INFORMATION SECURITY

ASSIGNMENT 3

NAMES:Muhammad Zain UL Abidin,Aitsam Tariq Bajwa

Roll no:21i-1551,21i-1583

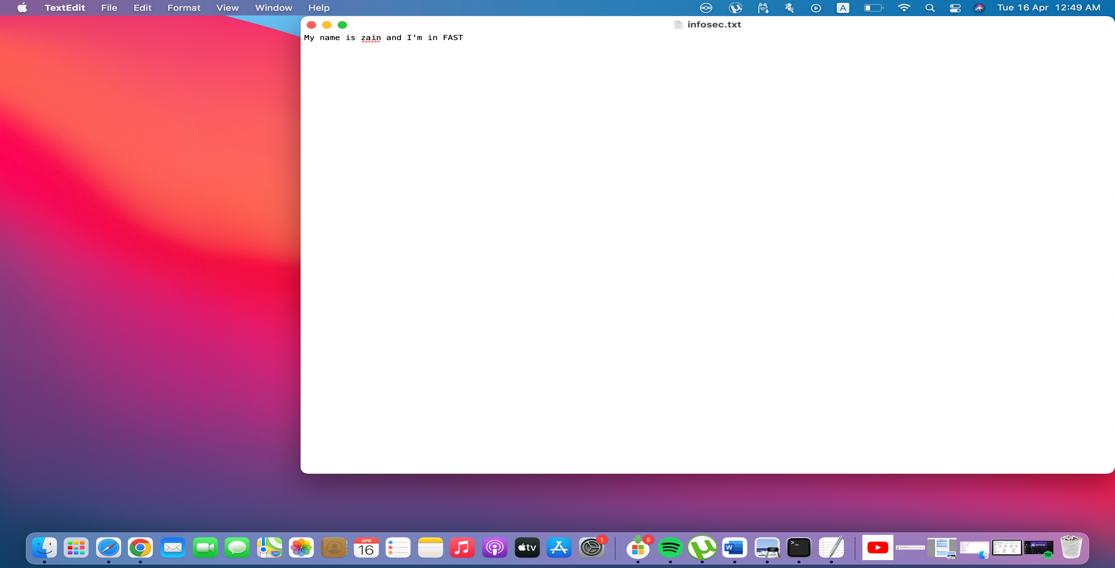
OpenSSL Practical Exercise

Encrypt and Decrypt Files

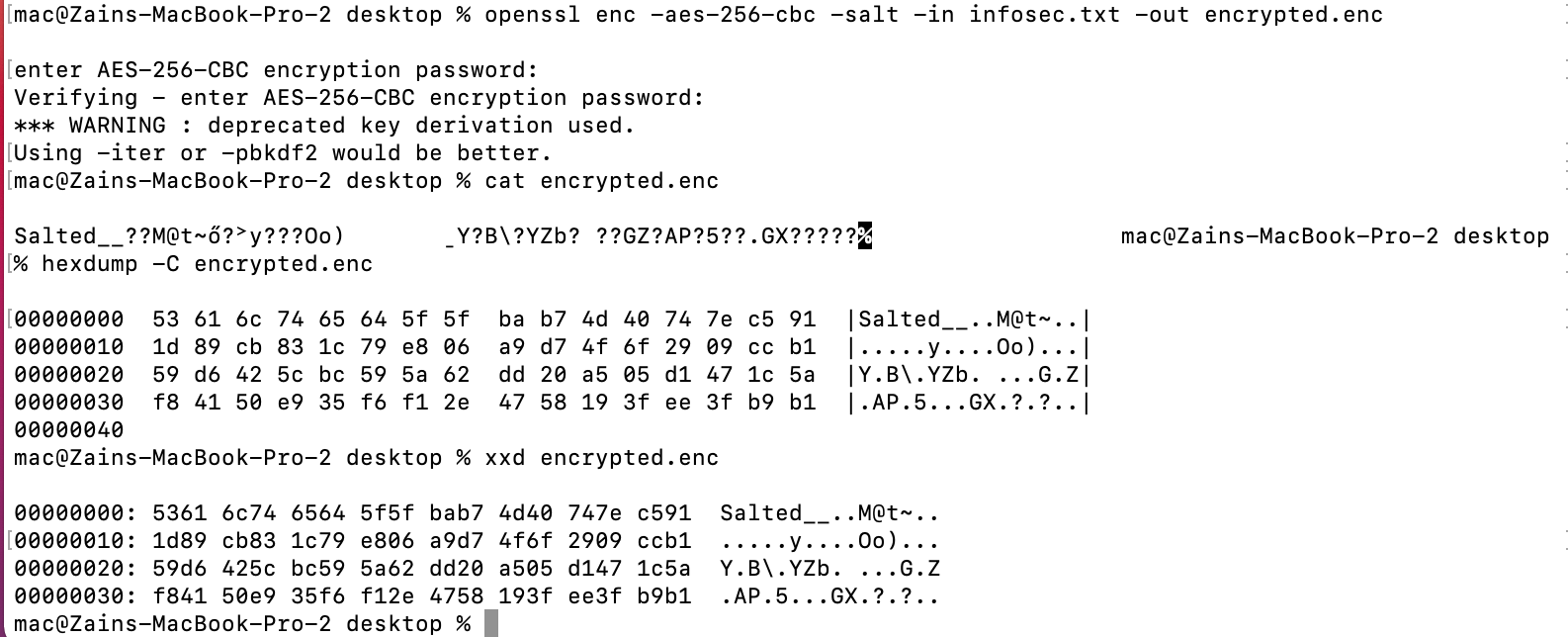
Use OpenSSL commands to encrypt and decrypt a file using AES-256 encryption.

Document the OpenSSL commands used and provide a brief explanation of each step.

