OpenSSL Lab: RSA Encryption and Decryption

OBJECTIVE

This lab provides a step-by-step guide to generating RSA keys, encrypting a text file, and decrypting it using OpenSSL. Each command is explained in detail for better understanding.

Be Ready:

```bash
mkdir Desktop/rsa-lab
```

```bash
cd Desktop/rsa-lab
```

1 Generating a Sample Text File

Command:

```bash
echo "This is my secret message for the RSA lab." > secret.txt
```

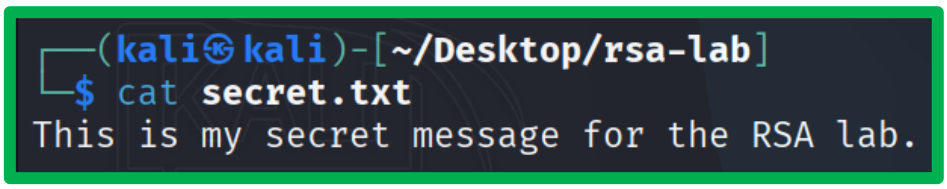
Explanation:

- **echo**: Prints text to the terminal.
- **> secret.txt**: Redirects the output into a file named secret.txt.
- This creates a plaintext file that we will later encrypt.

Verification:

```bash
cat secret.txt
```

(Should display the secret message.)



2 Generate an RSA Private Key

Command (Modern OpenSSL):

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

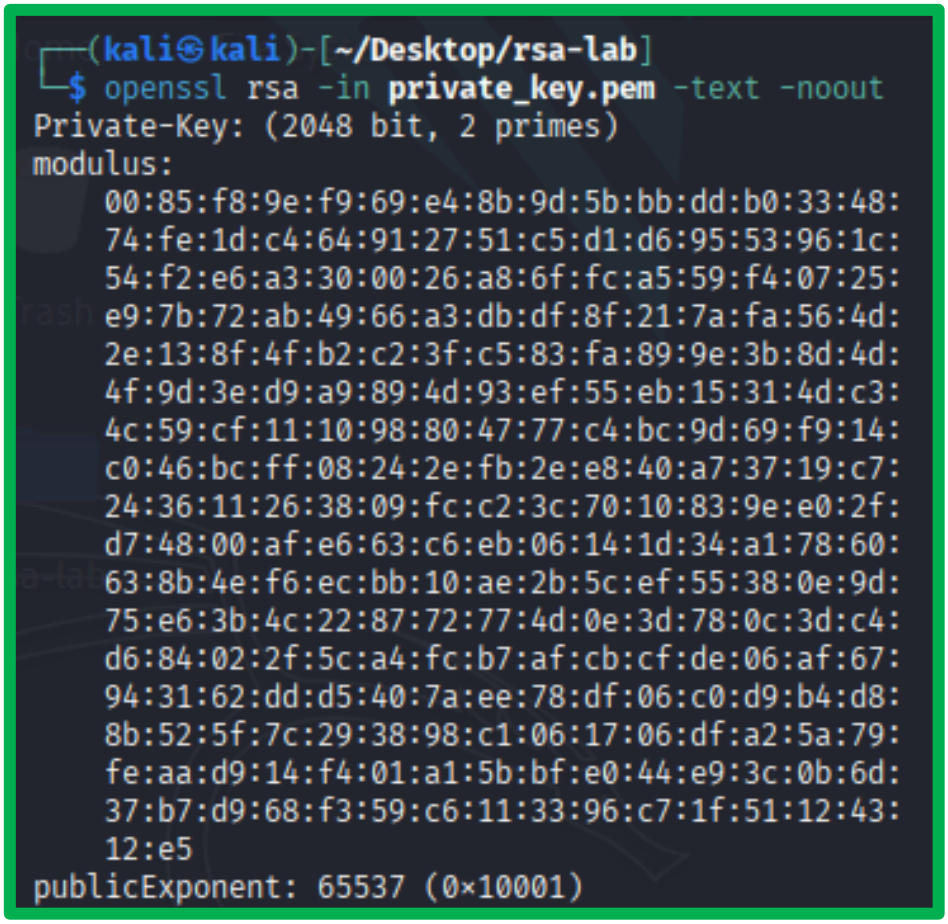
Explanation:

- **genpkey**: Generates a private key (preferred in newer OpenSSL versions).
- **-algorithm RSA**: Specifies the RSA algorithm.
- **-out private_key.pem**: Saves the key in PEM format to private_key.pem.
- **-pkeyopt rsa_keygen_bits:2048**: Sets the key size to 2048 bits (secure for most use cases).
- If prompted, enter a passphrase to encrypt the private key (recommended for security).

