Complete Certificate Generation, Keystore & Truststore Process (with Commands & Explanations

Keystore:

A keystore is a secure storage that holds private keys and their associated certificates.

- Used by servers to prove their identity (like HTTPS servers)
- Contains private key (secret) and public cert (shared)
- Format: `.jks`, `.p12` (PKCS#12), `.pfx`

`Truststore:`

A truststore is a secure storage that holds public certificates of other systems you trust.

- Used by clients to validate server certificates
- Contains only public certs (no private keys)
- Format: .jks

Overview of the Flow:

1. Generate a Private Key
2. Create a Certificate Signing Request (CSR)
3. Generate Self-Signed Certificate
4. Bundle Private Key + Cert into Keystore (.p12)
5. Create Truststore & Import Certificate
6. Verify Contents of Keystore & Truststore

Step 1: Generate a Private Key

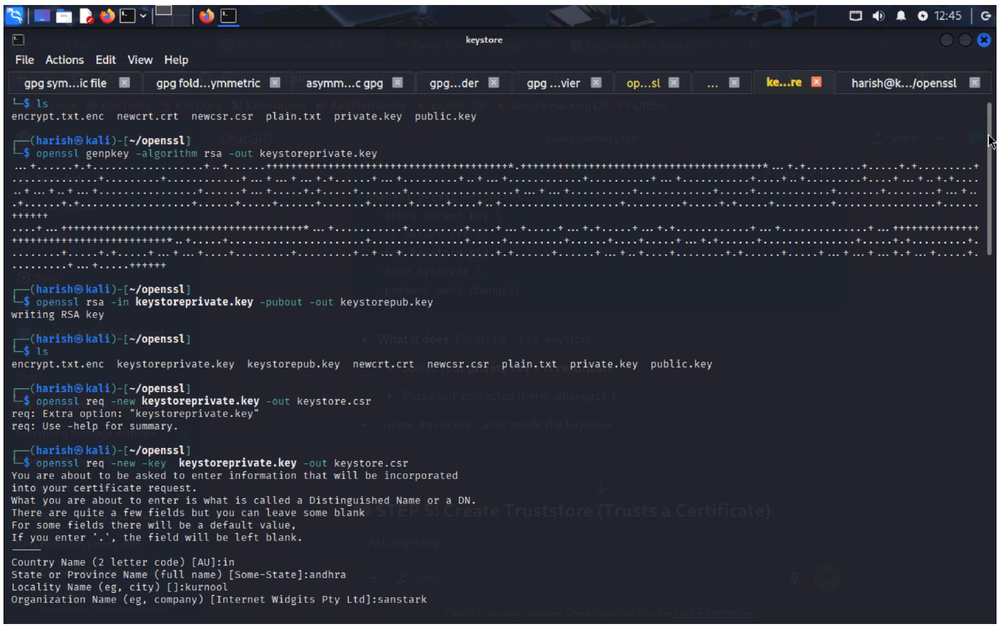
Command:openssl genpkey -algorithm RSA -out keystoreprivate.key

Step 2: Generate a Certificate Signing Request (CSR)

Command:openssl req -new -key server.key -out keystore.csr

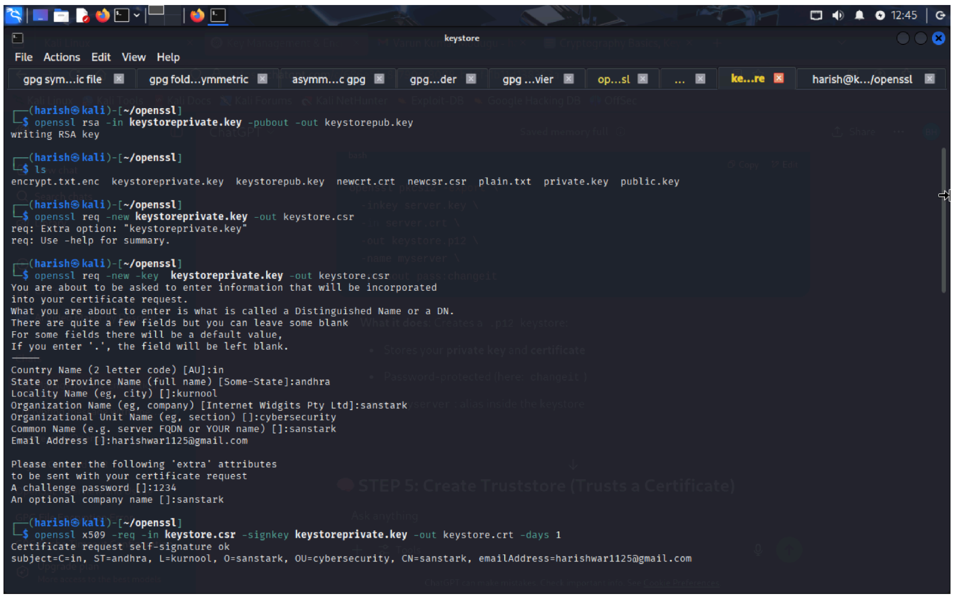
Step 3: Self-Sign the Certificate (own CA)

