL contains an open-source implementation of the SSL and TLS protocols. The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

Versions are available for most Unix and Unix-like operating systems (including Solaris, Linux, macOS, QNX, and the various open-source BSD operating systems), OpenVMS and Microsoft Windows. IBM provides a port for the System i (OS/400).

THE BASIC OF OPENSSL



Particular keys are used for encrypt & decrypt massage



private key

```
#openssl genrsa -out privkey.pem 2048
```

local sign

CSR
(Certificate Signing Request)

```
#openssl req -new -key privkey.pem -out cert.csr
```

CAsign

public key

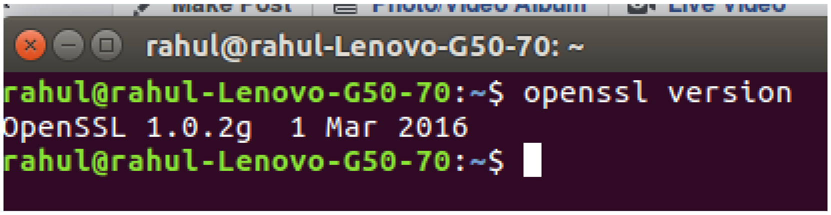
```
#openssl req -new -x509 -days 3650-key privkey.pem -out cacert.pem
```

Installing OpenSSL Toolkit

```
wget http://www.openssl.org/source/openssl-1.0.1g.tar.gz
tar -xvzf openssl-1.0.1g.tar.gz
cd openssl-1.0.1g
./config --prefix=/usr/
make
sudo make install
```

Checking the version of OpenSSL

$openssl version



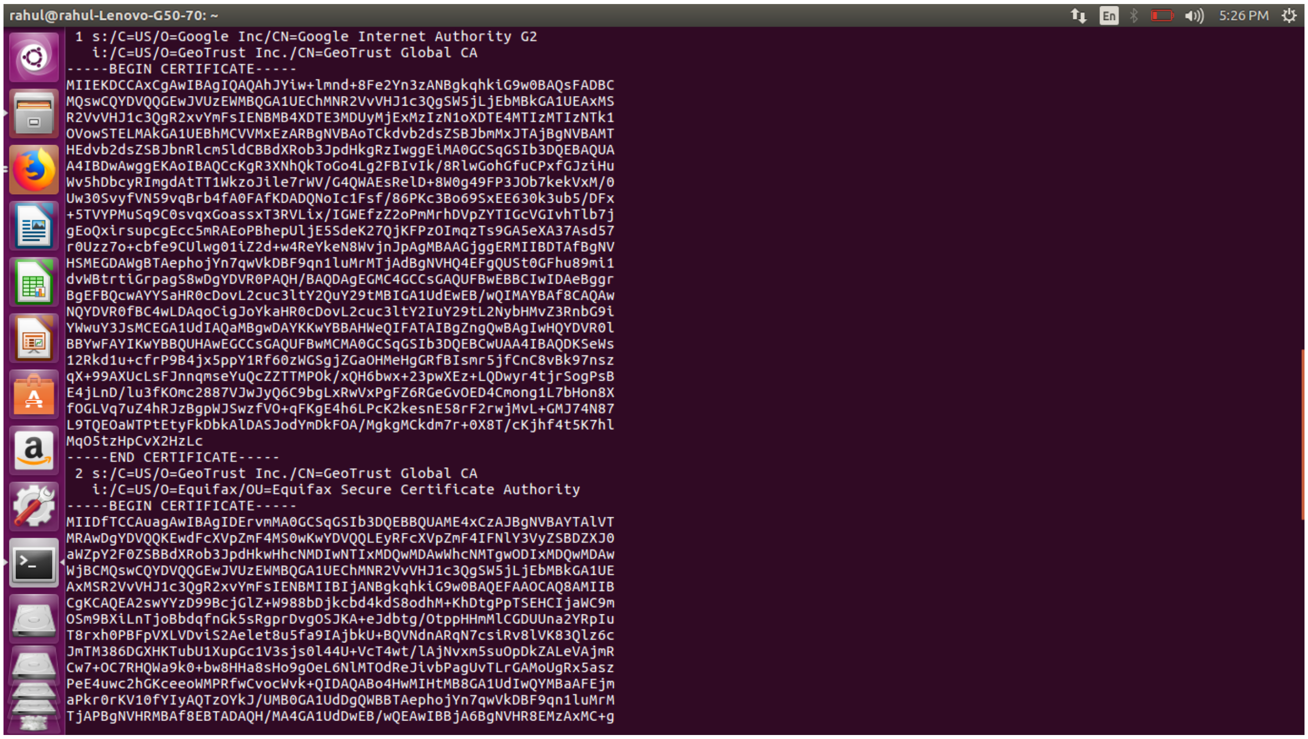
```
rahul@rahul-Lenovo-G50-70:~$ openssl version
OpenSSL 1.0.2g  1 Mar 2016
rahul@rahul-Lenovo-G50-70:~$
```

APPLICATIONS OF Openssl

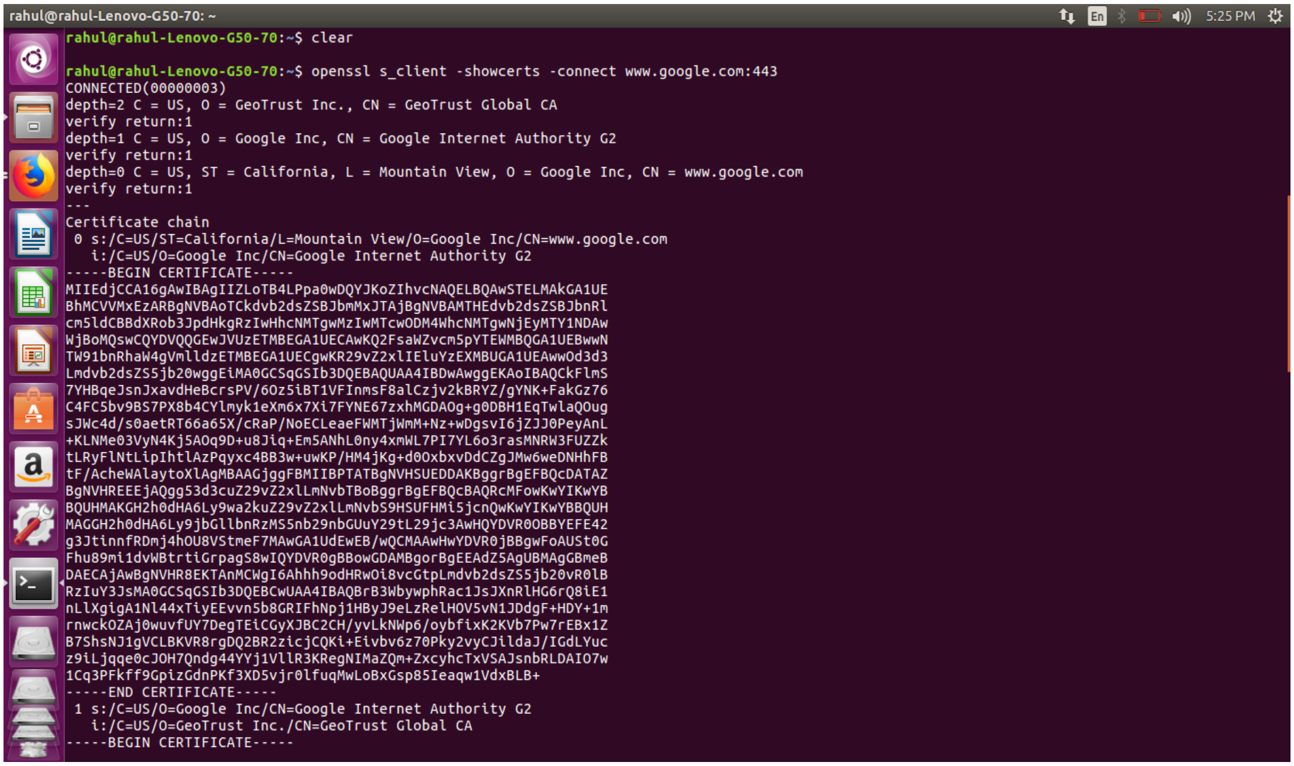
Checking A Remote Certificate Chain With OpenSSL

If we deal with SSL/TLS long enough we will run into situations where we need to examine what certificates are being presented by a server to the client. The best way to examine the raw output is via OpenSSL.

