OpenSSL is an open-source toolkit used in TLS/SSL and **cryptography** to ensure **secure communications** through its robust command-line tool, which can be used to generate keys, create certificates, and its software library is used to implement SSL and TLS in software and internet servers widely use it to handle encryption

What OpenSSL is used for

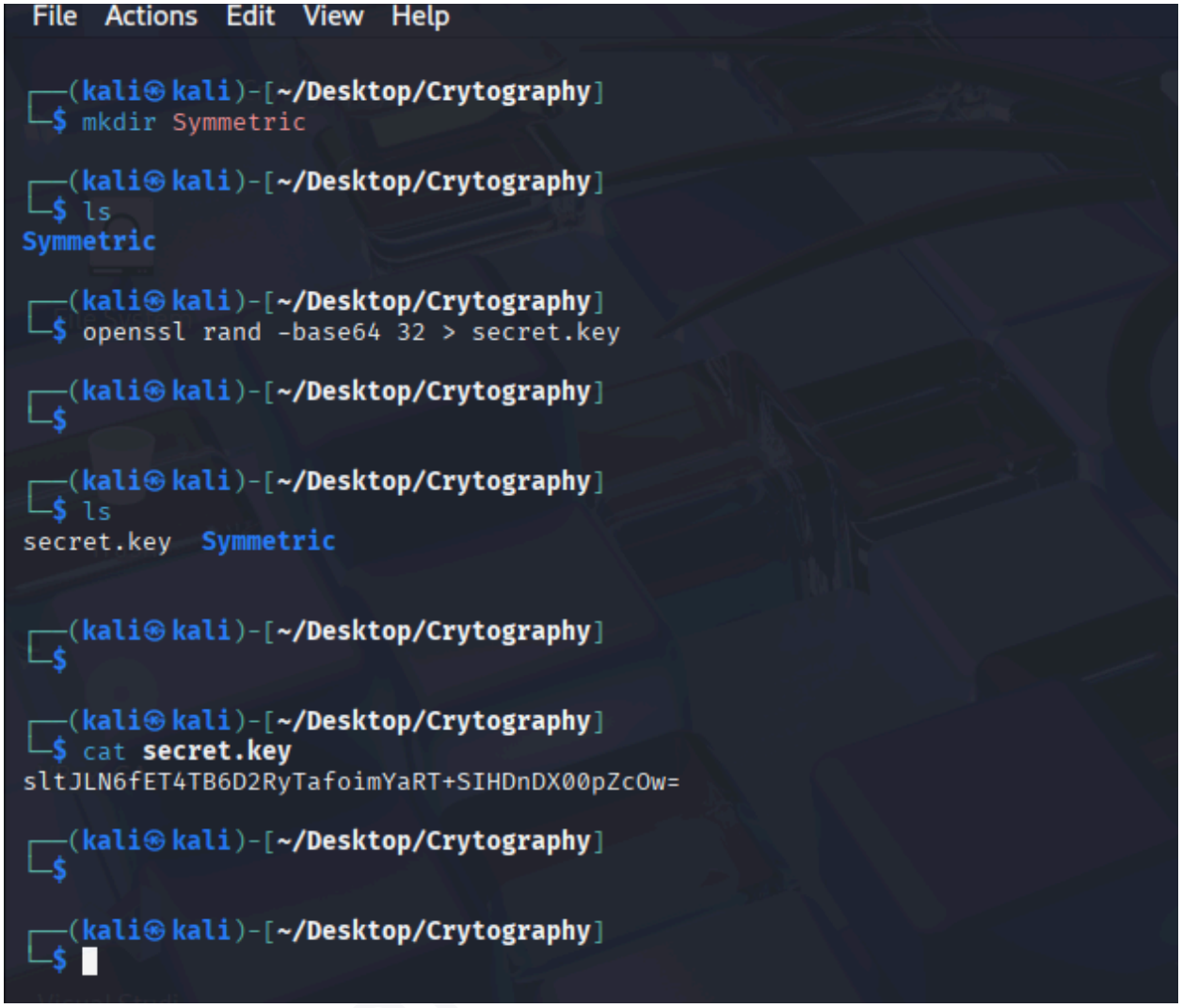
- Cryptography
- Ensuring Secure Communications
- Command-line tasks: The OpenSSL command-line tool is used as a versatile utility for
    - Generating Cryptographic Keys
    - Creating Certificate Signing Requests
    - Installing and managing SSL/TLS certificates
    - Checking Certificate information
    - Calculating message digests
    - Testing Connections

**Key Generation in OpenSSL**

```
Openssl rand -base64 32 > secret.key
```

This command generates 256 bits of cryptographically secure random data, converts it into a text-safe format, and then saves it in the file named “secret.key” We can use this command to create passwords, encryption keys for symmetric encryption