Command: openssl x509 -req -in server.csr -signkey server.key -out keystore.crt -days 1



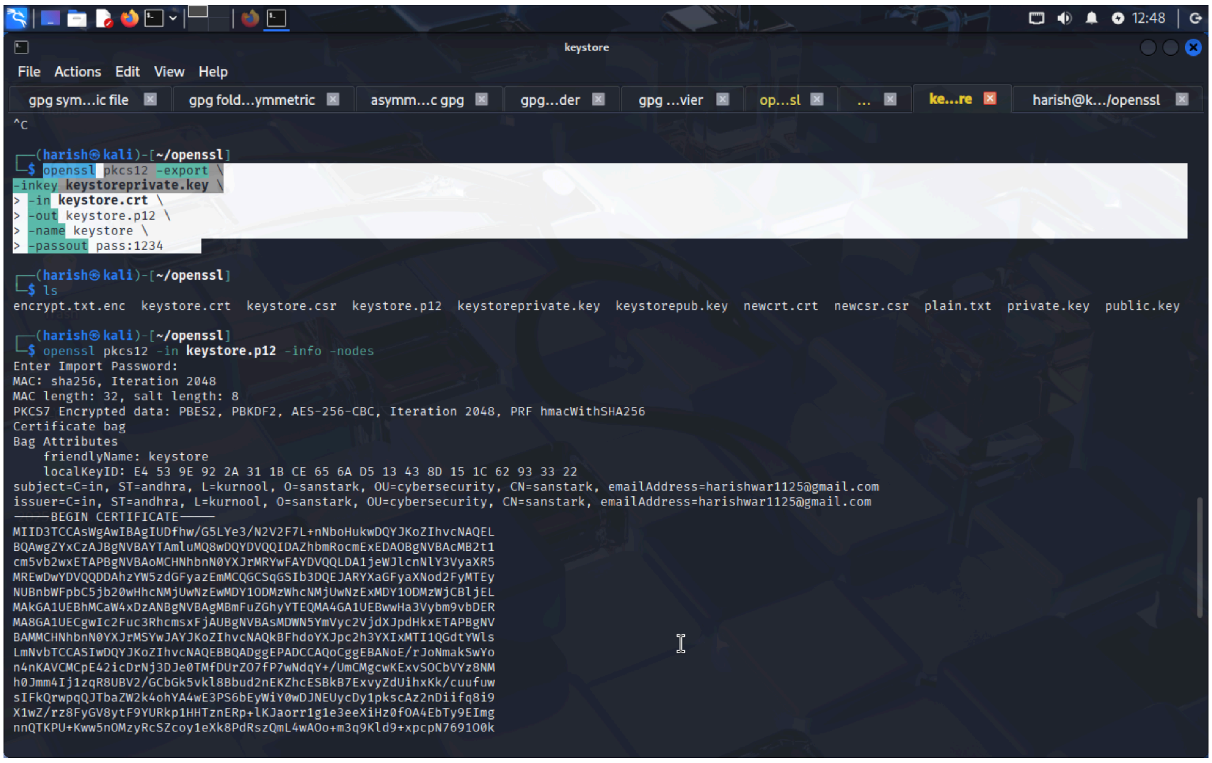
```
└─$ ls
encrypt.txt.enc  newcrt.crt  newcsr.csr  plain.txt  private.key  public.key

┌──(harish㉿kali)-[~/openssl]
└─$ openssl genpkey -algorithm rsa -out keystoreprivate.key
...+......+.+..................+..+......++++++++++++++++++++++++++++++++++++++++*.++++++++++++++++++++++++++++++++++++++++*...+.+.........+.....+.+.........+
..............+.........+...........+...+...+...+.+......+...+.........+..+...+..............+...........+...+...........+....+.+.........+....+...+..+.+....
..+...+..+...+...............+.....+...+.....+.+.....+.......+.........+...+...+...........+...........+.....+...........+.......+.......+...+..
.+......+.+................+.......+.....+.....+.......+.....+.....+....+.+..................+.........+.....+.+......+.........+.......+.........+......
++++++
....+...++++++++++++++++++++++++++++++++++++++++*...+..........+.......+....+...+.....+...+.+.....+...+.+...........+...+............+...++++++++++++++
++++++++++++++++++++++++++*..+.....+.....................+..............+.........+......+......+.....+...+.+.......+..............+.....+.+.........+.
.......+.....+.+.....+...+...+....+.........+.........+..+...+...........+.+.......+.....+...+..+....+.........+.+......+.....+...+...+...+.+...+.....+.
.........+...+.....++++++

┌──(harish㉿kali)-[~/openssl]
└─$ openssl rsa -in keystoreprivate.key -pubout -out keystorepub.key
writing RSA key

┌──(harish㉿kali)-[~/openssl]
└─$ ls
encrypt.txt.enc  keystoreprivate.key  keystorepub.key  newcrt.crt  newcsr.csr  plain.txt  private.key  public.key

┌──(harish㉿kali)-[~/openssl]
└─$ openssl req -new keystoreprivate.key -out keystore.csr
req: Extra option: "keystoreprivate.key"
req: Use -help for summary.

┌──(harish㉿kali)-[~/openssl]
└─$ openssl req -new -key  keystoreprivate.key -out keystore.csr
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]:andhra
Locality Name (eg, city) []:kurnool
Organization Name (eg, company) [Internet Widgits Pty Ltd]:sanstark
```