$openssl s_client -showcerts -connect www.google.com:443



```
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
-----BEGIN CERTIFICATE-----
MIIEKDCCAxCgAwIBAgIQAQAhJYiw+lmnd+8Fe2Yn3zANBgkqhkiG9w0BAQsFADBC
MQswCQYDVQQGEwJVUzEWMBQGA1UEChMNR2VvVHJ1c3QgSW5jLjEbMBkGA1UEAxMS
R2VvVHJ1c3QgR2xvYmFsIENBMB4XDTE3MDUyMjExMzIzN1oXDTE4MTIzMTIzNTk1
OVowSTELMAkGA1UEBhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMT
HEdvb2dsZSBJbnRlcm5ldCBBdXRob3JpdHkgRzIwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQCcKgR3XNhQkToGo4Lg2FBIvIk/8RlwGohGfuCPxfGJziHu
Wv5hDbcyRImgdAtTT1WkzoJile7rWV/G4QWAEsRelD+8W0g49FP3JOb7kekVxM/0
Uw30SvyfVN59vqBrb4fA0FAfKDADQNoIc1Fsf/86PKc3Bo69SxEE630k3ub5/DFx
+5TVYPMuSq9C0svqxGoassxT3RVLix/IGWEfzZ2oPmMrhDVpZYTIGcVGIvhTlb7j
gEoQxirsupcgEcc5mRAEoPBhepUljE5SdeK27QjKFPzOImqzTs9GA5eXA37Asd57
r0Uzz7o+cbfe9CUlwg01iZ2d+w4ReYkeN8WvjnJpAgMBAAGjggERMIIBDTAfBgNV
HSMEGDAWgBTAephojYn7qwVkDBF9qn1luMrMTjAdBgNVHQ4EFgQUSt0GFhu89mi1
dvWBtrtiGrpagS8wDgYDVR0PAQH/BAQDAgEGMC4GCCsGAQUFBwEBBCIwIDAeBggr
BgEFBQcwAYYSaHR0cDovL2cuc3ltY2QuY29tMBIGA1UdEwEB/wQIMAYBAf8CAQAw
NQYDVR0fBC4wLDAqoCigJoYkaHR0cDovL2cuc3ltY2IuY29tL2NybHMvZ3RnbG9i
YWwuY3JsMCEGA1UdIAQaMBgwDAYKKwYBBAHWeQIFATAIBgZngQwBAgIwHQYDVR0l
BBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMA0GCSqGSIb3DQEBCwUAA4IBAQDKSeWs
12Rkd1u+cfrP9B4jx5ppY1Rf60zWGSgjZGaOHMeHgGRfBIsmr5jfCnC8vBk97nsz
qX+99AXUcLsFJnnqmseYuQcZZTTMPOk/xQH6bwx+23pwXEz+LQDwyr4tjrSogPsB
E4jLnD/lu3fKOmc2887VJwJyQ6C9bgLxRwVxPgFZ6RGeGvOED4Cmong1L7bHon8X
fOGLVq7uZ4hRJzBgpWJSwzfVO+qFKgE4h6LPcK2kesnE58rF2rwjMvL+GMJ74N87
L9TQEOaWTPtEtyFkDbkAlDASJodYmDkFOA/MgkgMCkdm7r+0X8T/cKjhf4t5K7hl
MqO5tzHpCvX2HzLc
-----END CERTIFICATE-----
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
-----BEGIN CERTIFICATE-----
MIIDfTCCAuagAwIBAgIDErvmMA0GCSqGSIb3DQEBBQUAME4xCzAJBgNVBAYTAlVT
MRAwDgYDVQQKEwdFcXVpZmF4MS0wKwYDVQQLEyRFcXVpZmF4IFNlY3VyZSBDZXJ0
aWZpY2F0ZSBBdXRob3JpdHkwHhcNMDIwNTIxMDQwMDAwWhcNMTgwODIxMDQwMDAw
WjBCMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNR2VvVHJ1c3QgSW5jLjEbMBkGA1UE
AxMSR2VvVHJ1c3QgR2xvYmFsIENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEA2swYYzD99BcjGlZ+W988bDjkcbd4kdS8odhM+KhDtgPpTSEHCIjaWC9m
OSm9BXiLnTjoBbdqfnGk5sRgprDvgOSJKA+eJdbtg/OtppHHmMlCGDUUna2YRpIu
T8rxh0PBFpVXLVDviS2Aelet8u5fa9IAjbkU+BQVNdnARqN7csiRv8lVK83Qlz6c
JmTM386DGXHKTubU1XupGc1V3sjs0l44U+VcT4wt/lAjNvxm5suOpDkZALeVAjmR
Cw7+OC7RHQWa9k0+bw8HHa8sHo9gOeL6NlMTOdReJivbPagUvTLrGAMoUgRx5asz
PeE4uwc2hGKceeoWMPRfwCvocWvk+QIDAQABo4HwMIHtMB8GA1UdIwQYMBaAFEjm
aPkr0rKV10fYIyAQTzOYkJ/UMB0GA1UdDgQWBBTAephojYn7qwVkDBF9qn1luMrM
TjAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBBjA6BgNVHR8EMzAxMC+g
```



```
rahul@rahul-Lenovo-G50-70:~$ clear

rahul@rahul-Lenovo-G50-70:~$ openssl s_client -showcerts -connect www.google.com:443
CONNECTED(00000003)
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify return:1
depth=1 C = US, O = Google Inc, CN = Google Internet Authority G2
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google Inc, CN = www.google.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
-----BEGIN CERTIFICATE-----
MIIEdjCCA16gAwIBAgIIZLoTB4LPpa0wDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
cm5ldCBBdXRob3JpdHkgRzIwHhcNMTgwMzIwMTcwODM4WhcNMTgwNjEyMTY1NDAw
WjBoMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTEWMBQGA1UEBwwN
TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2xlIEluYzEXMBUGA1UEAwwOd3d3
Lmdvb2dsZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCkFlmS
7YHBqeJsnJxavdHeBcrsPV/6Oz5iBT1VFInmsF8alCzjv2kBRYZ/gYNK+FakGz76
C4FC5bv9BS7PX8b4CYlmyk1eXm6x7Xi7FYNE67zxhMGDAOg+g0DBH1EqTwlaQOug
sJWc4d/s0aetRT66a65X/cRaP/NoECLeaeFWMTjWmM+Nz+wDgsvI6jZJJ0PeyAnL
+KLNMe03VyN4Kj5AOq9D+u8Jiq+Em5ANhL0ny4xmWL7PI7YL6o3rasMNRW3FUZZk
tLRyFlNtLipIhtlAzPqyxc4BB3w+uwKP/HM4jKg+d0OxbxvDdCZgJMw6weDNHhFB
tF/AcheWAlaytoXlAgMBAAGjggFBMIIBPTATBgNVHSUEDDAKBggrBgEFBQcDATAZ
BgNVHREEEjAQgg53d3cuZ29vZ2xlLmNvbTBoBggrBgEFBQcBAQRcMFowKwYIKwYB
BQUHMAKGH2h0dHA6Ly9wa2kuZ29vZ2xlLmNvbS9HSUFHMi5jcnQwKwYIKwYBBQUH
MAGGH2h0dHA6Ly9jbGllbnRzMS5nb29nbGUuY29tL29jc3AwHQYDVR0OBBYEFE42
g3JtinnfRDmj4hOU8VStmeF7MAwGA1UdEwEB/wQCMAAwHwYDVR0jBBgwFoAUSt0G
Fhu89mi1dvWBtrtiGrpagS8wIQYDVR0gBBowGDAMBgorBgEEAdZ5AgUBMAgGBmeB
DAECAjAwBgNVHR8EKTAnMCWgI6Ahhh9odHRwOi8vcGtpLmdvb2dsZS5jb20vR0lB
RzIuY3JsMA0GCSqGSIb3DQEBCwUAA4IBAQBrB3WbywphRac1JsJXnRlHG6rQ8iE1
nLlXgigA1Nl44xTiyEEvvn5b8GRIFhNpj1HByJ9eLzRelHOV5vN1JDdgF+HDY+1m
rnwckOZAj0wuvfUY7DegTEiCGyXJBC2CH/yvLkNWp6/oybfixK2KVb7Pw7rEBx1Z
B7ShsNJ1gVCLBKVR8rgDQ2BR2zicjCQKi+Eivbv6z70Pky2vyCJildaJ/IGdLYuc
z9iLjqqe0cJOH7Qndg44YYj1VllR3KRegNIMaZQm+ZxcyhcTxVSAJsnbRLDAIO7w
1Cq3PFkff9GpizGdnPKf3XD5vjr0lfuqMwLoBxGsp85Ieaqw1VdxBLB+
-----END CERTIFICATE-----
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
-----BEGIN CERTIFICATE-----
```