**openssl**: it invokes the OpenSSL library to function
**rand**: It is used to generate a random number through CSPRNG (Cryptographically Secure Pseudo-Random number Generator)
**-base64**: it is used to encode the output which is binary to standard ASCII characters
**32**: This is the length argument specifying the number of random bytes to generate, and this is equivalent to 256 bits
**>**: this redirects the output into the file specified, which is secret.key in this case, instead of outputting into the screen
**secret.key**: the destination filename




**Symmetric Encryption of a file with a generated key**

`openssl enc -aes-256-cbc -salt -in testing_file.txt -out encrypted_testing.bin -pass file:secret.key`

This command encrypts the file named “testing_file.txt” using the AES-256 algorithm and derives the password from the secret.key file created in the previous step

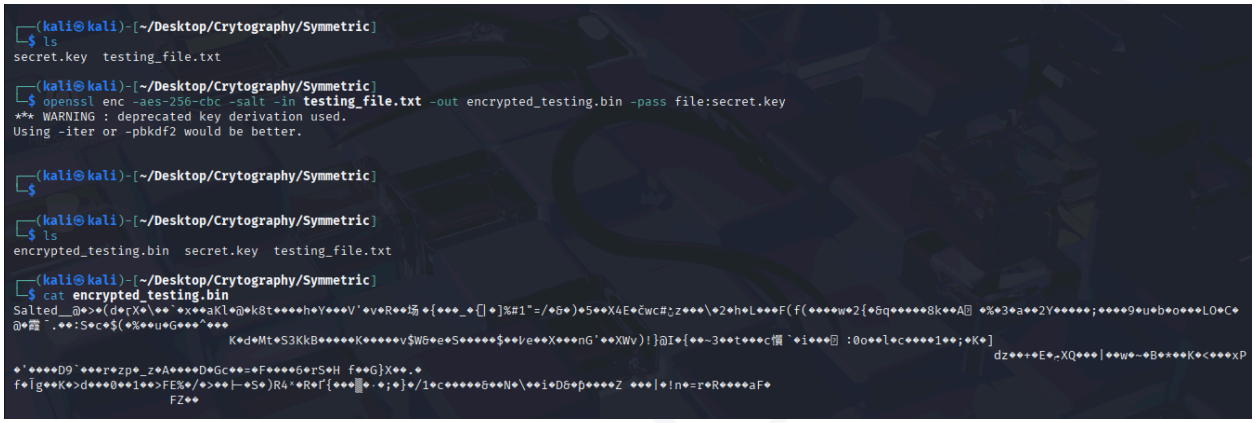
**openssl enc**: this calls the Encryption module in OpenSSL, and it is used for symmetric encryption

**-aes-256-cbc**: this defines the Cipher Suite used to scramble the data

- **aes**: this represents the algorithm of Advanced Encryption Standard
- **256**: this is the key size in bits

- **cbc**: This is the mode of operation, which is short for Cipher Block Chaining

**-salt**: This is used to generate a random string of data and combine it with the password before creating the actual encryption key
**-in testing_file.txt**: this takes in the file (testing_file.txt) to be encrypted
**-out encrypted_testing.bin**: this defines the output file as encrypted_testing.bin
**-pass file:secret.key**: this is the password which is in the file "secret.key" used by OpenSSL to encrypt the text file




**Symmetric Decryption of a file with a generated key**

```
openssl enc -d -aes-256-cbc -in encrypted_testing.bin -out restored_file.txt -pass file:secret.key
```

This command reverses the process of encryption, which was earlier carried out. The differences in the command are that the filter "-d" is applied after enc to symbolize decryption, and then -in takes in an encrypted file to then output -out a decrypted file (restored_file.txt)