Fig: keys,csr



```
┌──(harish㉿kali)-[~/openssl]
└─$ openssl rsa -in keystoreprivate.key -pubout -out keystorepub.key
writing RSA key

┌──(harish㉿kali)-[~/openssl]
└─$ ls
encrypt.txt.enc  keystoreprivate.key  keystorepub.key  newcrt.crt  newcsr.csr  plain.txt  private.key  public.key

┌──(harish㉿kali)-[~/openssl]
└─$ openssl req -new keystoreprivate.key -out keystore.csr
req: Extra option: "keystoreprivate.key"
req: Use -help for summary.

┌──(harish㉿kali)-[~/openssl]
└─$ openssl req -new -key  keystoreprivate.key -out keystore.csr
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]:andhra
Locality Name (eg, city) []:kurnool
Organization Name (eg, company) [Internet Widgits Pty Ltd]:sanstark
Organizational Unit Name (eg, section) []:cybersecurity
Common Name (e.g. server FQDN or YOUR name) []:sanstark
Email Address []:harishwar1125@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []:sanstark

┌──(harish㉿kali)-[~/openssl]
└─$ openssl x509 -req -in keystore.csr -signkey keystoreprivate.key -out keystore.crt -days 1
Certificate request self-signature ok
subject=C=in, ST=andhra, L=kurnool, O=sanstark, OU=cybersecurity, CN=sanstark, emailAddress=harishwar1125@gmail.com
```