The following points can be observed by checking a remote certificate chain with OpenSsl:

1. The certificate chain consists of two certificates. At level 0 there is the server certificate with some parsed information. s: is the subject line of the certificate and i: contains information about the issuing CA.

2. This particular server (www.woot.com) has sent an intermediate certificate as well. Subject and issuer information is provided for each certificate in the presented chain. Chains can be much longer than 2 certificates in length.

3. The server certificate section is a duplicate of level 0 in the chain. If you're only looking for the end entity certificate then you can rapidly find it by looking for this section.

4. No client certificate CAs were sent. If the server was configured to potentially accept client certs the returned data would include a list of "acceptable client CAs".

5. Connection was made via TLSv1/SSLv3 and the chosen cipher was RC4-MD5. Incidentally, this typically means that the server you're connecting to is IIS.

Encryption Techniques

Symmetric Encryption

Asymmetric Encryption

Symmetric Encryption

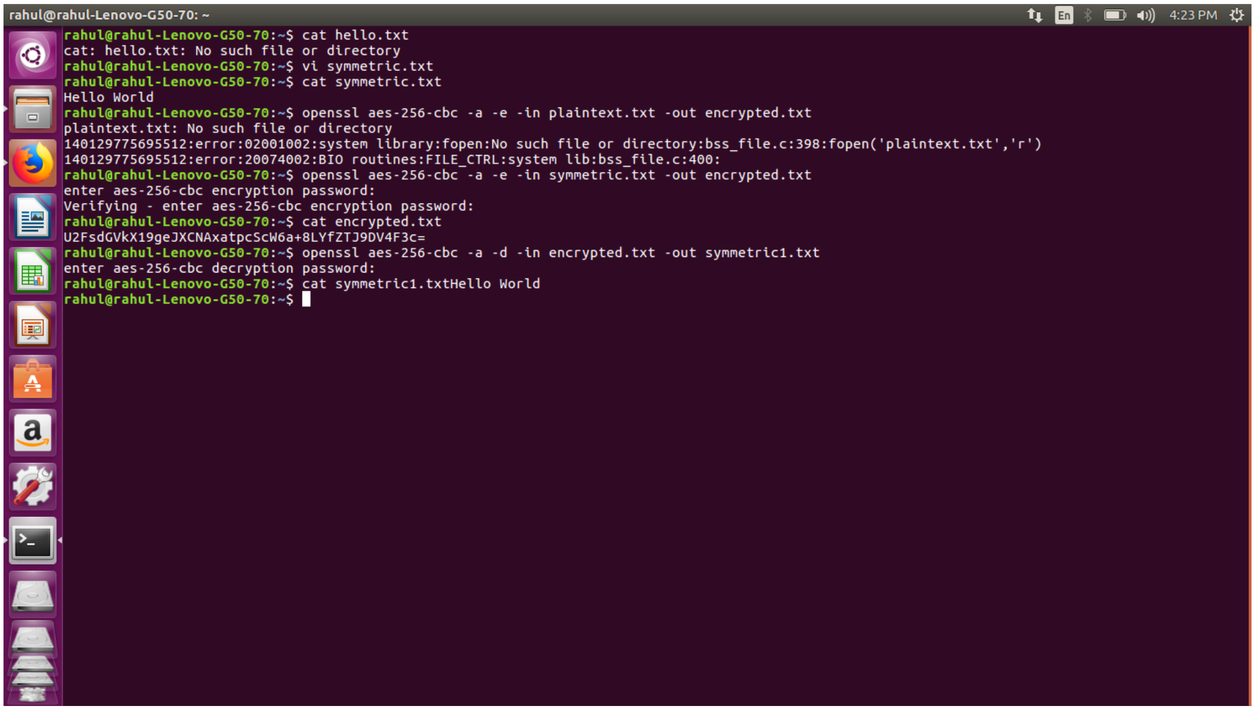
Symmetric Encryption are algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. The keys may be identical or there may be a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to public-key encryption (also known as asymmetric key encryption).

<u>To encrypt:</u>

$openssl aes-256-cbc -a -e -in plaintext.txt -out encrypted.txt

<u>To decrypt:</u>

$openssl aes-256-cbc -a -d -in encrypted.txt -out plaintext.txt



```
rahul@rahul-Lenovo-G50-70:~$ cat hello.txt
cat: hello.txt: No such file or directory
rahul@rahul-Lenovo-G50-70:~$ vi symmetric.txt
rahul@rahul-Lenovo-G50-70:~$ cat symmetric.txt
Hello World
rahul@rahul-Lenovo-G50-70:~$ openssl aes-256-cbc -a -e -in plaintext.txt -out encrypted.txt
plaintext.txt: No such file or directory
140129775695512:error:02001002:system library:fopen:No such file or directory:bss_file.c:398:fopen('plaintext.txt','r')
140129775695512:error:20074002:BIO routines:FILE_CTRL:system lib:bss_file.c:400:
rahul@rahul-Lenovo-G50-70:~$ openssl aes-256-cbc -a -e -in symmetric.txt -out encrypted.txt
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
rahul@rahul-Lenovo-G50-70:~$ cat encrypted.txt
U2FsdGVkX19geJXCNAxatpcScW6a+8LYfZTJ9DV4F3c=
rahul@rahul-Lenovo-G50-70:~$ openssl aes-256-cbc -a -d -in encrypted.txt -out symmetric1.txt
enter aes-256-cbc decryption password:
rahul@rahul-Lenovo-G50-70:~$ cat symmetric1.txtHello World
rahul@rahul-Lenovo-G50-70:~$ 
```

Asymmetric Encryption

Asymmetric cryptography, also known as public key cryptography, uses public and private keys to encrypt and decrypt data. The keys are simply large numbers that have been paired together but are not identical (asymmetric). One key in the pair can be shared with everyone; it is called the public_key. The other key in the pair is kept secret; it is called the private_key. Either of the keys can be used to encrypt a message; the opposite key from the one used to encrypt the message is used for decryption.

For Asymmetric encryption we must first generate private key and extract the public key.

