

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Project 03: OpenSSL

Làm quen với OpenSSL

Môn học: Nhập môn mã hoá - mật mã

Sinh viên thực hiện:

Tạ Công Hoàng (21120074)

Giảng viên lý thuyết:

Thầy Nguyễn Đình Thúc

Giảng viên thực hành:

Thầy Nguyễn Văn Quang Huy

Ngày 15 tháng 12 năm 2023

Mục lục

1	Note về bài làm	2
2	Câu 1	2
2.1	Cấu trúc và các thành phần có trong file ‘priv.pem’	2
2.2	Cấu trúc và các thành phần có trong file ‘pub.pem’	2
2.3	Chi tiết về mã nguồn	3
2.4	Video demo	5
3	Câu 2	5
3.1	Mô tả cách OpenSSL mã hoá và giải mã RSA	5
3.1.1	Quá trình mã hoá	5
3.1.2	Quá trình giải mã	7
3.2	Chi tiết về mã nguồn	9
3.3	Video demo	10
4	Câu 3	10
4.1	Quá trình kí sử dụng private key	10
4.2	Quá trình xác nhận sử dụng public key	11
4.3	Chi tiết về mã nguồn	11
4.4	Video demo	13
5	Tài liệu tham khảo	15

1 Note về bài làm

Các tên file hay đường dẫn trong bài làm của em đều được gán cứng a, nên thầy có thể sửa đổi để kiểm tra test case a. Và các file 'priv.pem', 'pub.pem' đều được tạo từ OpenSSL sử dụng các lệnh thầy cung cấp, cũng như file 'cipher' được tạo từ file 'plain' và 'sign' được tạo từ 'mess' thông qua các lệnh thầy cung cấp. Và bài làm của em chỉ đúng với gói lệnh '*pkeyutl*' mặc định mà thầy cung cấp không phải tất cả gói lệnh của OpenSSL.

2 Câu 1

2.1 Cấu trúc và các thành phần có trong file 'priv.pem'

Dựa vào lệnh (được tham khảo ở [OpenSSL Cookbook](#))

```
1 $ openssl rsa -text -noout -in priv.pem
```

ta có cấu trúc có trong file 'priv.pem' (xem thêm ở hình 1) như sau:

- Private-Key: (2048 bit, 2 primes) : cấu trúc của khoá. 2048 là độ dài của khoá và được tạo thành từ 2 số nguyên tố.
- Modulus (n): Đây là tích của hai số nguyên tố lớn p và q ($n = p \times q$).
- Public Exponent (e): Đây là số mũ công khai trong RSA. Nó được sử dụng cùng với modulus để mã hóa dữ liệu. Thông thường, e được chọn là một số nguyên tố lớn như 65537 (theo mặc định của OpenSSL), để tối ưu hóa quá trình mã hóa.
- Private Exponent (d): Số mũ bí mật trong RSA. d được sử dụng cùng với modulus để giải mã dữ liệu. Nó được tính sao cho d là nghịch đảo của e modulo $\phi(n)$, với ϕ là hàm Euler phi.
- Prime1 (p): Một trong hai số nguyên tố lớn được sử dụng để tạo ra modulus n .
- Prime2 (q): Số nguyên tố thứ hai được sử dụng để tạo ra modulus n .
- Exponent1 (dmp1): Đây là $d \bmod (p - 1)$. Nó là một phần của quá trình tối ưu hóa giải mã sử dụng Thuật toán Trung Quốc về phần dư (Chinese Remainder Theorem - CRT).
- Exponent2 (dmq1): Tương tự, dmq1 là $d \bmod (q - 1)$.
- Coefficient (iqmp): Đây là nghịch đảo của q theo modulo p .

và cấu trúc của khoá private-key được biểu diễn dưới chuẩn [PKCS#1](#).

