

Task 1: RSA key generation

Scenario:

As a cybersecurity engineer at DataSafe Bank, you are tasked with securing communication channels between the bank's internal systems and its customers. To achieve this, you must generate RSA keys that will be used for encrypting and decrypting sensitive information transmitted over insecure channels.

Task:

Generate a 2048-bit RSA private key and its corresponding public key. Save the private key as `private_key.pem` and the public key as `public_key.pem`.

Step 1: Generate an RSA private key

Hint:

To generate the RSA private key:

- Use the OpenSSL `genpkey` command.
- Specify the algorithm as RSA and set the key length to 2048 bits.

Command:

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048
```

Expected output:

- `private_key.pem` file containing the RSA private key.

Action required:

Capture a screenshot of the command you used to generate the terminal's private key and confirmation output.



Step 2: Extract the RSA public key from the private key

Hint:

To extract the RSA public key from the private key:

- Use the OpenSSL rsa command.
- Use the -pubout option to specify that the output should be the public key.

Command:

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Expected output:

- public_key.pem file containing the RSA public key.

Action required:

Capture a screenshot of the command you used to extract the public key and the confirmation output in the terminal.

Sample output:

```
theia@theia-manishkr: /home/project x
theia@theia-manishkr:/home/project$ openssl rsa -pubout -in private_key.pem -out public_key.pem
writing RSA key
theia@theia-manishkr:/home/project$ ls -l
total 20
-rw-r--r-- 1 theia users 901 Dec 5 10:55 contract.txt
-rw-r--r-- 1 theia users 137 Dec 5 10:55 customer_data.txt
-rw-r--r-- 1 theia users 1704 Dec 5 10:56 private_key.pem
-rw-r--r-- 1 theia users 451 Dec 5 10:57 public_key.pem
-rw-r--r-- 1 theia users 200 Dec 5 10:55 sensitive_data.txt
theia@theia-manishkr:/home/projects$
```

Task 2: RSA key encryption and decryption

Scenario:

Once the RSA keys have been generated, you must encrypt sensitive customer data (such as account numbers) to ensure secure transmission. The customer data should be encrypted with the public key, and the recipient (bank server) should decrypt it using the private key.

Task:

Using the RSA keys generated in Task 1, encrypt a sensitive customer data file (customer_data.txt) and decrypt it back to its original form.

Step 1: Encrypt customer data using the public key

Hint:

To encrypt the customer data:

- Use the OpenSSL `pkeyutl` command with the `-encrypt` option.
- Specify the input file as `customer_data.txt` and the output file as `encrypted_data.bin`.
- Use the `-pubin` option to indicate that the key is a public key.

```
openssl pkeyutl -encrypt -inkey public_key.pem -pubin -in customer_data.txt -out encrypted_data.bin
```

Expected output:

- encrypted_data.bin file containing the encrypted data.

Action required:

Capture a screenshot of the command you used to encrypt the customer data and the confirmation output in the terminal.

Sample output:

```
theia@theia-manishkr:/home/project x
```

```
theia@theia-manishkr:/home/project$ openssl pkeyutl -encrypt -inkey public_key.pem -pubin -in customer_data.txt -out encrypted_data.bin
```

```
theia@theia-manishkr:/home/project$ cat encrypted_data.bin
```

```
.000000E@Tx000  
D0000000  
98000r,0|0000W00^000000000000.NR000000000000  
#000Dw0000000000U000w000  
AjT6SmY00X000X;00000g0000qRT000c2A4\00h00Q00xJ00Lg0000Sta0
```

```
theia@theia-manishkr:/home/projects$
```

Step 2: Decrypt the data using the private key

Hint:

To decrypt the encrypted data:

- Use the OpenSSL pkeyutl command with the -decrypt option.
- Specify the input file as encrypted_data.bin and the output file as decrypted_data.txt.