$openssl genrsa -aes256 -out private.key 8912

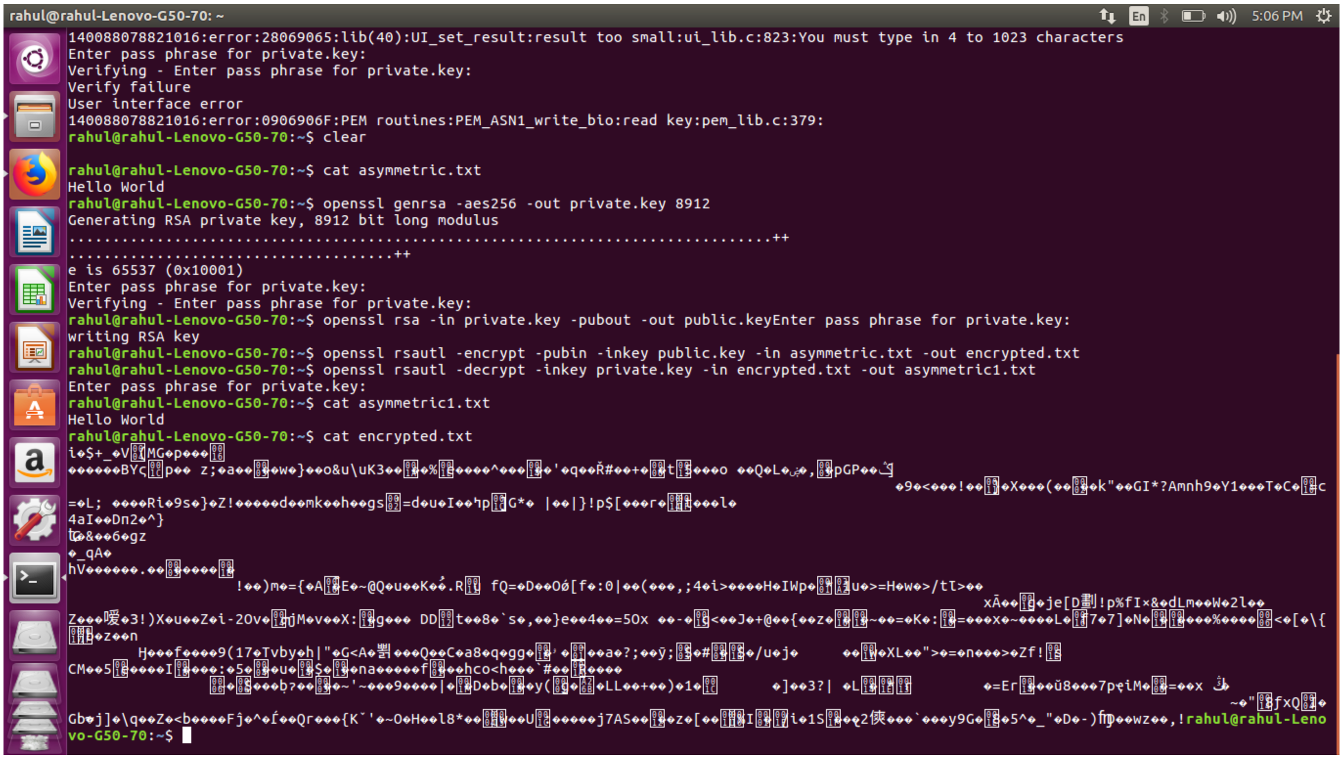
$openssl rsa -in private.key -pubout -out public.key

To encrypt:

openssl rsautl -encrypt -pubin -inkey public.key -in plaintext.txt -out encrypted.txt

To decrypt:

openssl rsautl -decrypt -inkey private.key -in encrypted.txt -out plaintext.txt



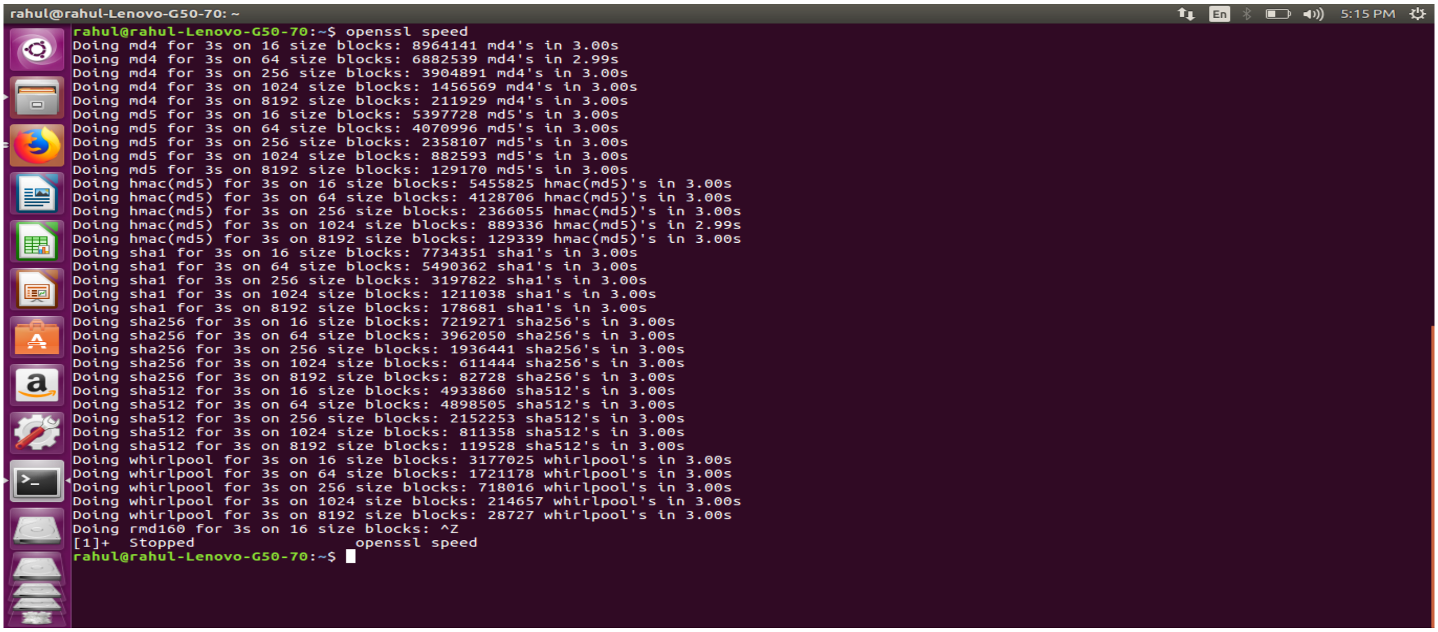
```
140088078821016:error:28069065:lib(40):UI_set_result:result too small:ui_lib.c:823:You must type in 4 to 1023 characters
Enter pass phrase for private.key:
Verifying - Enter pass phrase for private.key:
Verify failure
User interface error
140088078821016:error:0906906F:PEM routines:PEM_ASN1_write_bio:read key:pem_lib.c:379:
rahul@rahul-Lenovo-G50-70:~$ clear

rahul@rahul-Lenovo-G50-70:~$ cat asymmetric.txt
Hello World
rahul@rahul-Lenovo-G50-70:~$ openssl genrsa -aes256 -out private.key 8912
Generating RSA private key, 8912 bit long modulus
.................................................................................++
.....................................++
e is 65537 (0x10001)
Enter pass phrase for private.key:
Verifying - Enter pass phrase for private.key:
rahul@rahul-Lenovo-G50-70:~$ openssl rsa -in private.key -pubout -out public.keyEnter pass phrase for private.key:
writing RSA key
rahul@rahul-Lenovo-G50-70:~$ openssl rsautl -encrypt -pubin -inkey public.key -in asymmetric.txt -out encrypted.txt
rahul@rahul-Lenovo-G50-70:~$ openssl rsautl -decrypt -inkey private.key -in encrypted.txt -out asymmetric1.txt
Enter pass phrase for private.key:
rahul@rahul-Lenovo-G50-70:~$ cat asymmetric1.txt
Hello World
rahul@rahul-Lenovo-G50-70:~$ cat encrypted.txt
i�$+_�V□MG�p���□
������BYç□p�� z;�a��□�w�}��o&u\uK3��□�%□����^���□�'�q��Ř#��+�□t□����o ��Q�L�,□pGP��
                                                                                     �9�<���!��□�X���(��□�k"��GI*?Amnh9�Y1���T�C�□c
=�L; ����Ri�9s�}�Z!�����d��mk��h��gs□=d�u�I��ħp□G*� |��|}!p$[���r�□���l�
4aI��Dn2�^}
ŧ�&��6�gz
�_qA�
hV������.��□����□
                  !��)m�={�A□E�~@Q�u��K��.R□ fQ=�D��Oø[f�:0|��(���,;4�i>����H�IWp�□□u�>=H�w�>/tI>��
                                                                                            xĀ��□�je[D□!p%fI×&�dLm��W�2l��
Z���暖�3!)X�u��Z�i-2Ov�□ĵM�v��X:□g��� DD□t��8�`s�,��}e��4��=5Ox ��-�□<��J�+@��{��z�□□~��=�K�:□=���x�~����L�□7�7]�N�□□���%����□<�[�\{
□�z��n
       Ḩ���f����9(17�Tvby�h|"�G<A�□���Q��C�a8�q�gg�□ �□��a�?;��ÿ;□�#□□�/u�j�     ��□�XL��">�=�n���>�Zf!□
CM��5□����I□���:�5�□�u�□$�□�na�����f□��hco<h���`#��□����
                 □�□���b?��□�~'~���9����|�□D�b�□�y(□�□�LL��+��)�1�□      �]��3?| �L□□□        �=Er□��ŭ8���7pęiM�□=��x
                                                                                                              ~�"□fxQ□�
Gb�j]�\q��Z�<b����Fĵ�^�Í��Qr���{Kˇ'�~O�H��l8*��□��U□�����j7AS��□�z�[��□I□□i�1S□�ę2□���`���y9G�□�5^�_"�D�-)□��wz��,!rahul@rahul-Leno
vo-G50-70:~$
```