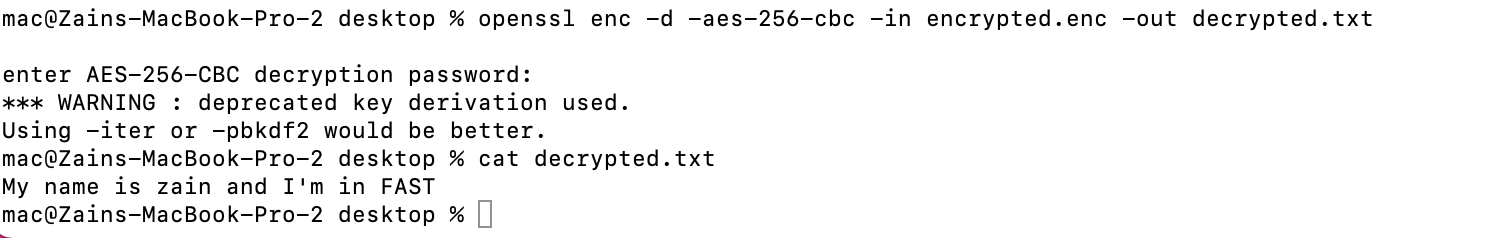
This is a text file with a simple sentence written in it



These are the encrypted file data in different formats including binary and hexadecimal format.



Verifying the text by decrypting it



Openssl commands used

Encryption:

openssl enc -aes-256-cbc -salt -in example.txt -out example.enc

* **-aes-256-cbc**: Specifies the AES 256-bit encryption algorithm in Cipher Block Chaining (CBC) mode.
* **-salt**: Adds salt to the encryption for additional security.
* **-in example.txt**: Specifies the input file to be encrypted.

**-out example.enc**: Specifies the output file where the encrypted data will be written.

Decryption:

openssl enc -d -aes-256-cbc -in example.enc -out example.txt

* **-d**: Specifies the decryption mode.
* **-aes-256-cbc**: Specifies the AES 256-bit encryption algorithm in Cipher Block Chaining (CBC) mode, matching the encryption used.
* **-in example.enc**: Specifies the input file to be decrypted.

**-out example.txt**: Specifies the output file where the decrypted data will be written.

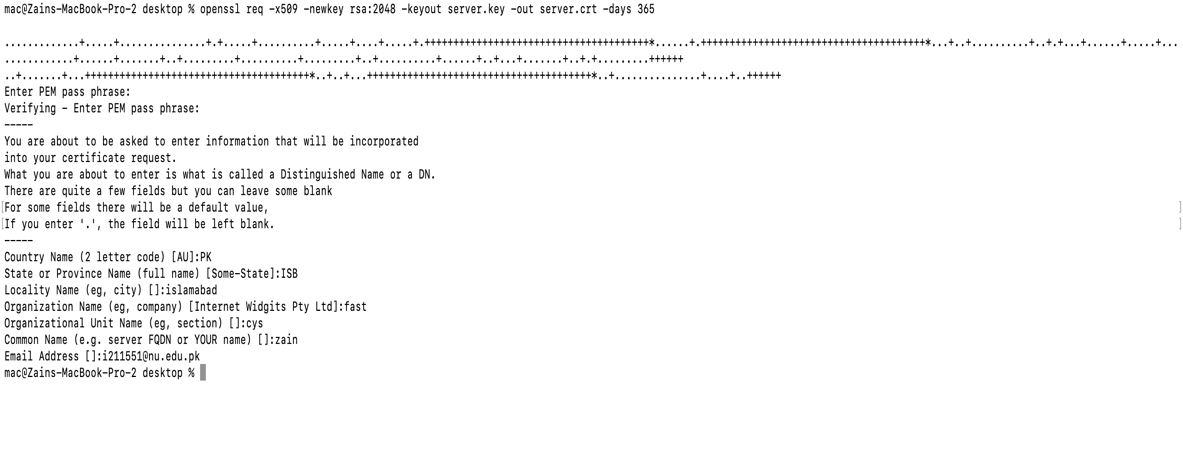
Manage Certificates

Generate a self-signed certificate and a certificate signing request (CSR) using OpenSSL.

Sign the CSR with the self-signed certificate to create a CA-signed certificate.

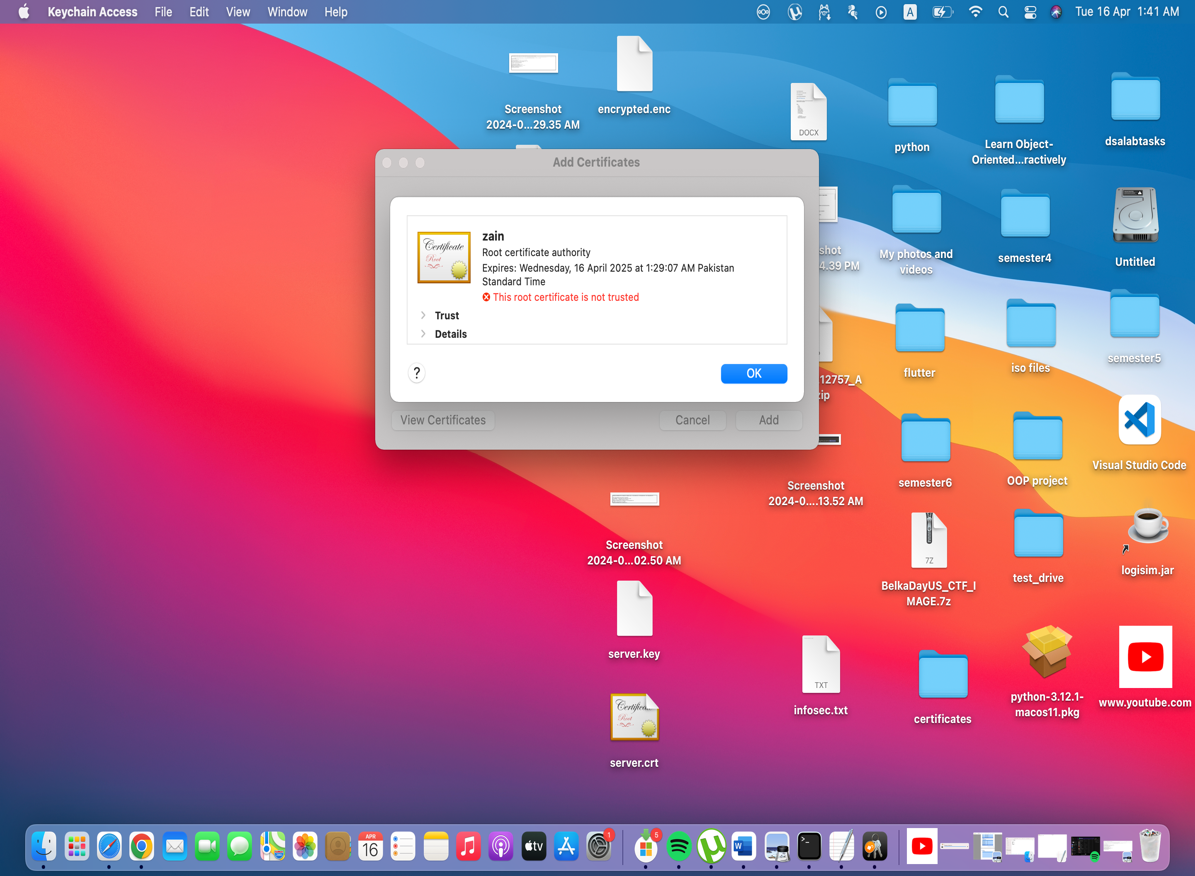
Document the OpenSSL commands used and provide a brief explanation of each step.