Fig:csr-info,cert

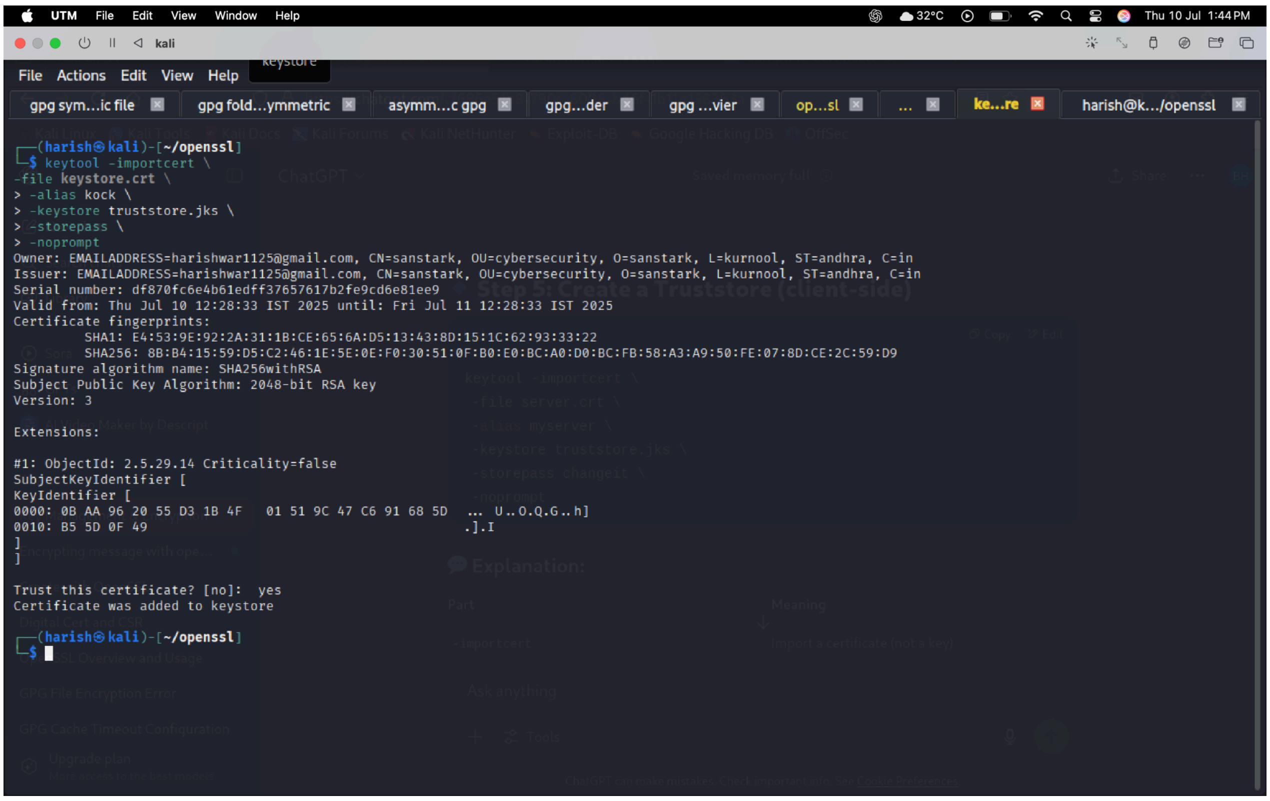


```
^C

┌──(harish㉿kali)-[~/openssl]
└─$ openssl pkcs12 -export \
-inkey keystoreprivate.key \
> -in keystore.crt \
> -out keystore.p12 \
> -name keystore \
> -passout pass:1234

┌──(harish㉿kali)-[~/openssl]
└─$ ls
encrypt.txt.enc  keystore.crt  keystore.csr  keystore.p12  keystoreprivate.key  keystorepub.key  newcrt.crt  newcsr.csr  plain.txt  private.key  public.key

┌──(harish㉿kali)-[~/openssl]
└─$ openssl pkcs12 -in keystore.p12 -info -nodes
Enter Import Password:
MAC: sha256, Iteration 2048
MAC length: 32, salt length: 8
PKCS7 Encrypted data: PBES2, PBKDF2, AES-256-CBC, Iteration 2048, PRF hmacWithSHA256
Certificate bag
Bag Attributes
    friendlyName: keystore
    localKeyID: E4 53 9E 92 2A 31 1B CE 65 6A D5 13 43 8D 15 1C 62 93 33 22
subject=C=in, ST=andhra, L=kurnool, O=sanstark, OU=cybersecurity, CN=sanstark, emailAddress=harishwar1125@gmail.com
issuer=C=in, ST=andhra, L=kurnool, O=sanstark, OU=cybersecurity, CN=sanstark, emailAddress=harishwar1125@gmail.com
-----BEGIN CERTIFICATE-----
MIID3TCCAsWgAwIBAgIUDfhw/G5LYe3/N2V2F7L+nNboHukwDQYJKoZIhvcNAQEL
BQAwgZYxCzAJBgNVBAYTAmluMQ8wDQYDVQQIDAZhbmRocmExEDAOBgNVBAcMB2t1
cm5vb2wxETAPBgNVBAoMCHNhbnN0YXJrMRYwFAYDVQQLDA1jeWJlcnNlY3VyaXR5
MREwDwYDVQQDDAhzYW5zdGFyazEmMCQGCSqGSIb3DQEJARYXaGFyaXNod2FyMTEy
NUBnbWFpbC5jb20wHhcNMjUwNzEwMDY1ODMzWhcNMjUwNzExMDY1ODMzWjCBljEL
MAkGA1UEBhMCaW4xDzANBgNVBAgMBmFuZGhyYTEQMA4GA1UEBwwHa3Vybm9vbDER
MA8GA1UECgwIc2Fuc3RhcmsxFjAUBgNVBAsMDWN5YmVyc2VjdXJpdHkxETAPBgNV
BAMMCHNhbnN0YXJrMSYwJAYJKoZIhvcNAQkBFhdoYXJpc2h3YXIxMTI1QGdtYWls
LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANoE/rJoNmakSwYo
n4nKAVCMCpE42icDrNj3DJe0TMfDUrZO7fP7wNdqY+/UmCMgcwKExvSOCbVYz8NM
h0Jmm4Ij1zqR8UBV2/GCbGk5vkl8Bbud2nEKZhcESBkB7ExvyZdUihxKk/cuufuw
sIFkQrwpqQJTbaZW2k4ohYA4wE3PS6bEyWiY0wDJNEUycDy1pkscAz2nDiifq8i9
X1wZ/rz8FyGV8ytF9YURkp1HHTznERp+lKJaorr1g1e3eeXiHz0fOA4EbTy9EImg
nnQTKPU+Kww5nOMzyRcSZcoy1eXk8PdRszQmL4wAOo+m3q9Kld9+xpcpN769100k
```