**Asymmetric Encryption**

**Private Key Generation**

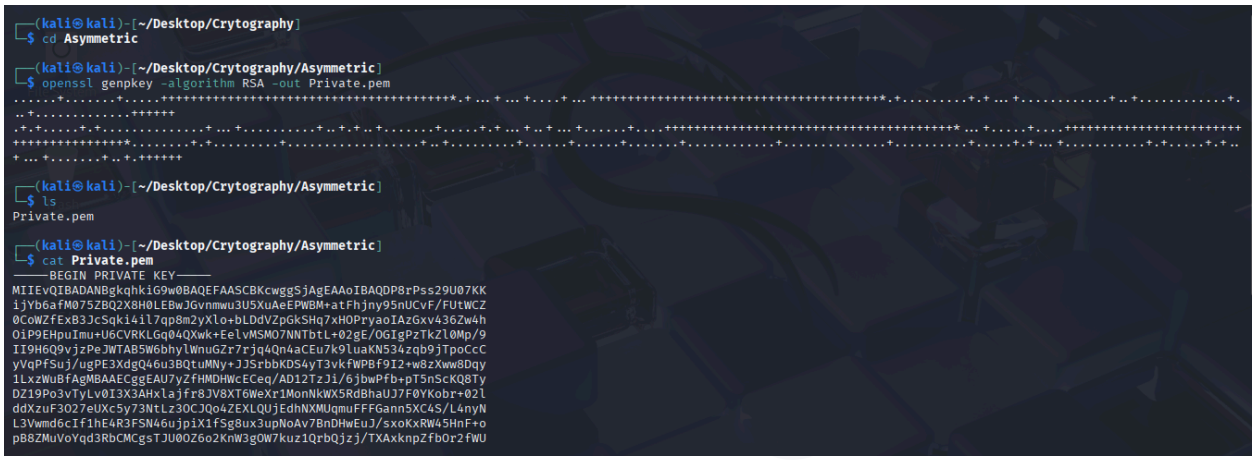
```
openssl genpkey -algorithm RSA -out Private.pem
```

This command is used to generate a private key using the RSA algorithm. This private key is a part of a key mechanism used to decrypt a file along with a public key for encryption in Asymmetric encryption/decryption

**openssl genpkey**: this calls the OpenSSL library to generate the private key

**-algorithm RSA**: This specifies the use of the Rivest-Shamir-Adleman cryptosystem.

**-out Private.pem**: The file where the key is saved.




**Public Key Generation**

```
openssl pkey -in Private.pem -pubout -out Public.pem
```

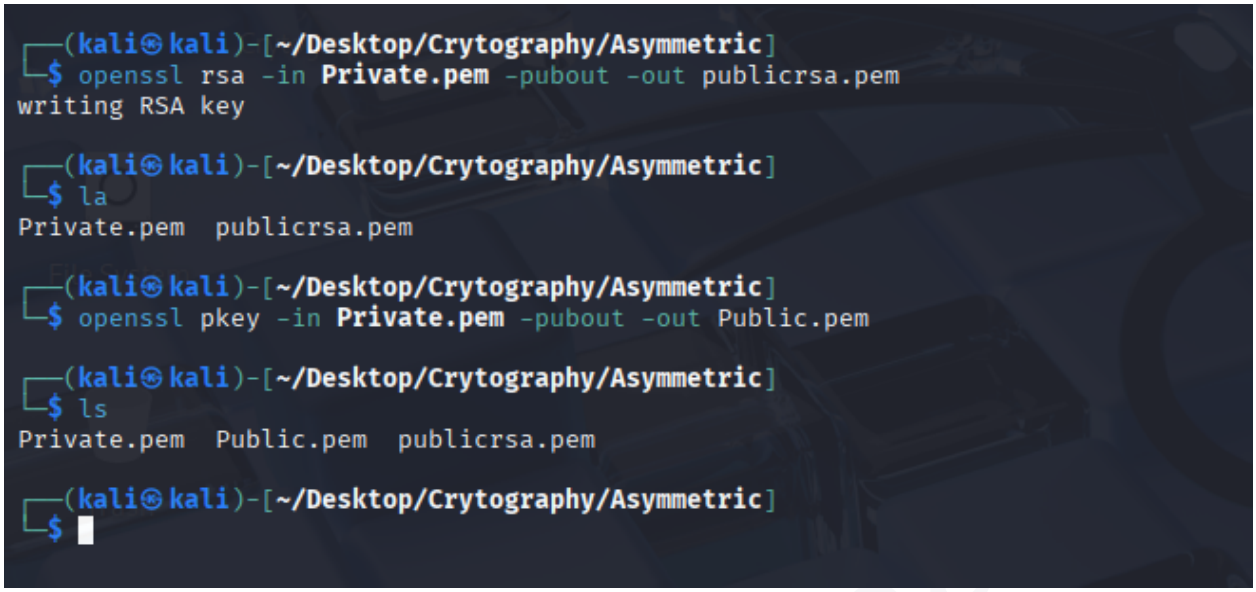
This command takes the Private key and uses it to produce the public key

**openssl pkey:**This calls the openssl to generate the key, in this case, the public key

**-in Private.pem:** this provides the private key file

**-pubout**: This tells OpenSSL to output only the public key component

**-out Public.pem**: This is where the public key is saved and stored




**Asymmetric Encryption of a file with a Public key**

```
Openssl pkeyutl -encrypt -inkey Public.pem -pubin  -in
testing_file.txt -out encrypted_testing_message.bin
```

This command is used to encrypt the file (testing_file.txt) to produce the encrypted output ( encrypted_testing_message.bin)

**openssl pkeyutl**: This is used to call a public key utility used for performing low-level operations like encrypt, sign, and verify using public or private keys

