

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.)

```
(kali@kali)-[~/Desktop/rsa-lab]  
$ cat secret.txt  
This is my secret message for the RSA lab.
```

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.)

```
(kali@kali)-[~/Desktop/rsa-lab]
$ openssl rsa -in private_key.pem -text -noout
Private-Key: (2048 bit, 2 primes)
modulus:
 00:85:f8:9e:f9:69:e4:8b:9d:5b:bb:dd:b0:33:48:
 74:fe:1d:c4:64:91:27:51:c5:d1:d6:95:53:96:1c:
 54:f2:e6:a3:30:00:26:a8:6f:fc:a5:59:f4:07:25:
 e9:7b:72:ab:49:66:a3:db:df:8f:21:7a:fa:56:4d:
 2e:13:8f:4f:b2:c2:3f:c5:83:fa:89:9e:3b:8d:4d:
 4f:9d:3e:d9:a9:89:4d:93:ef:55:eb:15:31:4d:c3:
 4c:59:cf:11:10:98:80:47:77:c4:bc:9d:69:f9:14:
 c0:46:bc:ff:08:24:2e:fb:2e:e8:40:a7:37:19:c7:
 24:36:11:26:38:09:fc:c2:3c:70:10:83:9e:e0:2f:
 d7:48:00:af:e6:63:c6:eb:06:14:1d:34:a1:78:60:
 63:8b:4e:f6:ec:bb:10:ae:2b:5c:ef:55:38:0e:9d:
 75:e6:3b:4c:22:87:72:77:4d:0e:3d:78:0c:3d:c4:
 d6:84:02:2f:5c:a4:fc:b7:af:cb:cf:de:06:af:67:
 94:31:62:dd:d5:40:7a:ee:78:df:06:c0:d9:b4:d8:
 8b:52:5f:7c:29:38:98:c1:06:17:06:df:a2:5a:79:
 fe:aa:d9:14:f4:01:a1:5b:bf:e0:44:e9:3c:0b:6d:
 37:b7:d9:68:f3:59:c6:11:33:96:c7:1f:51:12:43:
 12:e5
publicExponent: 65537 (0x10001)
```

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
+y7oQKc3GcckNhEm0An8wjxwEIOe4C/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
```

```
(kali㉿kali)-[~/Desktop/rsa-lab]
$ cat encrypted.enc
/0z[0jGXVb7*<,0000000nQ2000y0'WU$00~y0&c0=wT0,00\03+\035
0300 0G0000
P000c000=00000j000K0_';
o0G[0b400<00]0YJ0p005000f000000X00H00l0000Qm0j0ns00800B0mMx0OB,000W000.0og R0
000000 0U0肪0006~h000b|405%0&#NQ
```

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.)

```
(kali@kali)-[~/Desktop/rsa-lab]  
$ cat decrypted.txt  
This is my secret message for the RSA lab.
```

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

```
(kali㉿kali)-[~/Desktop/rsa-lab]
$ ls
decrypted.txt  private_key.pem  secret.txt
encrypted.enc  public_key.pem   signature.bin
```

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*