Fig: Keystore pcks12+cert

keytool is a command-line utility provided by **Java (JDK)** to manage **keystores**, which are storage files used to hold:

- Private keys
- Public key certificates
- Secret keys



```
┌──(harish㉿kali)-[~/openssl]
└─$ keytool -importcert \
-file keystore.crt \
> -alias kock \
> -keystore truststore.jks \
> -storepass \
> -noprompt
Owner: EMAILADDRESS=harishwar1125@gmail.com, CN=sanstark, OU=cybersecurity, O=sanstark, L=kurnool, ST=andhra, C=in
Issuer: EMAILADDRESS=harishwar1125@gmail.com, CN=sanstark, OU=cybersecurity, O=sanstark, L=kurnool, ST=andhra, C=in
Serial number: df870fc6e4b61edff37657617b2fe9cd6e81ee9
Valid from: Thu Jul 10 12:28:33 IST 2025 until: Fri Jul 11 12:28:33 IST 2025
Certificate fingerprints:
         SHA1: E4:53:9E:92:2A:31:1B:CE:65:6A:D5:13:43:8D:15:1C:62:93:33:22
         SHA256: 8B:B4:15:59:D5:C2:46:1E:5E:0E:F0:30:51:0F:B0:E0:BC:A0:D0:BC:FB:58:A3:A9:50:FE:07:8D:CE:2C:59:D9
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 0B AA 96 20 55 D3 1B 4F   01 51 9C 47 C6 91 68 5D  ... U..O.Q.G..h]
0010: B5 5D 0F 49                                        .].I
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore

┌──(harish㉿kali)-[~/openssl]
└─$ 
```

Fig: Keystore successful

My Understanding of Cryptography and Key Management

Cryptography is all about securing data using keys. I understood that with a private key, we can generate a public key, and from there, create a CSR (Certificate Signing Request) that holds our identity. Without a CSR, we can still make a certificate, but it becomes self-signed, useful only in our own environment. In the real world, a Certificate Authority (CA) like GoDaddy verifies our CSR and issues trusted certificates. These certificates ensure our identity is trusted across systems. Everything in cryptography revolves around securing data, proving identity, and ensuring safe communication