**-encrypt**:This defines the action of encrypting the input data

**-inkey Public.pem**:This is used to specify the file that contains the key to use for encryption

**-pubin:** This is used to tell OpenSSL that the key in use is a public key (If not included, it would read it as a Private Key by default, and an error would be output)

**-in testing_file.txt**:This specifies the input file(testing_file.txt) to be encrypted.

**-out encrypted_testing_message.bin**:This specifies where to store the encrypted file


**Asymmetric Decryption of a file with a Private key**

`openssl pkeyutl -decrypt -inkey Private.pem -in encrypted_testing_message.bin -out test_output.txt`

This command is used to reverse the process of encryption to get back the actual initial file

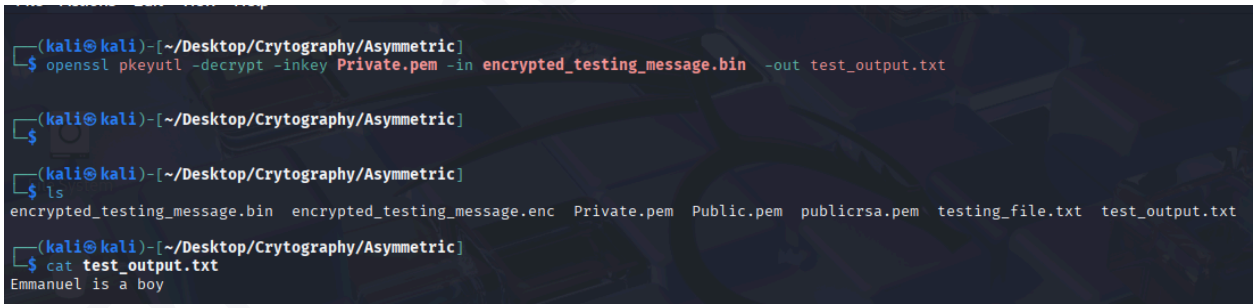
**openssl pkeyutl**:This is the public key utility as discussed earlier

**-decrypt:**This defines the action of decrypting the input data

**-inkey Private.pem**:This is used to specify the Private key used for decryption

**-in encrypted_testing_message.bin:**This specifies the input file(encrypted_testing_message.bin) to be decrypted.

**-out test_output.txt:**This specifies where to store the decrypted file



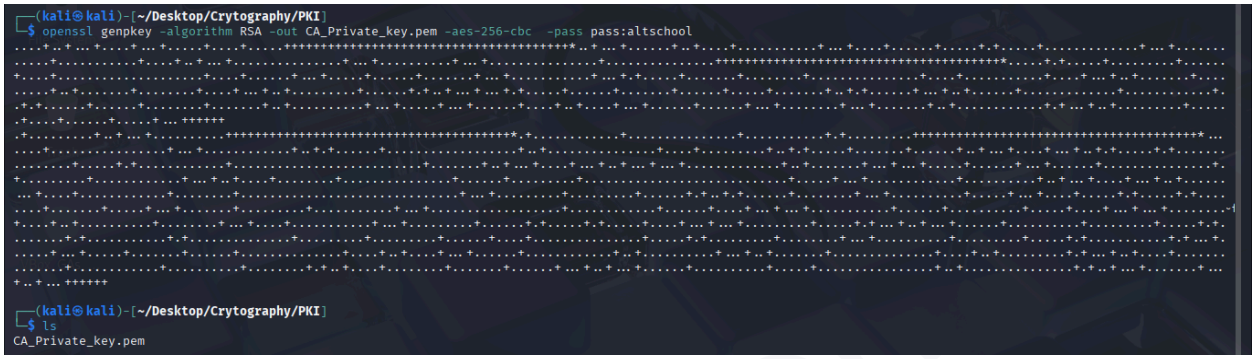

**Public Key Infrastructure (PKI)**

Public Key Infrastructure (PKI) is a set of policies, roles, software, and hardware necessary to create, manage, distribute, use, store, and **revoke digital certificates** and **manage public-key encryption**. Its primary purpose is to **bind public keys with the identities of users**, thereby enabling secure communication and data exchange.

**Certificate Authority Private Key Generation**

```
openssl genpkey -algorithm RSA -out CA_Private_key.pem -aes-256-cbc
-pass pass:altschool
```

This command creates a Private key with the RSA algorithm and encrypts it with AES and has a password of altschool




**Self-signed Root Certificate Creation for the CA**

```
openssl req -x509 -new -key CA_Private_Key.pem -days 365 -out
CA_Certificate.pem -passin pass:altschool
```

This command creates a self-signed root certificate for the Certificate Authority (CA) and this is very important in the public key infrastructure

**openssl req:** this tells OpenSSL to use the tool for certificate requests and generation

**-x509**: This switches the mode from “Certificate Signing Request(CSR)” to “Signed-Signed Certificate,” ensuring that the certificate generated is the Root Certificate.