```
openssl pkeyutl -decrypt -inkey private_key.pem -in encrypted_data.bin -out decrypted_data.txt
```

Expected output:

- decrypted_data.txt file containing the decrypted data should match the original customer_data.txt.

Action required:

Capture a screenshot of the command you used to decrypt the customer data and the confirmation output in the terminal.

Sample output:A screenshot of a terminal window with a title bar showing 'theia@theia-manishkr: /home/project'. The terminal displays the following commands and output:

```
theia@theia-manishkr:/home/projects$ openssl pkeyutl -decrypt -inkey private_key.pem -in encrypted_data.bin -out decrypted_data.txt
theia@theia-manishkr:/home/projects$ cat decrypted_data.txt
CustomerID: 12345
Name: John Doe
Account Number: 9876543210
Balance: $15,230.45
Email: john.doe@example.com
Phone: +1-555-678-9101
theia@theia-manishkr:/home/projects$
```

Task 3: AES encryption**Scenario:**

DataSafe Bank must securely store sensitive data (for example, transaction logs, user passwords) in its database. To ensure confidentiality, this data is encrypted using AES encryption before storage. The encrypted data can only be decrypted using the correct AES key.

Task:

Encrypt a sensitive data file (sensitive_data.txt) using AES-256 encryption and decrypt it back using the AES key.

Step 1: Encrypt data using AES-256-CBC**Hint:**

To encrypt the data using AES-256 encryption:

- Use the OpenSSL enc command with the -aes-256-cbc option to specify the AES-256-CBC algorithm.
- Use the -salt option to add random data (salt) to make encryption more secure.

- Provide the input file (sensitive_data.txt) and specify the output file (encrypted_data_aes.bin).
- Use a password for encryption with the -pass pass:securepassword option.

Command:

```
openssl enc -aes-256-cbc -salt -in sensitive_data.txt -out encrypted_data_aes.bin -pass pass:securepassword
```

Expected output:

- encrypted_data_aes.bin file containing the AES-encrypted data.

Action required:

Capture a screenshot of the command you used to encrypt the data and the confirmation output in the terminal.

Sample output:



```
theia@theia-manishkr:/home/project$ openssl enc -aes-256-cbc -salt -in sensitive_data.txt -out encrypted_data_aes.bin -pass pass:securepassword
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
theia@theia-manishkr:/home/project$ cat encrypted_data_aes.bin
Salted__
[Hexadecimal data follows]
```

Step 2: Decrypt data using the AES key

Hint:

To decrypt the data using the AES key:

- Use the OpenSSL enc command with the -d option for decryption.
- Specify the encrypted input file (encrypted_data_aes.bin) and the output file for the decrypted data (decrypted_data_aes.txt).
- Use the same password used for encryption to decrypt the data.

Command:

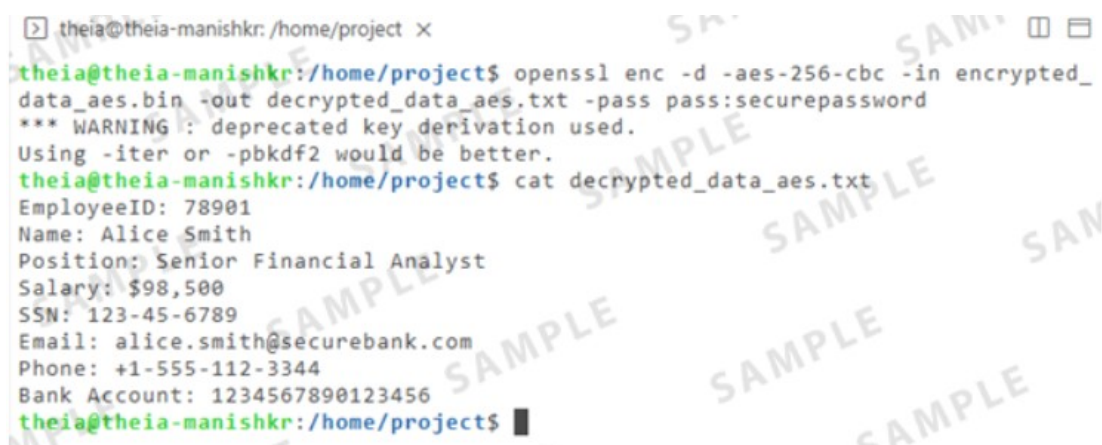
```
openssl enc -d -aes-256-cbc -in encrypted_data_aes.bin -out decrypted_data_aes.txt -pass  
pass:securepassword
```

Expected output:

- decrypted_data_aes.txt file containing the decrypted data should match the original sensitive_data.txt.

Action required:

Capture a screenshot of the command you used to decrypt the data and the confirmation output in the terminal.

Sample output:

```
theia@theia-manishkr: /home/project X  
theia@theia-manishkr:/home/project$ openssl enc -d -aes-256-cbc -in encrypted_  
data_aes.bin -out decrypted_data_aes.txt -pass pass:securepassword  
*** WARNING : deprecated key derivation used.  
Using -iter or -pbkdf2 would be better.  
theia@theia-manishkr:/home/project$ cat decrypted_data_aes.txt  
EmployeeID: 78901  
Name: Alice Smith  
Position: Senior Financial Analyst  
Salary: $98,500  
SSN: 123-45-6789  
Email: alice.smith@securebank.com  
Phone: +1-555-112-3344  
Bank Account: 1234567890123456  
theia@theia-manishkr:/home/project$
```

Task 4: Digital signature creation**Scenario:**

DataSafe Bank needs to ensure the integrity and authenticity of documents (for example, contracts and agreements) transmitted over email. You will create a digital signature using RSA, allowing the recipient to verify the document's integrity and origin.

Task:

Create a digital signature for a document (contract.txt) using your RSA private key. Verify the signature using the corresponding public key.

Step 1: Generate a digital signature for the document**Hint:**

To generate a digital signature:

- Use the OpenSSL dgst command with the -sha256 option to specify the SHA-256 hashing algorithm.
- Create the digital signature with the -sign option with the private key.
- Specify the document (contract.txt) and the output file for the signature (document_signature.bin).

Command:

```
openssl dgst -sha256 -sign private_key.pem -out document_signature.bin contract.txt
```

Expected output:

- document_signature.bin file containing the digital signature.

Action required:

Capture a screenshot of the command you used to generate the digital signature and the confirmation output in the terminal.

Sample output:

[illegible]

Step 2: Verify the digital signature

Hint:

To verify the digital signature:

- Use the OpenSSL dgst command with the -sha256 option for the SHA-256 algorithm.
- Use the -verify option with the public key to verify the signature.
- Specify the digital signature file (document_signature.bin) and the original document (contract.txt).

Command:

```
openssl dgst -sha256 -verify public_key.pem -signature document_signature.bin contract.txt
```

Expected output:

- Verification result: "Verified OK" or "Verification Failure".

Action required:

Capture a screenshot of the command you used to verify the terminal's digital signature and confirmation output.

Sample output:


```
theia@theia-manishkr: /home/project x
theia@theia-manishkr:/home/project$ openssl dgst -sha256 -verify public_key.pem -signature document_signature.bin contract.txt
Verified OK
theia@theia-manishkr:/home/project$ █
```

Task 5: Diffie-Hellman key exchange

Scenario:

DataSafe Bank must establish a secure, shared secret between two parties (an employee and a bank server) without transmitting the secret over the network. To achieve this, the Diffie-Hellman key exchange protocol will be used.

Task:

Implement Diffie-Hellman key exchange to securely exchange keys between two parties (Alice and Bob)

Step 1: Generate DH parameters and generate keys

Hint:

To generate the Diffie-Hellman parameters:

- Use the OpenSSL dhparam command.
- Specify the key length (2048 bits) to generate secure parameters.
- Generate public and private keys for Alice and Bob

Commands:

- **Generate DH parameters:**