1)Self signed certificate

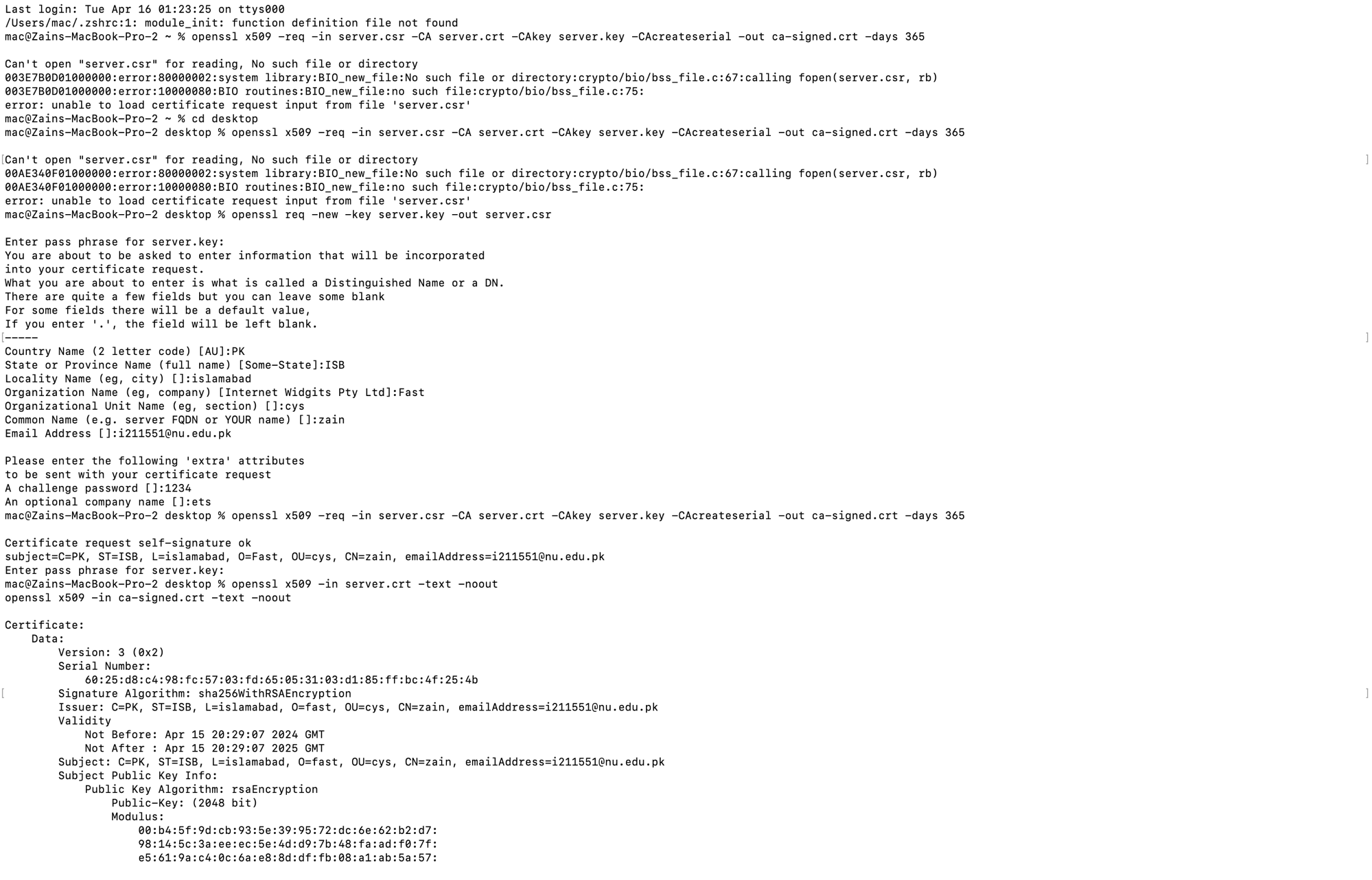


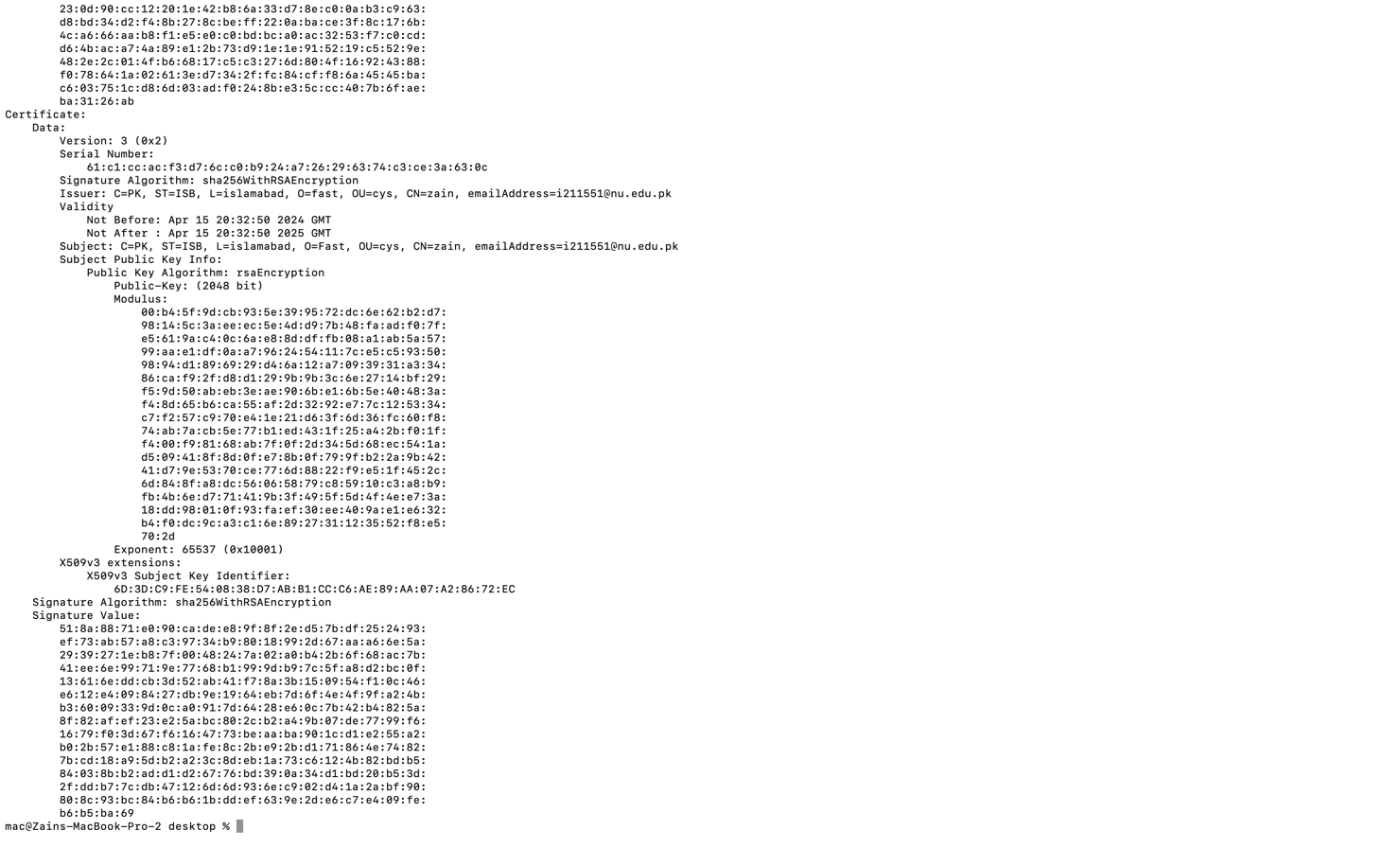


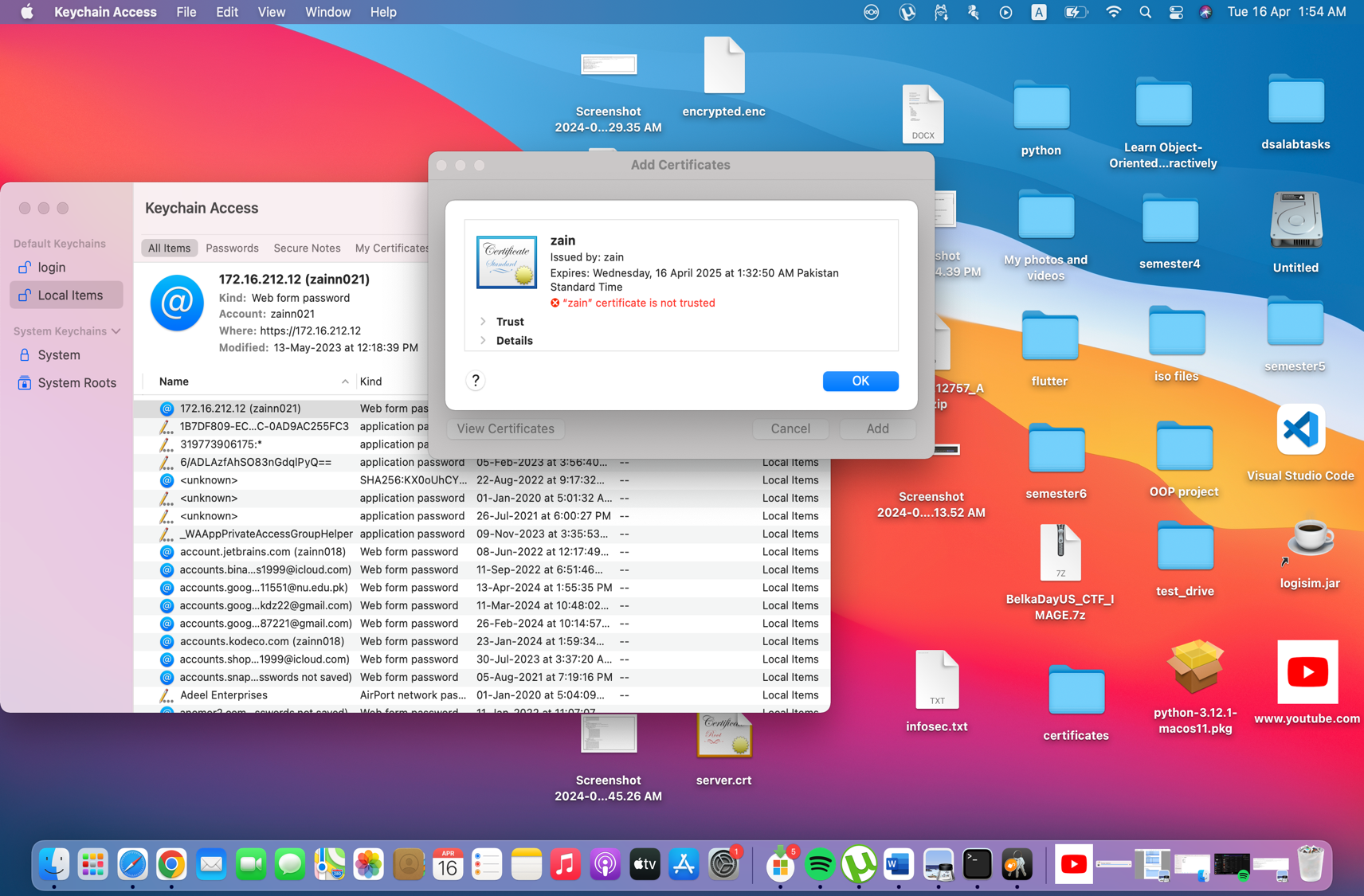




CA certified certificate







Documenting the OpenSSl commands

**openssl req -x509 -newkey rsa:2048 -keyout server.key -out server.crt -days 365**

openssl req: This command is used for generating a CSR (Certificate Signing Request) and managing X.509 certificates.

-x509: This option tells OpenSSL to output a self-signed certificate instead of a CSR.

-newkey rsa:4096: This option generates a new RSA private key with a key length of 4096 bits.

-keyout server.key: This specifies the file where the generated private key will be saved .

-out ca-signed.cert: This specifies the file where the generated self-signed certificate will be saved (ca-signed.cert).

-days 365: This specifies the validity period of the certificate in days

**openssl x509 -req -in server.csr -CA server.crt -CAkey server.key -CAcreateserial -out ca-signed.crt -days 365**

openssl x509: This command is used for various X.509 certificate operations, including signing and managing certificates.