To check the speed of system using Openssl benchmarking option

OpenSSL comes with an in-built benchmarking option called ‘speed’. It tells us how many operations it can perform in a given time.

$openssl speed



```
rahul@rahul-Lenovo-G50-70:~$ openssl speed
Doing md4 for 3s on 16 size blocks: 8964141 md4's in 3.00s
Doing md4 for 3s on 64 size blocks: 6882539 md4's in 2.99s
Doing md4 for 3s on 256 size blocks: 3904891 md4's in 3.00s
Doing md4 for 3s on 1024 size blocks: 1456569 md4's in 3.00s
Doing md4 for 3s on 8192 size blocks: 211929 md4's in 3.00s
Doing md5 for 3s on 16 size blocks: 5397728 md5's in 3.00s
Doing md5 for 3s on 64 size blocks: 4070996 md5's in 3.00s
Doing md5 for 3s on 256 size blocks: 2358107 md5's in 3.00s
Doing md5 for 3s on 1024 size blocks: 882593 md5's in 3.00s
Doing md5 for 3s on 8192 size blocks: 129170 md5's in 3.00s
Doing hmac(md5) for 3s on 16 size blocks: 5455825 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 64 size blocks: 4128706 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 256 size blocks: 2366055 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 1024 size blocks: 889336 hmac(md5)'s in 2.99s
Doing hmac(md5) for 3s on 8192 size blocks: 129339 hmac(md5)'s in 3.00s
Doing sha1 for 3s on 16 size blocks: 7734351 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 5490362 sha1's in 3.00s
Doing sha1 for 3s on 256 size blocks: 3197822 sha1's in 3.00s
Doing sha1 for 3s on 1024 size blocks: 1211038 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 178681 sha1's in 3.00s
Doing sha256 for 3s on 16 size blocks: 7219271 sha256's in 3.00s
Doing sha256 for 3s on 64 size blocks: 3962050 sha256's in 3.00s
Doing sha256 for 3s on 256 size blocks: 1936441 sha256's in 3.00s
Doing sha256 for 3s on 1024 size blocks: 611444 sha256's in 3.00s
Doing sha256 for 3s on 8192 size blocks: 82728 sha256's in 3.00s
Doing sha512 for 3s on 16 size blocks: 4933860 sha512's in 3.00s
Doing sha512 for 3s on 64 size blocks: 4898505 sha512's in 3.00s
Doing sha512 for 3s on 256 size blocks: 2152253 sha512's in 3.00s
Doing sha512 for 3s on 1024 size blocks: 811358 sha512's in 3.00s
Doing sha512 for 3s on 8192 size blocks: 119528 sha512's in 3.00s
Doing whirlpool for 3s on 16 size blocks: 3177025 whirlpool's in 3.00s
Doing whirlpool for 3s on 64 size blocks: 1721178 whirlpool's in 3.00s
Doing whirlpool for 3s on 256 size blocks: 718016 whirlpool's in 3.00s
Doing whirlpool for 3s on 1024 size blocks: 214657 whirlpool's in 3.00s
Doing whirlpool for 3s on 8192 size blocks: 28727 whirlpool's in 3.00s
Doing rmd160 for 3s on 16 size blocks: ^Z
[1]+  Stopped                 openssl speed
rahul@rahul-Lenovo-G50-70:~$ 
```

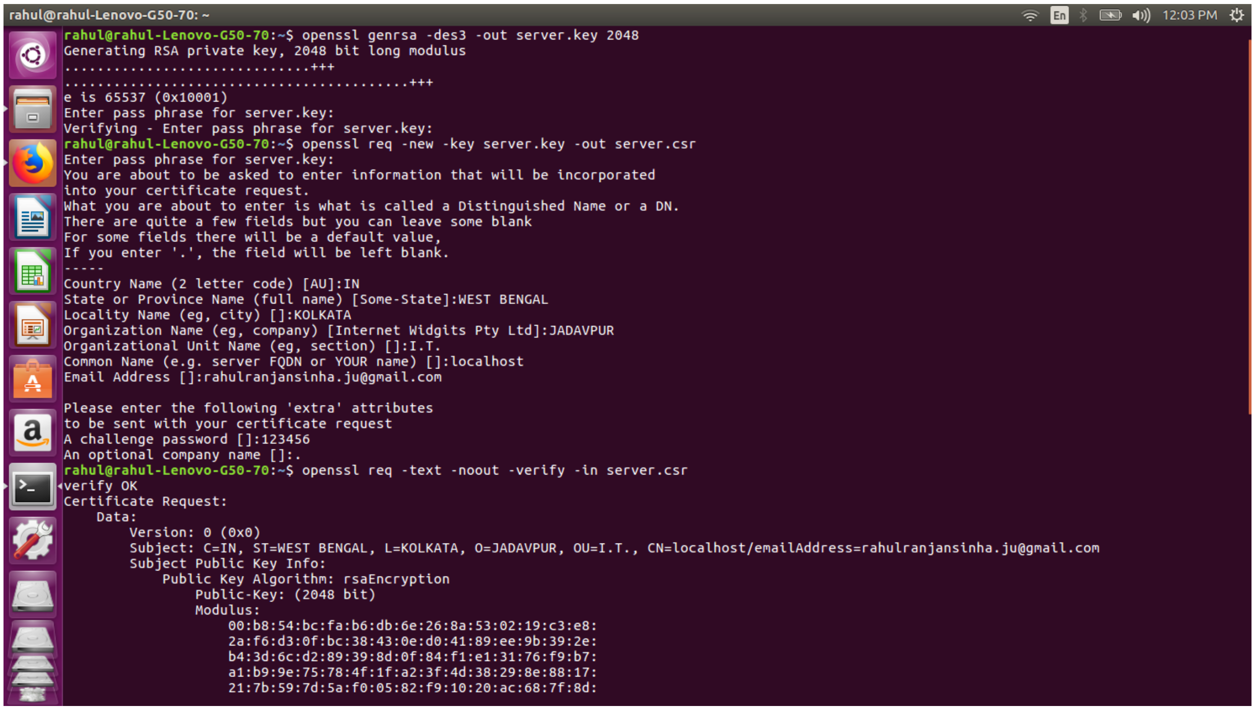
Generate CSR (Certificate Signing Request)

In public key infrastructure (PKI) systems, a **certificate signing request** (also **CSR** or **certification request**) is a message sent from an applicant to a certificate authority in order to apply for a digital identity certificate. It usually contains the public key for which the certificate should be issued, identifying information (such as a domain name) and integrity protection (e.g., a digital signature).