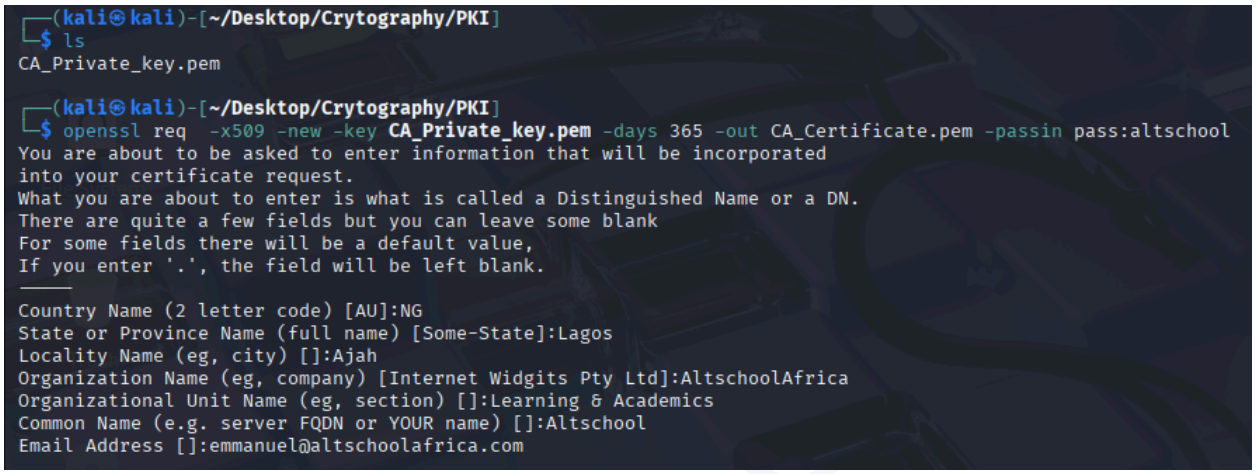
**-new**: it instructs OpenSSL to prompt for identity information, which the CA supplies to create the certificate.

**-key CA_Private_key.pem**: it takes the created Private key (CA_Private_key.pem) to sign the certificate.

**-days 365**: this sets the expiration date for the certificate used by the Certificate Authority

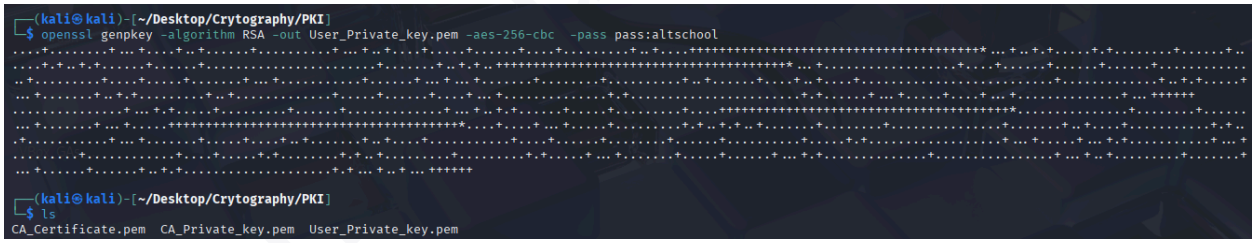
**-out CA_Certificate.pem**: this determines where to store the CA Certificate file

**-passin pass:altschool**: Since the Private key is encrypted, this command provides the password to decrypt the Private key for it to be use to sign the CA certificate




**User Private Key Generation**

This snippet below shows the creation of user's private key generation, which would be used to request a certificate