-req: Specifies that the input file (server.csr) is a CSR (Certificate Signing Request).

-in server.csr: Specifies the input file containing the CSR that needs to be signed.

-CA server.crt: Specifies the CA certificate file (server.crt) that will be used to sign the CSR.

-CAkey server.key: Specifies the private key file (server.key) corresponding to the CA certificate. This key is required to sign the CSR.

-CAcreateserial: This option tells OpenSSL to create a serial number file if it doesn't already exist. The serial number is used to uniquely identify the signed certificate.

-out ca-signed.crt: Specifies the output file where the CA-signed certificate will be saved. In this case, it's named ca-signed.crt

**openssl x509 -in server.crt -text -noout**

openssl x509: This command is used for various X.509 certificate operations.

-in server.crt: Specifies the input file containing the X.509 certificate (server.crt) for which you want to display information.

-text: This option tells OpenSSL to display the certificate details in text form, making it human-readable. It includes information such as the certificate's subject, issuer, validity period, public key details, and any extensions.

-noout: This option instructs OpenSSL not to output the encoded certificate data, only the human-readable text representation.

**openssl x509 -in ca-signed.crt -text -noout**

openssl x509: This command is used for signing, verifying, and manipulating X.509 certificates.

-in ca-signed.crt: This option specifies the filename of the input certificate (ca-signed.crt in this case). This is the certificate file that you want to examine.

-text: This option specifies that the output should be displayed in text format. The command will print detailed information about the certificate, including its subject, issuer, validity period, public key, and other attributes.

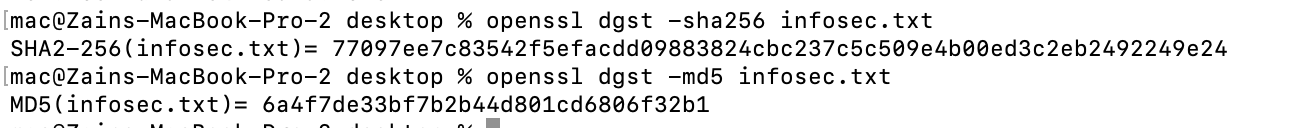
-noout: This option tells OpenSSL not to output the certificate itself, but only the text representation of its contents. Without this option, OpenSSL would print both the text representation and the encoded certificate data.

Generate Hashes

Generate and verify SHA-256 and MD5 hashes for a file using OpenSSL commands.

Document the OpenSSL commands used and provide a brief explanation of each step.

Generating SHA-256 and MD5 hashes for the file “infosec.txt”



Openssl commands:

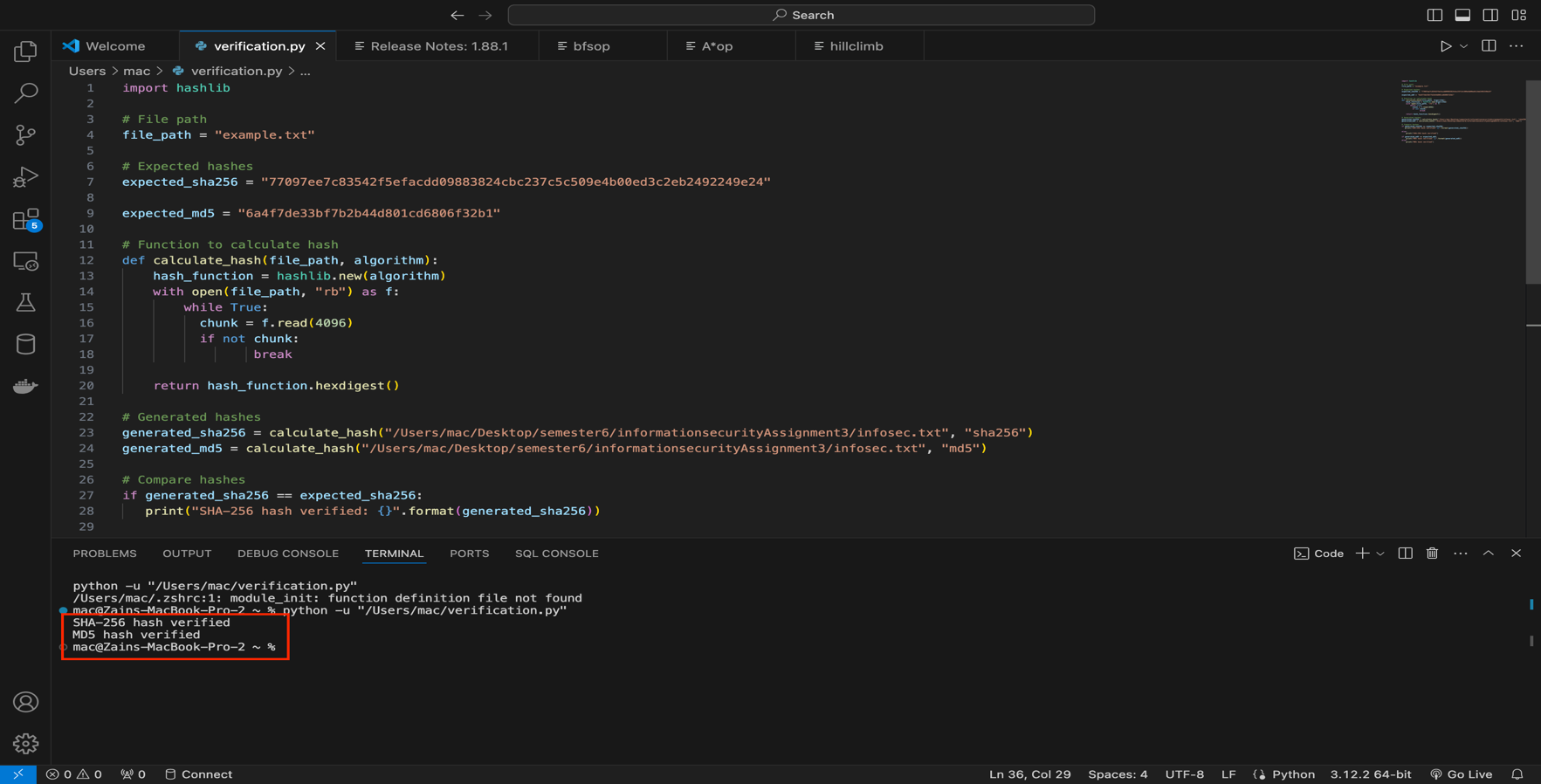
**openssl dgst -sha256 infosec.txt**

This generates the sha256 hash of the file

**openssl dgst -md5 infosec.txt**

This generates the md5 hash of the file

Verifying the hashes through python script:



Implement Secure Communication

Set up a secure TLS/SSL connection between a server and a client using OpenSSL.

Document the OpenSSL commands used and provide a brief explanation of each step.

Firstly generating a client private key and certificate

Using openssl req -x509 -nodes -newkey rsa:2048 -keyout client.key -out client.crt -days 365

**req: Generate a Certificate Signing Request (CSR).**

**-x509: Create a self-signed certificate.**

**-nodes: Do not encrypt the private key.**

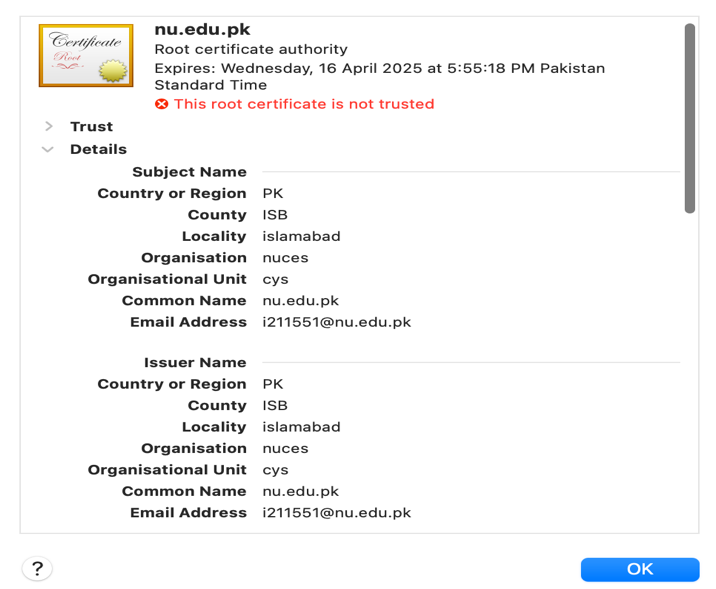
**-newkey rsa:2048: Generate a new RSA key pair with a key length of 2048 bits.**

**-keyout: Specify the output file for the private key.**

**-out: Specify the output file for the certificate.**

**-days 365: Set the validity period of the certificate to 365 days.**





Then we’ll configure the server by creating a server configuration file using the command

**echo -e "ssl\_certificate server.crt;\nssl\_certificate\_key server.key;\n" > server.conf**

Then we’ll start the server using the command

**openssl s\_server -accept 443 -cert server.crt -key server.key -www**

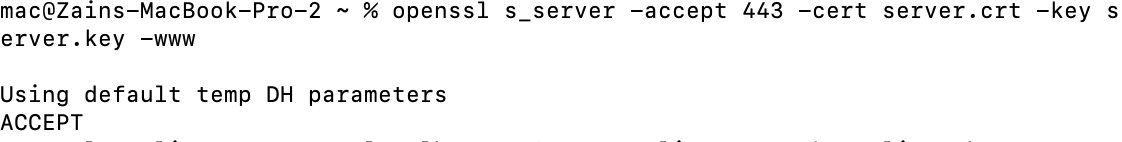
s\_server: Start an SSL/TLS server.