1. **openssl genrsa -des3 -out server.key 2048**
2. **openssl req -new -key server.key -out server.csr**

The first command will generate a 2048 bit (recommended) RSA private key. After running the command it will ask for the passphrase. If we want to create a key without the passphrase we can remove the **(-des3)** from the command.

The second command generates a **CSR** (Certificate Signing Request). The CA will use the .csr file and issue the certificate, but in our case, we can use this .csr file to create our self-signed certificate. Once we run the command, it will prompt us to enter our country, company name, etc.

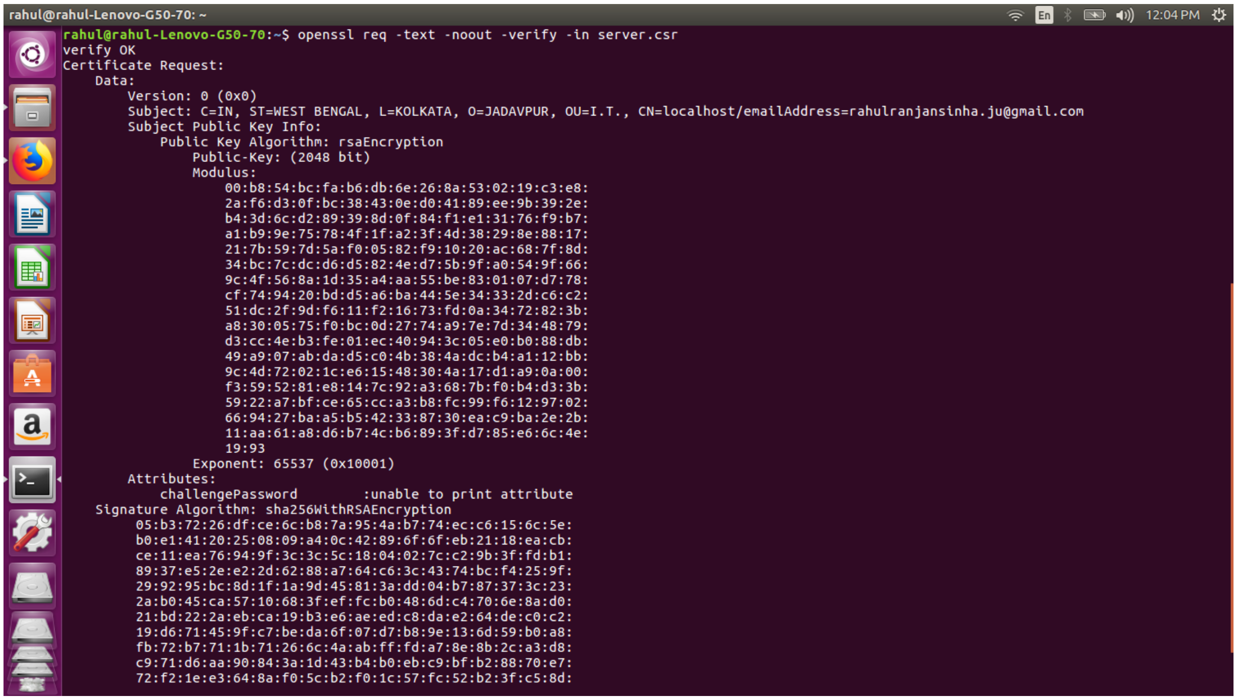


```
rahul@rahul-Lenovo-G50-70:~$ openssl genrsa -des3 -out server.key 2048
Generating RSA private key, 2048 bit long modulus
..............................+++
.........................................+++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
rahul@rahul-Lenovo-G50-70:~$ openssl req -new -key server.key -out server.csr
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]:IN
State or Province Name (full name) [Some-State]:WEST BENGAL
Locality Name (eg, city) []:KOLKATA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:JADAVPUR
Organizational Unit Name (eg, section) []:I.T.
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:rahulranjansinha.ju@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:.
rahul@rahul-Lenovo-G50-70:~$ openssl req -text -noout -verify -in server.csr
verify OK
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=IN, ST=WEST BENGAL, L=KOLKATA, O=JADAVPUR, OU=I.T., CN=localhost/emailAddress=rahulranjansinha.ju@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b8:54:bc:fa:b6:db:6e:26:8a:53:02:19:c3:e8:
                    2a:f6:d3:0f:bc:38:43:0e:d0:41:89:ee:9b:39:2e:
                    b4:3d:6c:d2:89:39:8d:0f:84:f1:e1:31:76:f9:b7:
                    a1:b9:9e:75:78:4f:1f:a2:3f:4d:38:29:8e:88:17:
                    21:7b:59:7d:5a:f0:05:82:f9:10:20:ac:68:7f:8d:
```

Checking CSR (Certificate Signing Request)

openssl req -text -noout -verify -in CSR.csr



```
rahul@rahul-Lenovo-G50-70:~$ openssl req -text -noout -verify -in server.csr
verify OK
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=IN, ST=WEST BENGAL, L=KOLKATA, O=JADAVPUR, OU=I.T., CN=localhost/emailAddress=rahulranjansinha.ju@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b8:54:bc:fa:b6:db:6e:26:8a:53:02:19:c3:e8:
                    2a:f6:d3:0f:bc:38:43:0e:d0:41:89:ee:9b:39:2e:
                    b4:3d:6c:d2:89:39:8d:0f:84:f1:e1:31:76:f9:b7:
                    a1:b9:9e:75:78:4f:1f:a2:3f:4d:38:29:8e:88:17:
                    21:7b:59:7d:5a:f0:05:82:f9:10:20:ac:68:7f:8d:
                    34:bc:7c:dc:d6:d5:82:4e:d7:5b:9f:a0:54:9f:66:
                    9c:4f:56:8a:1d:35:a4:aa:55:be:83:01:07:d7:78:
                    cf:74:94:20:bd:d5:a6:ba:44:5e:34:33:2d:c6:c2:
                    51:dc:2f:9d:f6:11:f2:16:73:fd:0a:34:72:82:3b:
                    a8:30:05:75:f0:bc:0d:27:74:a9:7e:7d:34:48:79:
                    d3:cc:4e:b3:fe:01:ec:40:94:3c:05:e0:b0:88:db:
                    49:a9:07:ab:da:d5:c0:4b:38:4a:dc:b4:a1:12:bb:
                    9c:4d:72:02:1c:e6:15:48:30:4a:17:d1:a9:0a:00:
                    f3:59:52:81:e8:14:7c:92:a3:68:7b:f0:b4:d3:3b:
                    59:22:a7:bf:ce:65:cc:a3:b8:fc:99:f6:12:97:02:
                    66:94:27:ba:a5:b5:42:33:87:30:ea:c9:ba:2e:2b:
                    11:aa:61:a8:d6:b7:4c:b6:89:3f:d7:85:e6:6c:4e:
                    19:93
                Exponent: 65537 (0x10001)
        Attributes:
            challengePassword        :unable to print attribute
    Signature Algorithm: sha256WithRSAEncryption
         05:b3:72:26:df:ce:6c:b8:7a:95:4a:b7:74:ec:c6:15:6c:5e:
         b0:e1:41:20:25:08:09:a4:0c:42:89:6f:6f:eb:21:18:ea:cb:
         ce:11:ea:76:94:9f:3c:3c:5c:18:04:02:7c:c2:9b:3f:fd:b1:
         89:37:e5:2e:e2:2d:62:88:a7:64:c6:3c:43:74:bc:f4:25:9f:
         29:92:95:bc:8d:1f:1a:9d:45:81:3a:dd:04:b7:87:37:3c:23:
         2a:b0:45:ca:57:10:68:3f:ef:fc:b0:48:6d:c4:70:6e:8a:d0:
         21:bd:22:2a:eb:ca:19:b3:e6:ae:ed:c8:da:e2:64:de:c0:c2:
         19:d6:71:45:9f:c7:be:da:6f:07:d7:b8:9e:13:6d:59:b0:a8:
         fb:72:b7:71:1b:71:26:6c:4a:ab:ff:fd:a7:8e:8b:2c:a3:d8:
         c9:71:d6:aa:90:84:3a:1d:43:b4:b0:eb:c9:bf:b2:88:70:e7:
         72:f2:1e:e3:64:8a:f0:5c:b2:f0:1c:57:fc:52:b2:3f:c5:8d:
```