```
openssl dhparam -out dhparam.pem 2048
```

- **Generate Alice's private key:**

```
openssl genpkey -paramfile dhparam.pem -out alice_private.pem
```

- **Generate Alice's public key:**

```
openssl pkey -in alice_private.pem -pubout -out alice_public.pem
```

- **Generate Bob's private key:**

```
openssl genpkey -paramfile dhparam.pem -out bob_private.pem
```

- **Generate Bob's public key:**

```
openssl pkey -in bob_private.pem -pubout -out bob_public.pem
```


Expected outputs:

- dhparam.pem file containing the DH parameters.
- Command and outputs for the public and private keys for both parties

Action required:

Capture a screenshot of the command you used to generate the DH parameters and the confirmation of output, along with the keys for both parties.

Sample output:

```
theia@theia-manishkr:/home/project x
theia@theia-manishkr:/home/project$ # Alice and Bob exchange public keys and compute shared secret
openssl pkeyutl -derive -inkey alice_private.pem -peerkey bob_public.pem -out alice_shared_secret.bin
openssl pkeyutl -derive -inkey bob_private.pem -peerkey alice_public.pem -out bob_shared_secret.bin

theia@theia-manishkr:/home/project$ ls -l
total 68
-rw-r--r-- 1 theia users 806 Dec 5 11:09 alice_private.pem
-rw-r--r-- 1 theia users 800 Dec 5 11:09 alice_public.pem
-rw-r--r-- 1 theia users 256 Dec 5 11:12 alice_shared_secret.bin
-rw-r--r-- 1 theia users 806 Dec 5 11:09 bob_private.pem
-rw-r--r-- 1 theia users 800 Dec 5 11:10 bob_public.pem
-rw-r--r-- 1 theia users 256 Dec 5 11:12 bob_shared_secret.bin
-rw-r--r-- 1 theia users 901 Dec 5 10:55 contract.txt
-rw-r--r-- 1 theia users 137 Dec 5 10:55 customer_data.txt
-rw-r--r-- 1 theia users 137 Dec 5 10:59 decrypted_data.txt
-rw-r--r-- 1 theia users 200 Dec 5 11:03 decrypted_data_aes.txt
-rw-r--r-- 1 theia users 424 Dec 5 11:09 dhparam.pem
-rw-r--r-- 1 theia users 256 Dec 5 11:04 document_signature.bin
-rw-r--r-- 1 theia users 256 Dec 5 10:58 encrypted_data.bin
-rw-r--r-- 1 theia users 224 Dec 5 11:01 encrypted_data_aes.bin
-rw-r--r-- 1 theia users 1704 Dec 5 10:56 private_key.pem
-rw-r--r-- 1 theia users 451 Dec 5 10:57 public_key.pem
-rw-r--r-- 1 theia users 200 Dec 5 10:55 sensitive_data.txt
theia@theia-manishkr:/home/project$
```

Step 2: Compute and verify the shared secret keys

Hint:

To generate the private and public keys for Alice and Bob, and then compute the shared secret:

- Alice and Bob exchange public keys, and each computes the shared secret using their private and public keys.

Commands:

Compute the shared secret for Alice:

```
openssl pkeyutl -derive -inkey alice_private.pem -peerkey bob_public.pem -out
alice_shared_secret.bin
```

Compute the shared secret key for Bob:

```
openssl pkeyutl -derive -inkey bob_private.pem -peerkey alice_public.pem -out
bob_shared_secret.bin
```

Expected output:

- alice_shared_secret.bin and bob_shared_secret.bin files containing the shared secret (both should match).

Action required:

Capture a screenshot that displays the shared secret keys. The screenshot should include the successful output showing that both shared secret keys match.

Sample output:

```
theia@theia-manishkr:/home/project $
theia@theia-manishkr:/home/project$ # Alice and Bob exchange public keys and compute shared secret
openssl pkeyutl -derive -inkey alice_private.pem -peerkey bob_public.pem -out alice_shared_secret.bin
openssl pkeyutl -derive -inkey bob_private.pem -peerkey alice_public.pem -out bob_shared_secret.bin

theia@theia-manishkr:/home/project$ ls -l
total 68
-rw-r--r-- 1 theia users 806 Dec 5 11:09 alice_private.pem
-rw-r--r-- 1 theia users 800 Dec 5 11:09 alice_public.pem
-rw-r--r-- 1 theia users 256 Dec 5 11:12 alice_shared_secret.bin
-rw-r--r-- 1 theia users 806 Dec 5 11:09 bob_private.pem
-rw-r--r-- 1 theia users 800 Dec 5 11:10 bob_public.pem
-rw-r--r-- 1 theia users 256 Dec 5 11:12 bob_shared_secret.bin
-rw-r--r-- 1 theia users 901 Dec 5 10:55 contract.txt
-rw-r--r-- 1 theia users 137 Dec 5 10:55 customer_data.txt
-rw-r--r-- 1 theia users 137 Dec 5 10:59 decrypted_data.txt
-rw-r--r-- 1 theia users 200 Dec 5 11:03 decrypted_data_aes.txt
-rw-r--r-- 1 theia users 424 Dec 5 11:09 dhparam.pem
-rw-r--r-- 1 theia users 256 Dec 5 11:04 document_signature.bin
-rw-r--r-- 1 theia users 256 Dec 5 10:58 encrypted_data.bin
-rw-r--r-- 1 theia users 224 Dec 5 11:01 encrypted_data_aes.bin
-rw-r--r-- 1 theia users 1704 Dec 5 10:56 private_key.pem
-rw-r--r-- 1 theia users 451 Dec 5 10:57 public_key.pem
-rw-r--r-- 1 theia users 200 Dec 5 10:55 sensitive_data.txt
theia@theia-manishkr:/home/project$ cmp alice_shared_secret.bin bob_shared_secret.bin
theia@theia-manishkr:/home/project$ sha256sum alice_shared_secret.bin
sha256sum bob_shared_secret.bin
967497ad1b020cce84cef248ee3cac05bcd32362afe2161cfc096db847cc7f69  alice_shared_secret.bin
967497ad1b020cce84cef248ee3cac05bcd32362afe2161cfc096db847cc7f69  bob_shared_secret.bin
theia@theia-manishkr:/home/project$
```

Task 6: Public Key Infrastructure (PKI) setup

Scenario:

DataSafe Bank requires the setup of a Certificate Authority (CA) to sign and issue certificates for internal services and servers. You are tasked with creating a root certificate and signing a server certificate for datasafebank.com.

Task:

Set up a Certificate Authority (CA) using OpenSSL. Generate a root certificate and a signed certificate for datasafebank.com.

Step 1: Generate the root private key and create the root certificate

Hint:

To generate the root private key:

- Use the OpenSSL genrsa command.
- Specify the key length (2048 bits) for security.

To create the root certificate:

- Use the OpenSSL req command with the -x509 option to generate a self-signed certificate.
- Specify the root private key (ca_private_key.pem) and set the certificate validity (3650 days).

Command:

- To generate the root private key:

```
openssl genrsa -out ca_private_key.pem 2048
```

- To create root certificate:

```
openssl req -x509 -new -key ca_private_key.pem -out ca_cert.pem -days 3650
```

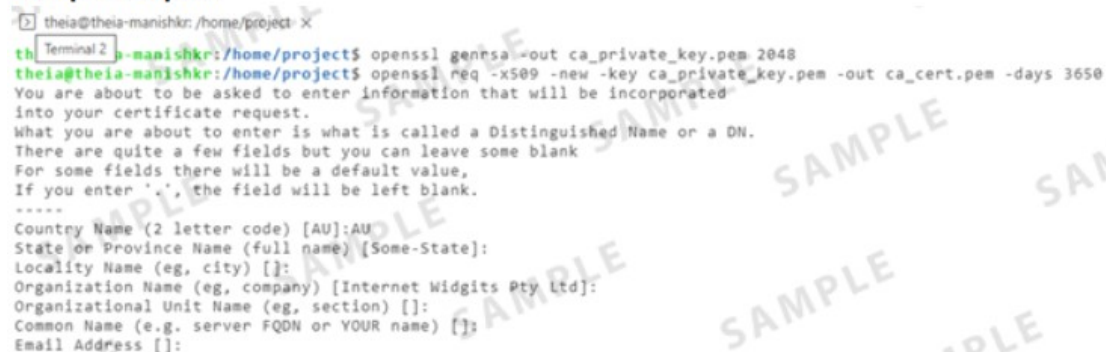
Expected output:

- ca_private_key.pem file containing the root private key.
- ca_cert.pem file containing the root certificate.

Action required:

Capture a screenshot of the command you used to generate the root private key and the confirmation output in the terminal, along with the command you used to create the terminal's root certificate and confirmation output.

Sample output:



```
theia@theia-manishkr:/home/project$ openssl genrsa -out ca_private_key.pem 2048
theia@theia-manishkr:/home/project$ openssl req -x509 -new -key ca_private_key.pem -out ca_cert.pem -days 3650
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]:AU
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

Step 2: Generate a CSR for the server

Hint:

To generate the certificate signing request (CSR) for the server:

- Use the OpenSSL req command with the -new option.
- Specify the private key for datasafebank.com and the output CSR file (datasafebank_csr.csr).

Command:

```
openssl req -new -key private_key.pem -out datasafebank_csr.csr
```

Expected output:

- datasafebank_csr.csr file containing the server's CSR.

Action required:

Capture a screenshot of the command you used to generate the CSR and the confirmation output in the terminal.

Sample output:

```
theia@theia-manishkr:/home/project$ openssl req -new -key private_key.pem -out securebank_csr.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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:manish
An optional company name []:
theia@theia-manishkr:/home/project$
```

Step 3: Sign the CSR with the root certificate**Hint:**

To sign the CSR using the root certificate:

- Use the OpenSSL x509 command with the -req option to sign the CSR.
- Specify the root certificate (ca_cert.pem) and the root private key (ca_private_key.pem).
- Set the server certificate's validity period (for example , 365 days).

Command:

```
openssl x509 -req -in datasafebank_csr.csr -CA ca_cert.pem -CAkey ca_private_key.pem -
CAcreateserial -out datasafebank_cert.pem -days 365
```

Expected output:

- datasafebank_cert.pem file containing the signed server certificate.

Action required:

Capture a screenshot of the command you used to sign the CSR and the confirmation output in the terminal.

Sample output:

```
theia@theia-manishkr: /home/project x
theia@theia-manishkr:/home/project$ openssl x509 -req -in securebank_csr.csr -CA ca_cert.pem -
CAkey ca_private_key.pem -CAcreateserial -out securebank_cert.pem -days 365
Certificate request self-signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
theia@theia-manishkr:/home/project$
```

Task 7: Hashing with SHA-256

Scenario:

DataSafe Bank needs to store passwords securely. Instead of storing passwords in plain text, you will hash them using SHA-256 to ensure they are stored securely.

Task:

Hash a password using SHA-256 and verify its integrity.

Step 1: Hash the password using SHA-256

Hint:

To hash the password using SHA-256:

- Use the OpenSSL dgst command with the -sha256 option to create a SHA-256 hash.
- Specify the input password file (password.txt) and the output file for the hashed password (hashed_password.txt).

Command:

```
openssl dgst -sha256 -out hashed_password.txt password.txt
```

Expected output:

- hashed_password.txt file containing the SHA-256 hash of the password.

Action required:

Capture a screenshot of the command you used to hash the password and the confirmation output in the terminal.