-accept 443: Listen on port 443 for incoming connections (you can change the port as needed).

-cert: Specify the server certificate file.

-key: Specify the server private key file.

-www: Serve web pages from the current directory.



The output "Using default temp DH parameters" indicates that OpenSSL is using default Diffie-Hellman (DH) parameters for the SSL/TLS handshake process and the

server is ready and accepting connections on the specified port (port 443).This means that the server is now listening for incoming SSL/TLS connections from clients.

**Now we’ll connect to the client**

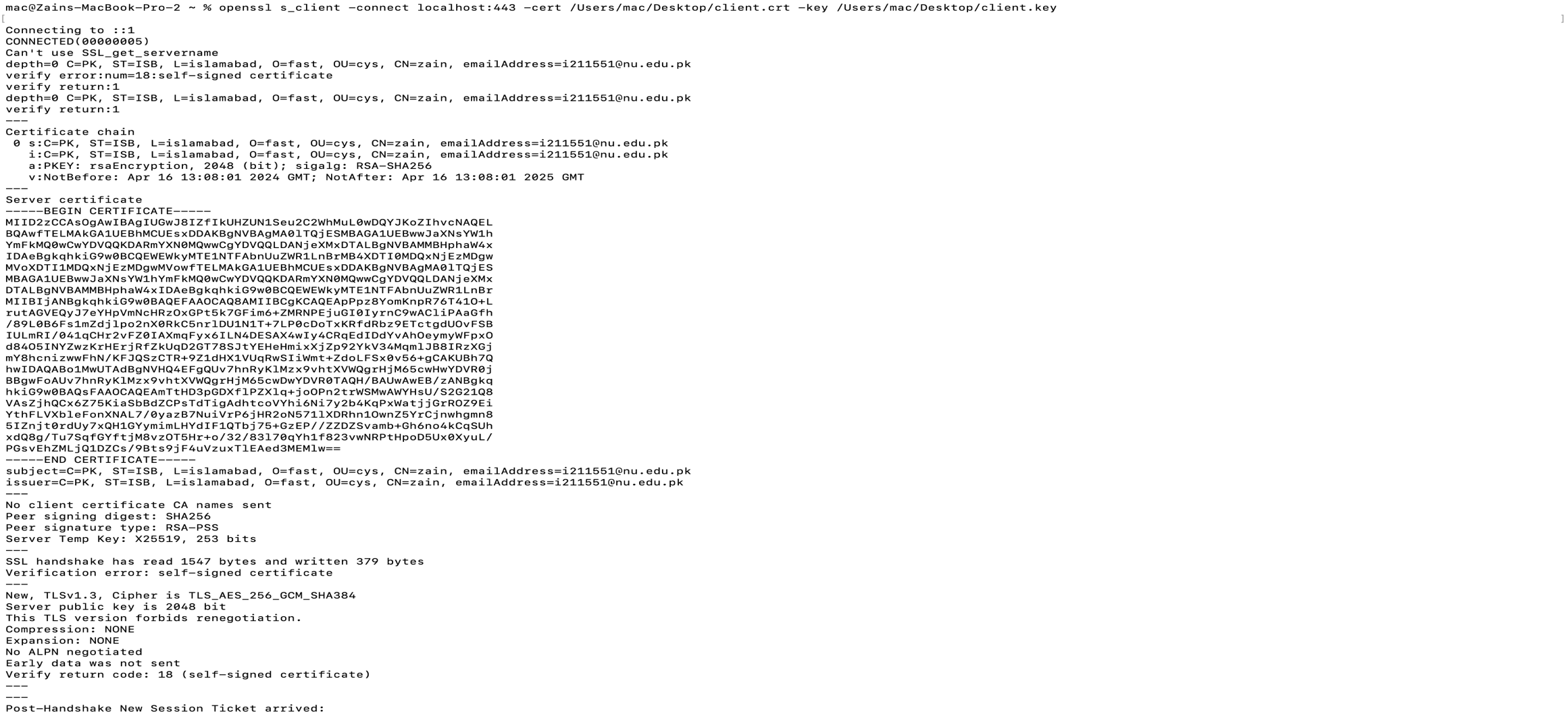
**openssl s\_client -connect localhost:443 -cert client.crt -key client.key**