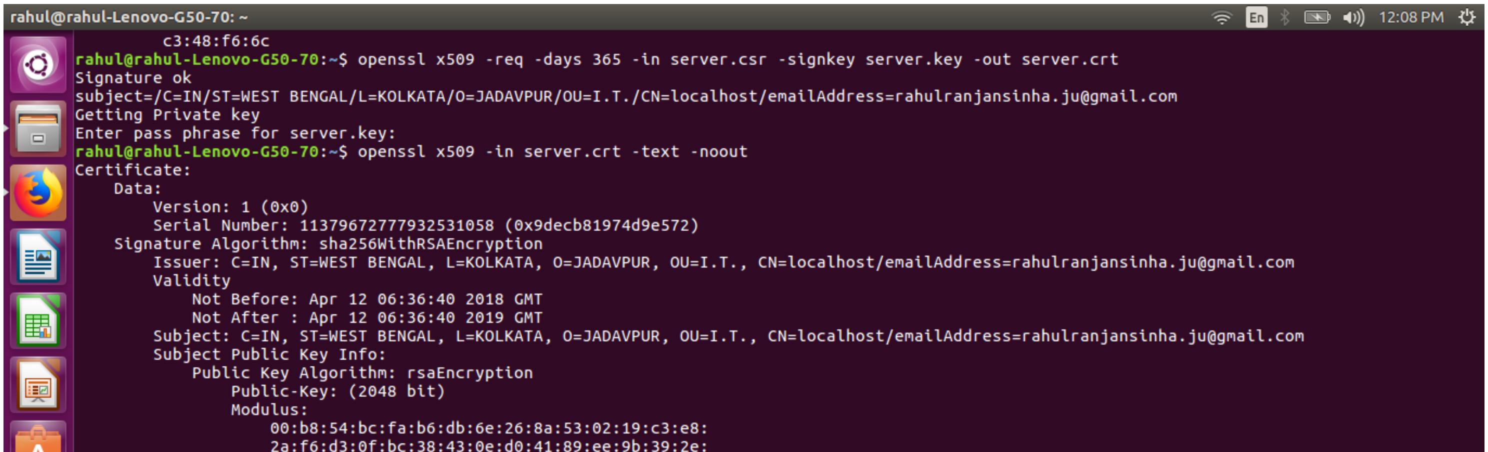
Create a Self-Signed SSL Certificate Using OpenSSL

To create this secure connection an SSL Certificate is used, which is installed on the web server. So, an SSL Certificate is a bit of code on your web server that provides security for your online communications. SSL certificates also contain identification information (i.e your organizational information).

A self-signed certificate is a certificate that is signed by its own creator rather than a trusted authority. Self-signed certificates are less trustworthy since any attacker can create a self-signed certificate and launch a **man in the middle attack.**

openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

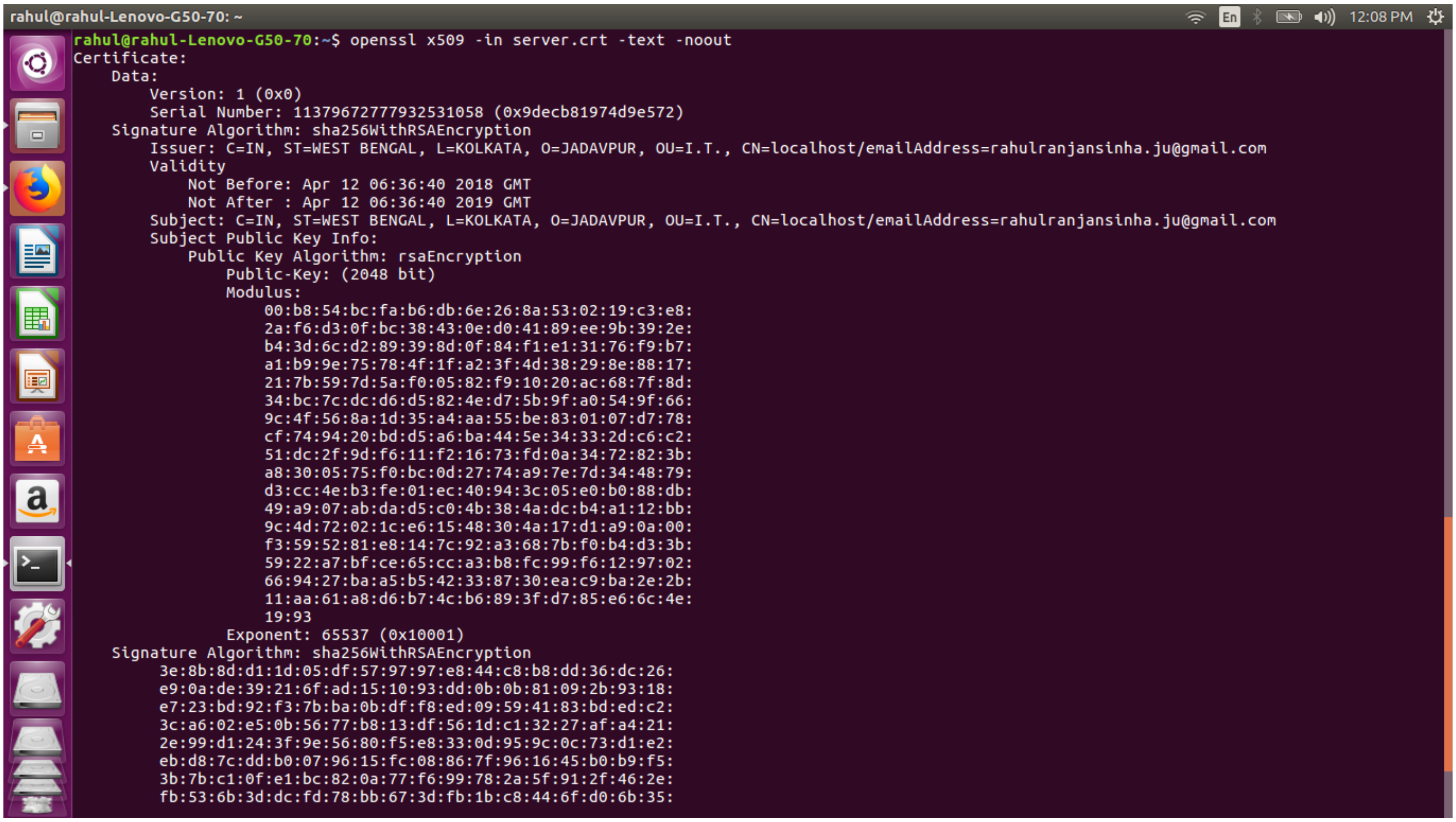
It creates the self-signed x509 certificate with 365 days validity, suitable for use on a web server.



```
          c3:48:f6:6c
rahul@rahul-Lenovo-G50-70:~$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
Signature ok
subject=/C=IN/ST=WEST BENGAL/L=KOLKATA/O=JADAVPUR/OU=I.T./CN=localhost/emailAddress=rahulranjansinha.ju@gmail.com
Getting Private key
Enter pass phrase for server.key:
rahul@rahul-Lenovo-G50-70:~$ openssl x509 -in server.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 11379672777932531058 (0x9decb81974d9e572)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=IN, ST=WEST BENGAL, L=KOLKATA, O=JADAVPUR, OU=I.T., CN=localhost/emailAddress=rahulranjansinha.ju@gmail.com
        Validity
            Not Before: Apr 12 06:36:40 2018 GMT
            Not After : Apr 12 06:36:40 2019 GMT
        Subject: C=IN, ST=WEST BENGAL, L=KOLKATA, O=JADAVPUR, OU=I.T., CN=localhost/emailAddress=rahulranjansinha.ju@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b8:54:bc:fa:b6:db:6e:26:8a:53:02:19:c3:e8:
                    2a:f6:d3:0f:bc:38:43:0e:d0:41:89:ee:9b:39:2e:
```