Sample output:

```
theia@theia-manishkr: /home/project x
theia@theia-manishkr: /home/project$ openssl dgst -sha256 -out hashed_password.txt password.txt
theia@theia-manishkr: /home/project$ cat hashed_password.txt
SHA2-256(password.txt)= e328c171147ef2447fad5e3f2362ddca4f960a9871dd48665a26ca9b274568ed
theia@theia-manishkr: /home/project$
```

Step 2: Verify the hash

Hint:

To verify the integrity of the hashed password:

- Use the OpenSSL dgst command again to check if the hash matches the original file.
- Ensure that the -verify option and the appropriate public key (public_key.pem) are used for verification.

Command:

```
openssl dgst -sha256 -verify public_key.pem -signature hashed_password.txt password.txt
```

Expected output:

- The original and verified hash values.

Action required:

Capture a screenshot of the command you used to verify the terminal's hash and confirmation output.

Sample output:

```
theia@theia-manishkr: /home/project$ # Generate the hash again
openssl dgst -sha256 password.txt

# Compare with the content of hashed_password.txt
cat hashed_password.txt
SHA2-256(password.txt)= e328c171147ef2447fad5e3f2362ddca4f960a9871dd48665a26ca9b274568ed
SHA2-256(password.txt)= e328c171147ef2447fad5e3f2362ddca4f960a9871dd48665a26ca9b274568ed
theia@theia-manishkr: /home/project$
```

Task 8: Digital certificate generation

Scenario:

DataSafe Bank needs to issue digital certificates to its employees and internal systems for secure communication. You will use OpenSSL to generate a CSR (Certificate Signing Request) and sign it with the root certificate.

Task:

Create a CSR for a server and generate a self-signed certificate.

Step 1: Generate a private key for the server

Hint:

To generate the private key for the server:

- Use the OpenSSL genpkey command with the -algorithm RSA option.
- Specify the key size (2048 bits).

Command:

```
openssl genpkey -algorithm RSA -out server_private_key.pem -pkeyopt rsa_keygen_bits:2048
```

Expected output:

- `server_private_key.pem` file containing the private key for the server.

Action required:

Please capture a screenshot of the command you used to generate the server's private key and the confirmation output in the terminal.

Sample output:

[illegible]

Step 2: Create a CSR

Hint:

To create the CSR for the server:

- Use the OpenSSL req command with the -new option.
- Specify the private key (server_private_key.pem) and the output CSR file (server_csr.csr).

Command:

1. `openssl req -new -key server_private_key.pem -out server_csr.csr`

Copied!

Expected output:

- server_csr.csr file containing the server's CSR.

Action required:

Capture a screenshot of the command you used to create the CSR and the confirmation output in the terminal.

Sample output:

```
theia@theia-manishkr: /home/project X
theia@theia-manishkr: /home/project$ openssl req -new -key server_private_key.pem -out server_csr.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]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:San Francisco
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SecureBank Inc.
Organizational Unit Name (eg, section) []:IT Department
Common Name (e.g. server FQDN or YOUR name) []:www.securebank.com
Email Address []:admin@securebank.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:manish
An optional company name []:SecureBank
theia@theia-manishkr: /home/project$
```

Step 3: Create a self-signed certificate

Hint:

To create a self-signed certificate:

- Use the OpenSSL x509 command with the -req option to sign the CSR using the server's private key.
- Specify the certificate's validity period (for example, 365 days).

Command:

```
openssl x509 -req -in server_csr.csr -signkey server_private_key.pem -out server_cert.pem -days 365
```

Expected output:

- server_cert.pem file containing the self-signed certificate.

Action required:

Capture a screenshot of the command you used to create the self-signed certificate and the confirmation output in the terminal.

Sample output:

```
heja@heja-manishkr:/home/project$ openssl x509 -req -in server_csr.csr -signkey server_private_key.pem -out server_cert.pem -days 365
Certificate request self-signature ok
subject=C = US, ST = California, L = San Francisco, O = SecureBank Inc., OU = IT Department, CN = www.securebank.com, emailAddress = admin@securebank.com
heja@heja-manishkr:/home/project$
```