**User Certificate Signing Requests Generation**

openssl req -new -key User_Private_key.pem -out user_CSR.pem

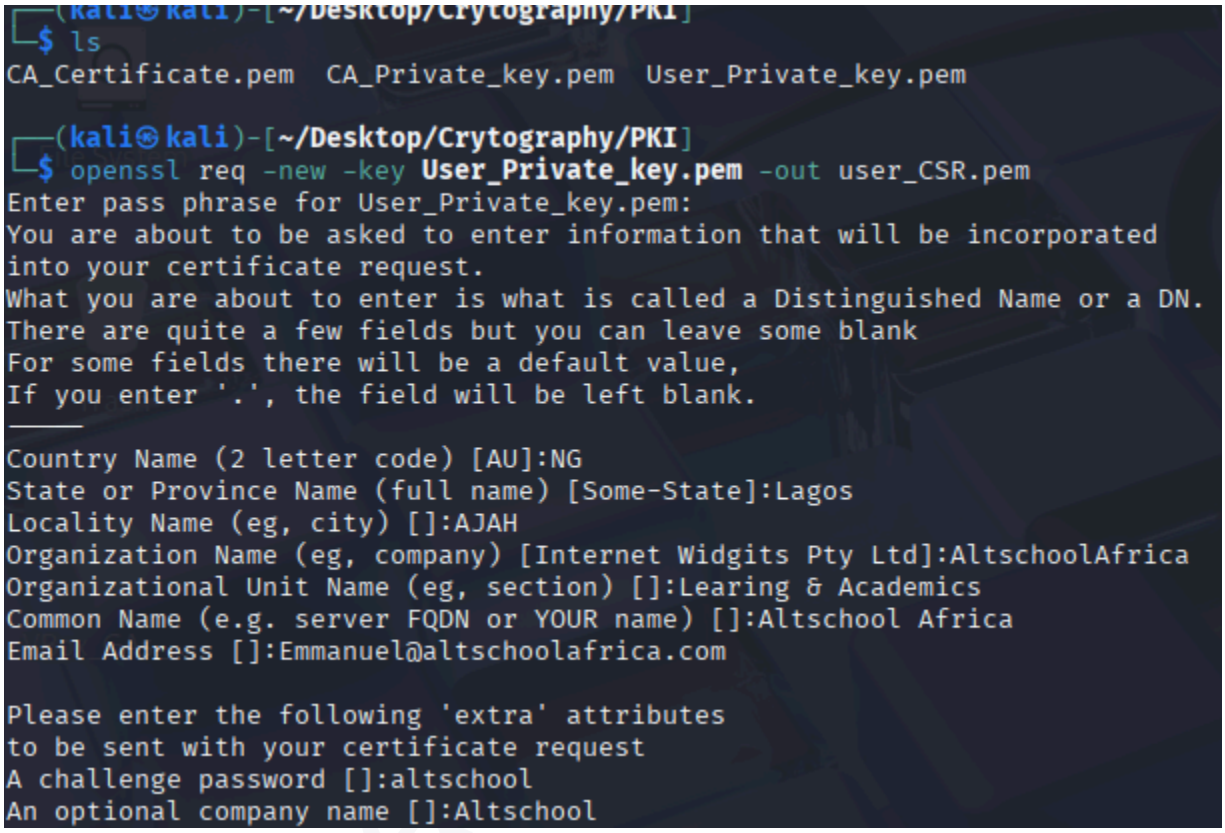
This command creates a Certificate Signing Request ,which is used by the user to request their Certificate by the Certificate Authority

**openssl req**:This invokes the request management tool to create the CSRs

**-new**:This identifies to OpenSSL to prompt for identity information in creating the CSRs

**-key User_Private_key.pem**:This points to the private key that would be associated with the certificate, this tells the CSRs that you are the one applying for it

**-out User_CSR.pem**:This determines where to store the CSR Certificate file




**Verification of Certificate Signing Request (CSR)**

```
openssl req -text  -noout -verify -in user_CSR.pem
```

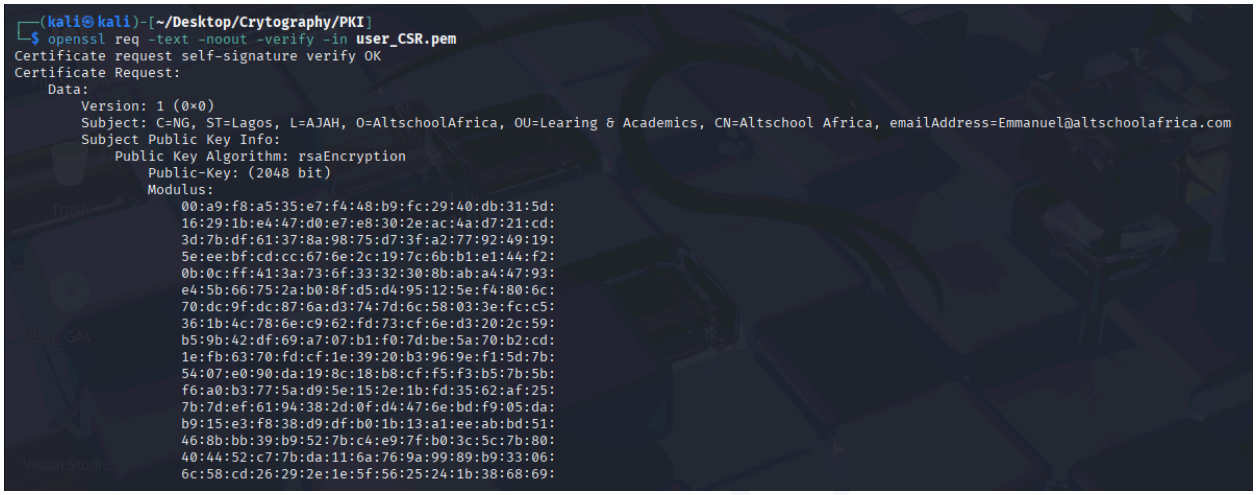
This command verifies the  Certificate Signing Request(CSR) to see if it is valid; it acts as quality control