Viewing the Private Key:

```bash
openssl rsa -in private_key.pem -text -noout
```

(Shows key details like modulus, exponents, and primes.)



3 Extract the Public Key from the Private Key

Command:

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

Explanation:

- **rsa**: Processes RSA keys.
- **-pubout**: Extracts the public key.
- **-in private_key.pem**: Reads the private key.
- **-out public_key.pem**: Saves the public key in public_key.pem.

Viewing the Public Key:

```bash
cat public_key.pem
```

(Shows the public key in PEM format.)

```
┌──(kali㉿kali)-[~/Desktop/rsa-lab]
└─$ cat public_key.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhfie+Wnki51bu92wM0h0
/h3EZJEnUcXR1pVTlhxU8uajMAAmqG/8pVn0ByXpe3KrSWaj29+PIXr6Vk0uE49P
ssI/xYP6iZ47jU1PnT7ZqYlNk+9V6xUxTcNMWc8REJiAR3fEvJ1p+RTARrz/CCQu
+y7oQKc3GcckNhEmOAn8wjxwEIOe4C/XSACv5mPG6wYUHTSheGBji0727LsQritc
71U4Dp115jtMIodyd00OPXgMPcTWhAIvXKT8t6/Lz94Gr2eUMWLd1UB67njfBsDZ
tNiLUl98KTiYwQYXBt+iWnn+qtkU9AGhW7/gROk8C203t9lo81nGETOWxx9REkMS
5QIDAQAB
-----END PUBLIC KEY-----
```

4 Encrypt the File Using the Public Key

Command:

```bash
openssl rsautl -encrypt -inkey public_key.pem -pubin -in secret.txt -out encrypted.enc
```

Explanation:

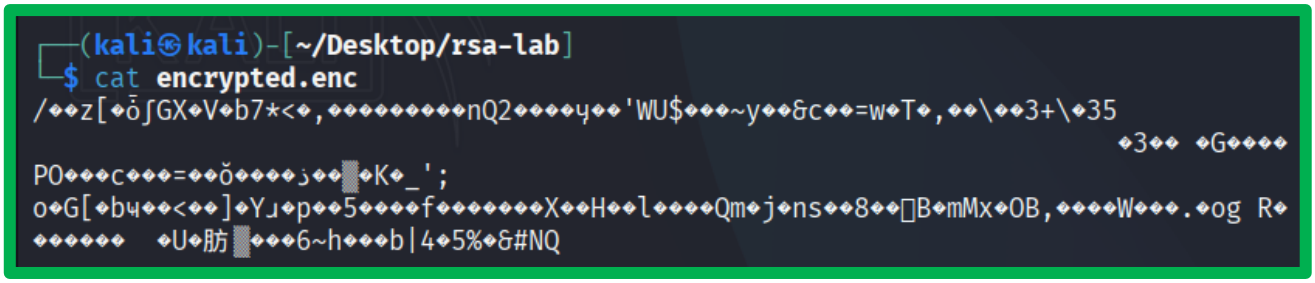
- **rsautl**: Performs RSA encryption/decryption.
- **-encrypt**: Encrypts the input file.
- **-inkey public_key.pem**: Uses the public key for encryption.
- **-pubin**: Indicates that the input key is a public key.
- **-in secret.txt**: The file to encrypt.
- **-out encrypted.enc**: Outputs the encrypted file in binary format.

Limitation:

- RSA can only encrypt small files (max ~245 bytes for 2048-bit keys).
- For larger files, use hybrid encryption (AES for data + RSA for the key).