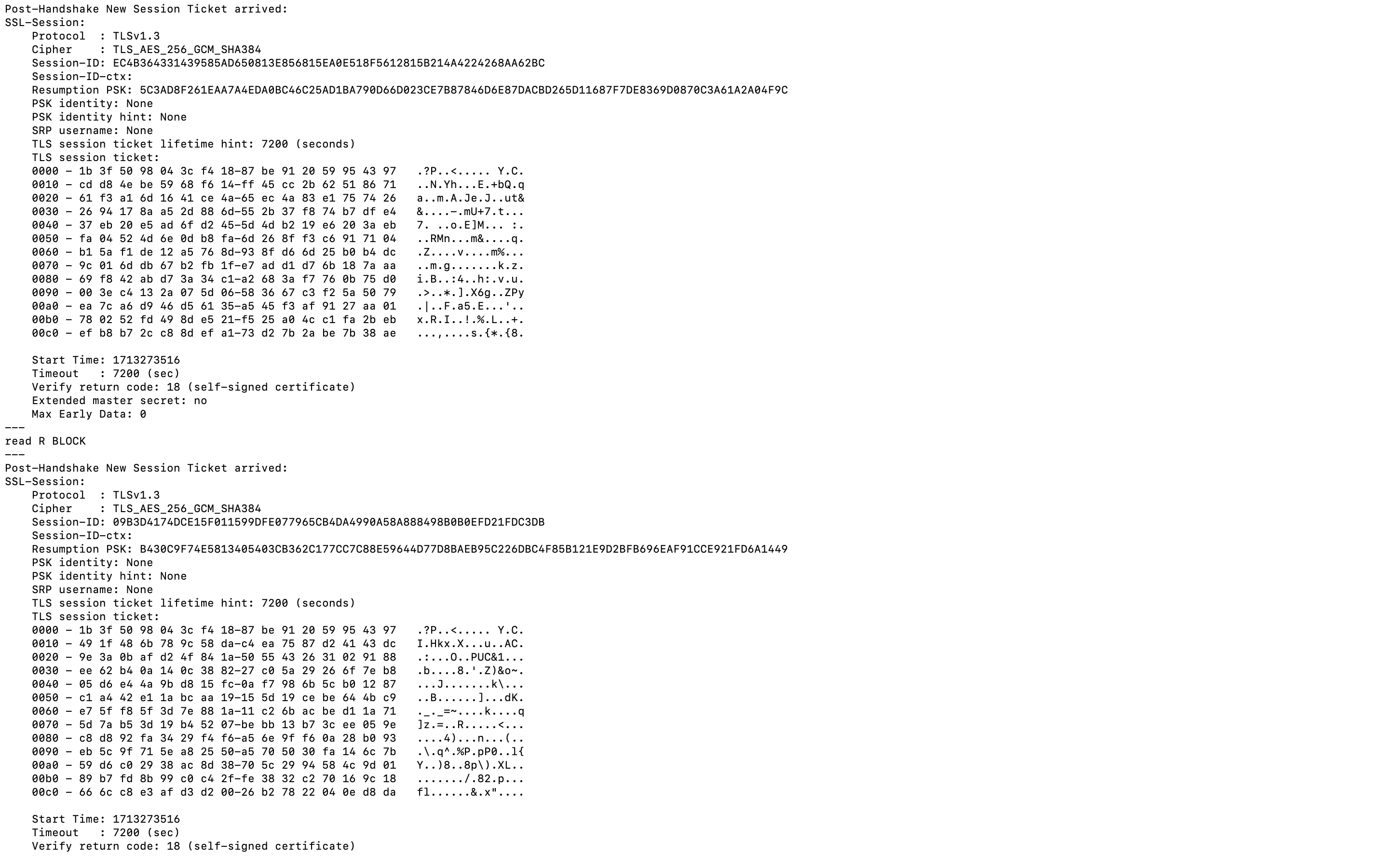
**s\_client**: Start an SSL/TLS client.

**-connect localhost:443**: Connect to the server running on localhost at port 443.

**-cert**: Specify the client certificate file.

**-key**: Specify the client private key file.





The output indicates that the SSL/TLS connection was successfully established with the server running on localhost at port 443, but the server's certificate is self-signed, resulting in a verification error.

Digital Signature and Verification

Generate a digital signature for a file using OpenSSL commands.

Verify the digital signature to ensure the file's integrity using OpenSSL commands.

Document the OpenSSL commands used and provide a brief explanation of each step.

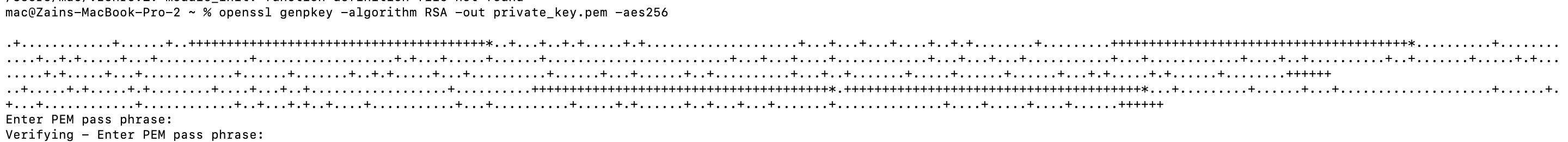
Firstly, generating a private key:

**openssl genpkey -algorithm RSA -out private\_key.pem -aes256**

genpkey: This subcommand is used to generate a private key.

-algorithm RSA: Specifies the algorithm to use for key generation.

-out private\_key.pem: Specifies the file path where the generated private key will be saved.

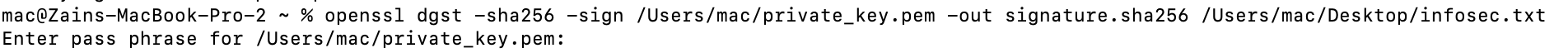
****

Extracting a public key from the private key file using

**openssl rsa -in private\_key.pem -pubout -out public\_key.pem**

This command extracts the public key from the private key file (private\_key.pem) and saves it to a new file named public\_key.pem.

**Generating a Digital signature of the infosec.txt file**



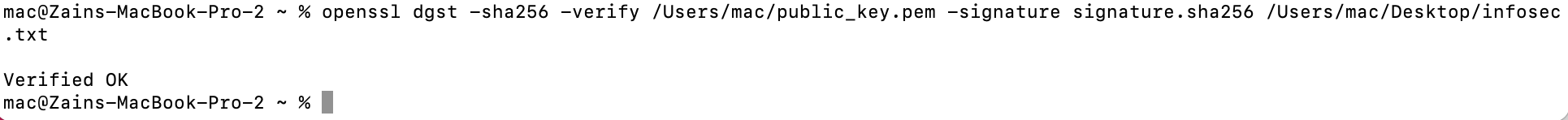
Now a digital signature is created and let’s verify it using the command

**openssl dgst -sha256 -verify /Users/mac/public\_key.pem -signature signature.sha256 /Users/mac/Desktop/infosec.txt**

openssl dgst: This command computes the hash digest of a file.

-sha256: Specifies the SHA-256 hash algorithm, which is used to generate the hash digest.

-signature signature.sha256: Specifies the path to the file containing the digital signature (signature.sha256).The digital signature is a cryptographic representation of the hash digest of the original file.

****

We can see that we got the “Verified OK” output which confirms the file integrity.