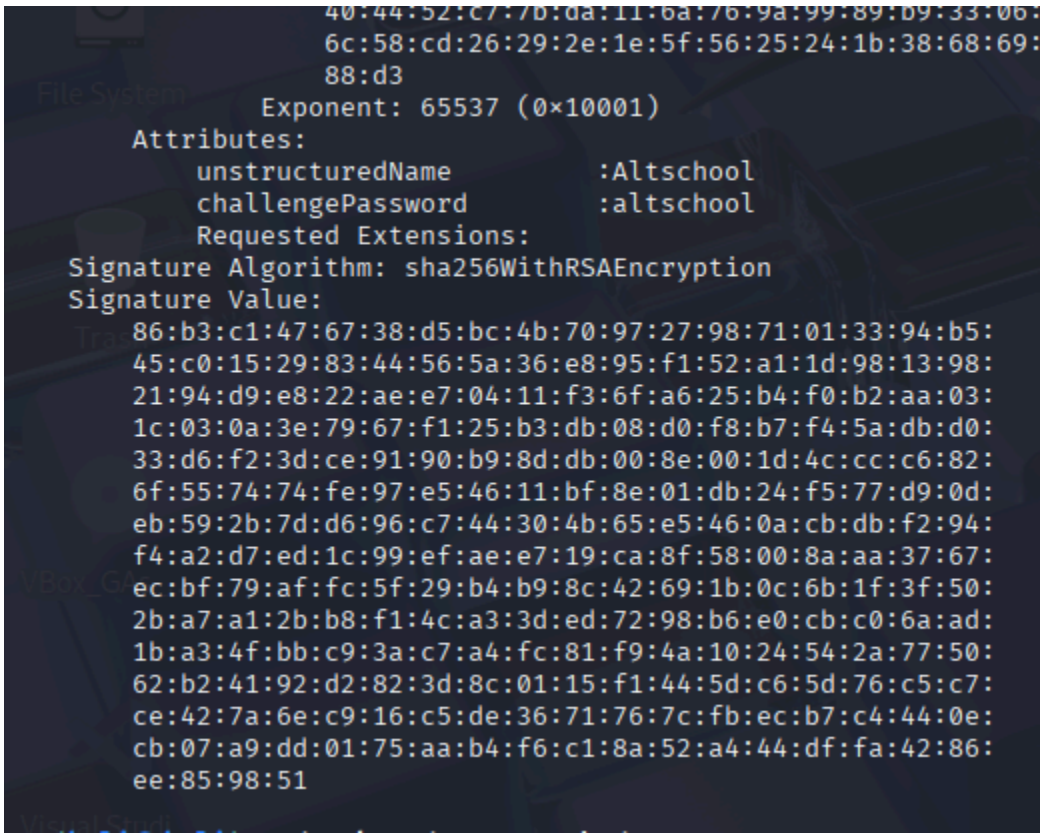
**-text**:This command ensures to decode the Base64 data and print it in human-readable English

**-noout**:This ensures that OpenSSL does not send the data to the terminal but ensures it is stored or saved somewhere else

**-verify**:This is used to perform a cryptographic validation as it would extract the Public key from the CSR and use it to verify the signature on the CSR

**-in user_CSR.pem**:This indicates the file that needs to be validated




**Request for the Real Certificate**

```
openssl x509 -req -in user_CSR.pem -CA CA_Certificate.pem -CAkey
CA_Private_key.pem -CAcreateserial  -out user_Certificate.pem -days
365 -passin pass:altschool
```

This command simulates the Certificate Authority(CA) to take the user application(the CSR), reviews it, and stamps it with the CA's Private key to give a valid and trusted certificate

**openssl x509**:This calls the tool used for certificate management

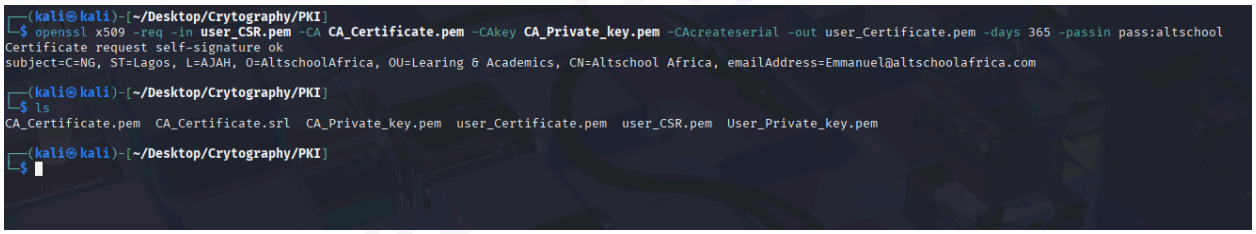
**req**: This tells OpenSSL that the input file provided is the CSR