Verification:

```bash
cat encrypted.enc
```



5 Decrypt the File Using the Private Key

Command:

```bash
openssl rsautl -decrypt -inkey private_key.pem -in encrypted.enc -out decrypted.txt
```

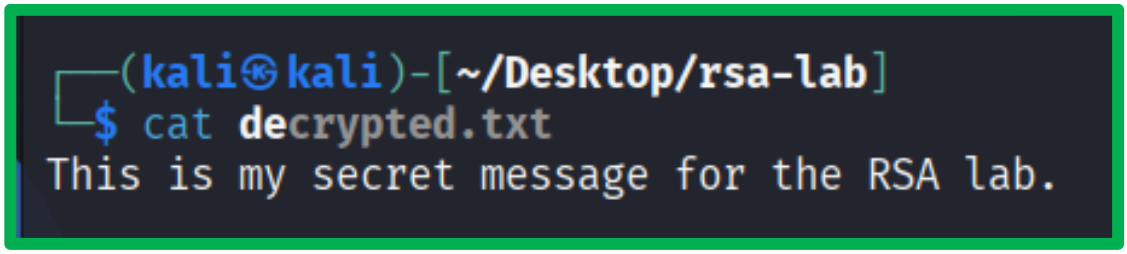
Explanation:

- **-decrypt**: Decrypts the input file.
- **-inkey private_key.pem**: Uses the private key (must match the public key used for encryption).
- **-in encrypted.enc**: The encrypted file.
- **-out decrypted.txt**: Restores the original message.

Verification:

```bash
cat decrypted.txt
```

(Should match secret.txt.)



6 Bonus: Signing and Verification

Why Sign?

- Ensures authenticity (the file was not altered).
- Proves the file came from the private key owner.

Step 1: Create a Digital Signature

```bash
openssl dgst -sha256 -sign private_key.pem -out signature.bin secret.txt
```

Explanation:

- **dgst**: Computes a hash digest.
- **-sha256**: Uses SHA-256 for hashing.
- **-sign private_key.pem**: Signs the hash with the private key.
- **-out signature.bin**: Saves the signature in binary format.

Step 2: Verify the Signature

```bash
openssl dgst -sha256 -verify public_key.pem -signature signature.bin secret.txt
```

Explanation:

- **-verify public_key.pem**: Uses the public key to verify.
- **-signature signature.bin**: The signature file.
- **secret.txt**: The original file.

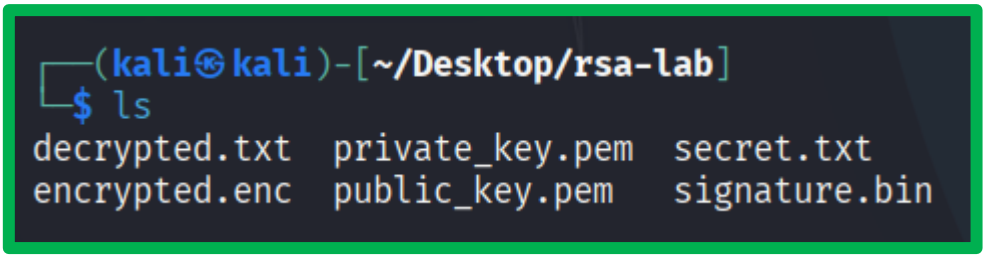
Expected Output:

```
┌──(kali㉿kali)-[~/Desktop/rsa-lab]
└─$ openssl dgst -sha256 -verify public_key.pem -signature signature.bin secret.txt
Verified OK
```

(If tampered, it will say "Verification Failure".)

Summary of Files Generated

	File	Purpose
1	secret.txt	Original plaintext file
2	private_key.pem	RSA private key (keep secret!)
3	public_key.pem	RSA public key (can be shared)
4	encrypted.enc	Encrypted version of secret.txt
5	decrypted.txt	Decrypted file (should match secret.txt)
6	signature.bin	Digital signature for authentication



Lab Submission

Students must submit:

- Screenshots of key commands & outputs.

Prepared by:

Marwa M. El-Azab
Cybersecurity Research Assistant (MSc Candidate), Teaching Assistant at AASTMT

Omar M. El-Azab
Cybersecurity Instructor, Technical Advisor, Adjunct Teaching Assistant at AASTMT