Checking the Certificate

openssl x509 -in certificate.crt -text -noout



```
rahul@rahul-Lenovo-G50-70:~$ openssl x509 -in server.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 11379672777932531058 (0x9decb81974d9e572)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=IN, ST=WEST BENGAL, L=KOLKATA, O=JADAVPUR, OU=I.T., CN=localhost/emailAddress=rahulranjansinha.ju@gmail.com
        Validity
            Not Before: Apr 12 06:36:40 2018 GMT
            Not After : Apr 12 06:36:40 2019 GMT
        Subject: C=IN, ST=WEST BENGAL, L=KOLKATA, O=JADAVPUR, OU=I.T., CN=localhost/emailAddress=rahulranjansinha.ju@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b8:54:bc:fa:b6:db:6e:26:8a:53:02:19:c3:e8:
                    2a:f6:d3:0f:bc:38:43:0e:d0:41:89:ee:9b:39:2e:
                    b4:3d:6c:d2:89:39:8d:0f:84:f1:e1:31:76:f9:b7:
                    a1:b9:9e:75:78:4f:1f:a2:3f:4d:38:29:8e:88:17:
                    21:7b:59:7d:5a:f0:05:82:f9:10:20:ac:68:7f:8d:
                    34:bc:7c:dc:d6:d5:82:4e:d7:5b:9f:a0:54:9f:66:
                    9c:4f:56:8a:1d:35:a4:aa:55:be:83:01:07:d7:78:
                    cf:74:94:20:bd:d5:a6:ba:44:5e:34:33:2d:c6:c2:
                    51:dc:2f:9d:f6:11:f2:16:73:fd:0a:34:72:82:3b:
                    a8:30:05:75:f0:bc:0d:27:74:a9:7e:7d:34:48:79:
                    d3:cc:4e:b3:fe:01:ec:40:94:3c:05:e0:b0:88:db:
                    49:a9:07:ab:da:d5:c0:4b:38:4a:dc:b4:a1:12:bb:
                    9c:4d:72:02:1c:e6:15:48:30:4a:17:d1:a9:0a:00:
                    f3:59:52:81:e8:14:7c:92:a3:68:7b:f0:b4:d3:3b:
                    59:22:a7:bf:ce:65:cc:a3:b8:fc:99:f6:12:97:02:
                    66:94:27:ba:a5:b5:42:33:87:30:ea:c9:ba:2e:2b:
                    11:aa:61:a8:d6:b7:4c:b6:89:3f:d7:85:e6:6c:4e:
                    19:93
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         3e:8b:8d:d1:1d:05:df:57:97:97:e8:44:c8:b8:dd:36:dc:26:
         e9:0a:de:39:21:6f:ad:15:10:93:dd:0b:0b:81:09:2b:93:18:
         e7:23:bd:92:f3:7b:ba:0b:df:f8:ed:09:59:41:83:bd:ed:c2:
         3c:a6:02:e5:0b:56:77:b8:13:df:56:1d:c1:32:27:af:a4:21:
         2e:99:d1:24:3f:9e:56:80:f5:e8:33:0d:95:9c:0c:73:d1:e2:
         eb:d8:7c:dd:b0:07:96:15:fc:08:86:7f:96:16:45:b0:b9:f5:
         3b:7b:c1:0f:e1:bc:82:0a:77:f6:99:78:2a:5f:91:2f:46:2e:
         fb:53:6b:3d:dc:fd:78:bb:67:3d:fb:1b:c8:44:6f:d0:6b:35:
```

Create https localhost (ssl) on ubuntu 16.04

Step 1: Generating the certificate

First, let's create a place to store the file.

```
mkdir ~/certificates
```

```
cd ~/certificates
```

Generate CSR and private key.

```
openssl req -x509 -newkey rsa:4096 -keyout apache.key -out apache.crt -days 365 -nodes
```

It will ask for information for the certificate request. Complete with the appropriate information.

```
Country Name (2 letter code) [AU]: IN

State or Province Name (full name) [Some-State]: WEST BENGAL

Locality Name (eg, city) []: KOLKATA

Organization Name (eg, company) [My Company]: JADAVPUR

Organizational Unit Name (eg, section) []: I.T.

Common Name (e.g. server FQDN or YOUR name) []: localhost

Email Address []:rahulranjansinha.ju@gmail.com
```

Now, move the certificate to Apache configuration folder.

```
mkdir /etc/apache2/ssl
```

```
mv ~/certificates/* /etc/apache2/ssl/.
```

The certificate is ready! Next, we will prepare Apache to work with the certificate.

Step 2: Firewall configuration

We have to make sure TCP port 443 is open. This port is used in SSL connections instead of port 80. In this tutorial, we will be using UFW.

Make sure UFW is enabled.

```
sudo ufw enable
```

Now allow the predefined Apache settings for the firewall.

```
sudo ufw allow 'Apache Full'
```

By typing "sudo ufw status", we can see a list of the current rules. Our configuration should resemble this:

```
To                  Action     From
--                  ------     ----
Apache Full         ALLOW      Anywhere
OpenSSH             ALLOW      Anywhere
Apache Full (v6)    ALLOW      Anywhere (v6)
OpenSSH (v6)        ALLOW      Anywhere (v6)
```

We should also allow OpenSSH here for future connections.

```
sudo ufw allow 'OpenSSH'
```

Step 3: Apache virtual host configuration

Navigate to the default Apache site config directory.

```
sudo nano /etc/apache2/sites-available/default-ssl.conf
```

This file tells the server where to look for the SSL certificate. With the comments removed, it should look like the following config.

```
<IfModule mod_ssl.c>
  <VirtualHost _default_:443>
   ServerAdmin webmaster@localhost

   DocumentRoot /var/www/html

   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined

   SSLEngine on

   SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
   SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
   <FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
   </FilesMatch>
   <Directory /usr/lib/cgi-bin>
    SSLOptions +StdEnvVars
   </Directory>

 </VirtualHost>
</IfModule>
```

Edit this line:

```
ServerAdmin email@example.net
```

Add this right below the ServerAdmin line:

```
ServerName ADD_YOUR_IP_OR_DOMAIN_NAME_HERE
```

Now, edit these lines with our certificate location:

```
SSLCertificateFile    /etc/apache2/ssl/apache.crt
SSLCertificateKeyFile /etc/apache2/ssl/apache.key
```

Our final file should resemble this:

```
<IfModule mod_ssl.c>
 <VirtualHost _default_:443>
  ServerAdmin email@example.net
  ServerName 203.0.113.122

  DocumentRoot /var/www/html

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

  SSLEngine on
  SSLCertificateFile    /etc/apache2/ssl/apache.crt

  SSLCertificateKeyFile /etc/apache2/ssl/apache.key


  <FilesMatch "\.(cgi|shtml|phtml|php)$">

   SSLOptions +StdEnvVars

  </FilesMatch>

  <Directory /usr/lib/cgi-bin>

   SSLOptions +StdEnvVars

  </Directory>


 </VirtualHost>

</IfModule>
```

Save and close the file.

Step 4: Enabling Apache SSL module

Enable the SSL module by typing:

```
sudo a2enmod ssl
```

Now enable the site we have just edited:

```
sudo a2ensite default-ssl.conf
```

Restart Apache:

```
sudo service apache2 restart
```

Let's access the new secure website! Open it in our browser (make sure you type https://).

https://localhost

Your browser will warn you that the certificate is invalid, as we expected. This happens because the certificate is not signed. Follow the steps offered by your browser to proceed to your site.

Step 5: Redirect all HTTP traffic to HTTPS (Optional)

Open the Apache default virtual host file:

```
nano /etc/apache2/sites-available/000-default.conf
```

Add this line inside the <VirtualHost *:80> tag:

```
Redirect / https://localhost/
```

Reload Apache configuration:

```
sudo service apache2 reload
```

All website traffic will now automatically redirect to HTTPS.