**-CA CA_Certificate.pem**:This specifies the Certificate of the CA that would do the signing

**-CAkey CA_Private_key.pem**:This is the CA's Private key

**-CAcreateserial**:This would create a unique serial number associated with this certificate that helps prevent replay attacks

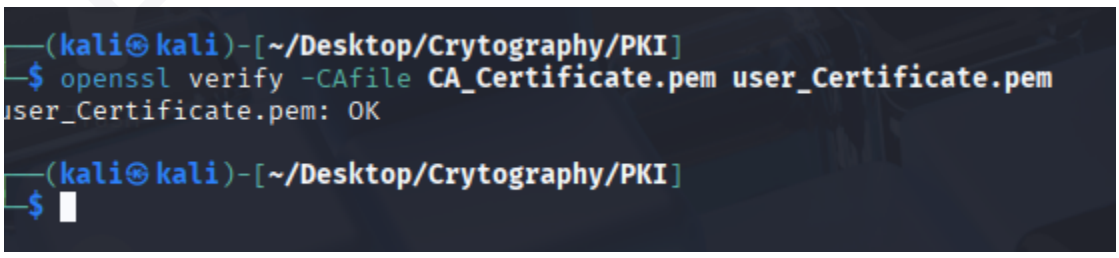
**-out user_Certificate.pem**:This is where the signed certificate is saved




**Verification of Requested Certificate with Root Certificate**

```
openssl verify -CAfile CA_Certificate.pem user_Certifcate.pem
```

This command is used to compare the generated requested certificate to the root CA certificate to confirm the validity



**Digital Signatures**
Digital signatures provide a cryptographic mechanism to **authenticate the source** of a digital message or document and ensure its **integrity**. They function similarly to a handwritten signature but are far more secure. **A digital signature is created by hashing the document and then encrypting that hash with the sender's private key.**
The signature proves:

1. **Authentication:** The data originated from the signer (non-repudiation).
2. **Integrity:** The data has not been altered since it was signed.

**Digitally Signing a Document**

```
openssl dgst -sha256 -sign User_Private_key.pem -out document.sig testing_file.txt
```

This command is used to digitally sign a document with  a private key

**openssl dgst**:This calls the tool to create digest and also hashing

**-sha256**:This determines the hash function which is used to hash the document

**-sign**:This determines the action to take which to sign off the document in  this case

**User_Private_key.pem**:This is the private key file used to sign the hash

**-out document.sig testing_file.txt**:This is the output file of the signature(document.sig) of the document(testing_file.txt)

```
┌──(kali㉿kali)-[~/Desktop/Crytography/Digital_Signatures]
└─$ ls
Public_key.pem  testing_file.txt  User_Private_key.pem

┌──(kali㉿kali)-[~/Desktop/Crytography/Digital_Signatures]
└─$ openssl dgt -sha256 -sign User_Private_key.pem -out document.sig testing_file.txt
Invalid command 'dgt'; type "help" for a list.

┌──(kali㉿kali)-[~/Desktop/Crytography/Digital_Signatures]
└─$ openssl dgst -sha256 -sign User_Private_key.pem -out document.sig testing_file.txt
Enter pass phrase for User_Private_key.pem:

┌──(kali㉿kali)-[~/Desktop/Crytography/Digital_Signatures]
└─$ ls
document.sig  Public_key.pem  testing_file.txt  User_Private_key.pem

┌──(kali㉿kali)-[~/Desktop/Crytography/Digital_Signatures]
└─$ cat document.sig
[illegible]
```

**Verification of the Signed Document**

`openssl  dgst -sha256 -verify Public_key.pem -signature document.sig testing_file.txt`

This command is used to verify the signed copy (document.sig)
Of the document(testing_file.txt)

**-verify Public_key.pem:** this action is to verify the document and confirmed used the user Public key, since it was signed with the Private key, verification would be carried out through the Public key

**-signature document.sig testing_file.txt**:This tells OpenSSL to confirm the signature.

```
┌──(kali㉿kali)-[~/Desktop/Crytography/Digital_Signatures]
└─$ ls
document.sig  Public_key.pem  testing_file.txt  User_Private_key.pem

┌──(kali㉿kali)-[~/Desktop/Crytography/Digital_Signatures]
└─$ openssl dgst -sha256 -verify Public_key.pem -signature document.sig testing_file.txt
Verified OK
```