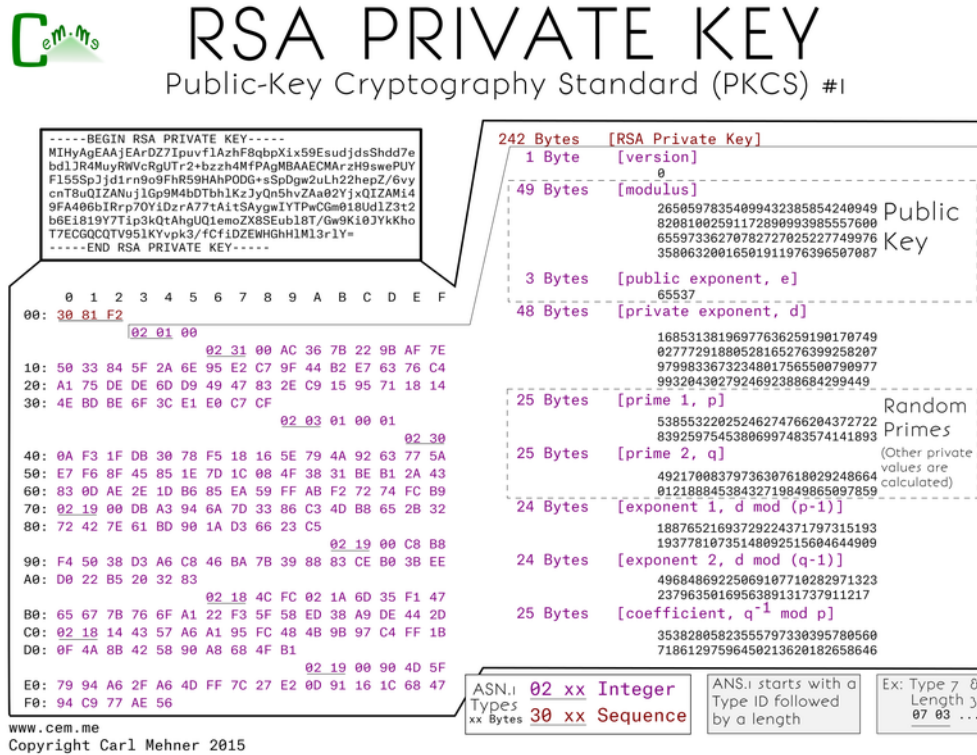
2.2 Cấu trúc và các thành phần có trong file 'pub.pem'

Bằng cách xài lệnh

```
1 $ openssl rsa -pubin -text -noout -in pub.pem
```

ta có cấu trúc có trong file 'pub.pem' (cụ thể ở hình 2) như sau:

- Public-Key: (1024 bit): khoá có độ dài 1024 bit.



Hình 1: Cấu trúc của file 'priv.pem', tham khảo tại [đây](#).

- Modulus (n): tương tự như private key nó là tích của hai số nguyên tố lớn p và q ($n = p \times q$).
- Exponent (e): là số mũ công khai.

và cả 2 (n và e) được tạo từ 'priv.pem' và cũng được biểu diễn dưới dạng chuẩn PKCS#1.

2.3 Chi tiết về mã nguồn

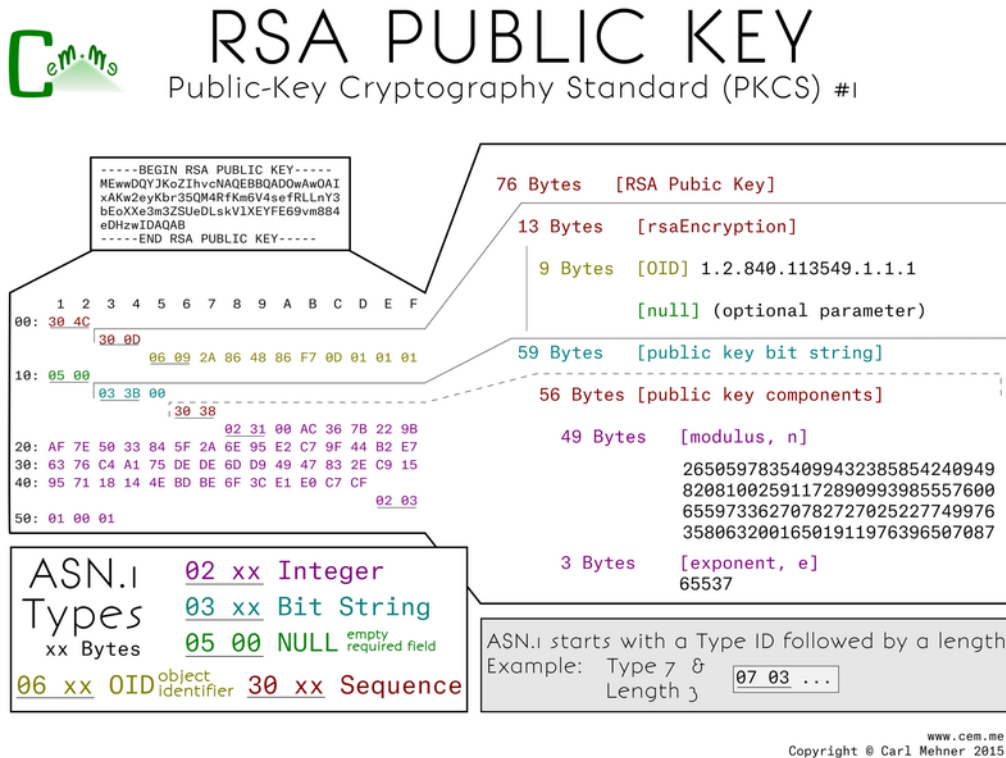
Từ các cấu trúc đọc được ở phần 2.1 và 2.2, em tổ chức chương trình để đọc và xuất ra màn hình các thành phần trong file 'priv.pem' và 'pub.pem' như sau:

- Ngôn ngữ sử dụng: Python.
- Các thư viện cần cài đặt: cryptography (có thể dùng lệnh `$ pip install cryptography` hoặc xem cách cài đặt cụ thể tại [đây](#)).
- Cách biên dịch, cách chạy: dùng lệnh `$ py Source/Cau1/main.py` để chạy chương trình.
- Kết quả sau khi chạy: chương trình sẽ đọc các tệp 'priv.pem' và 'pub.pem' sau đó in ra các thành phần có trong các tệp đó (kết quả được in dưới dạng số thập phân, xem thêm ở hình 3).

Chương trình cụ thể như sau:

```

1 # function to read priv.pem and pub.pem file and return it as binary data
2 def read_pem_file(file_path):
  
```



Hình 2: Cấu trúc của file 'pub.pem', tham khảo tại [đây](#).

```

3  with open(file_path, 'rb') as file:
4      pem_data = file.read()
5      return pem_data
6
7  #function to parse an RSA private key from PEM format and extracts various
   components of the key
8  def parse_private_key(private_key_pem):
9      private_key = serialization.load_pem_private_key(
10         private_key_pem,
11         password=None,
12         backend=default_backend()
13     )
14
15     if isinstance(private_key, rsa.RSAPrivateKey):
16         private_numbers = private_key.private_numbers()
17         return {
18             'p': private_numbers.p,
19             ... # return all component of private key
20         }
21     else:
22         return 'Unknown private key type'
23
24  # function to parse an RSA public key from PEM format and extracts key
   components.
25  def parse_public_key(public_key_pem):
26      public_key = serialization.load_pem_public_key(
27         public_key_pem,
    
```

```

28         backend=default_backend()
29     )
30
31     if isinstance(public_key, rsa.RSAPublicKey):
32         public_numbers = public_key.public_numbers()
33         return {
34             'public_exponent': public_numbers.e,
35             ... # return all component of public key
36         }
37     else:
38         return 'Unknown public key type'
39
40 # Paths to the PEM files, you must change it if your filename or filepath is
41 # different
42 script_dir = os.path.dirname(os.path.abspath(__file__))
43 private_key_path = os.path.join(script_dir, '..', 'priv.pem')
44 public_key_path = os.path.join(script_dir, '..', 'pub.pem')
45
46 # Read and parse the keys
47 try:
48     private_key_pem = read_pem_file(private_key_path)
49     public_key_pem = read_pem_file(public_key_path)
50
51     private_key_info = parse_private_key(private_key_pem)
52     public_key_info = parse_public_key(public_key_pem)
53 except Exception as e:
54     private_key_info = f"Error reading private key: {str(e)}"
55     public_key_info = f"Error reading public key: {str(e)}"
56
57 # Printing the details of the private and public keys to the console
58 print("Private Key Information:")
59 ... # print various details of the private key
60
61 print("\nPublic Key Information:")
62 ... # print various details of the public key

```

2.4 Video demo

Video demo được lưu ở thư mục Demo với tên 'Cau1.mp4' hoặc xem tại [đây](#).

3 Câu 2

3.1 Mô tả cách OpenSSL mã hoá và giải mã RSA

Dựa vào mã nguồn ở [git repository](#) của OpenSSL và các hàm ở file openssl\crypto\rsa\rsa_oss1.c, thì quá trình OpenSSL mã hoá và giải mã như sau:

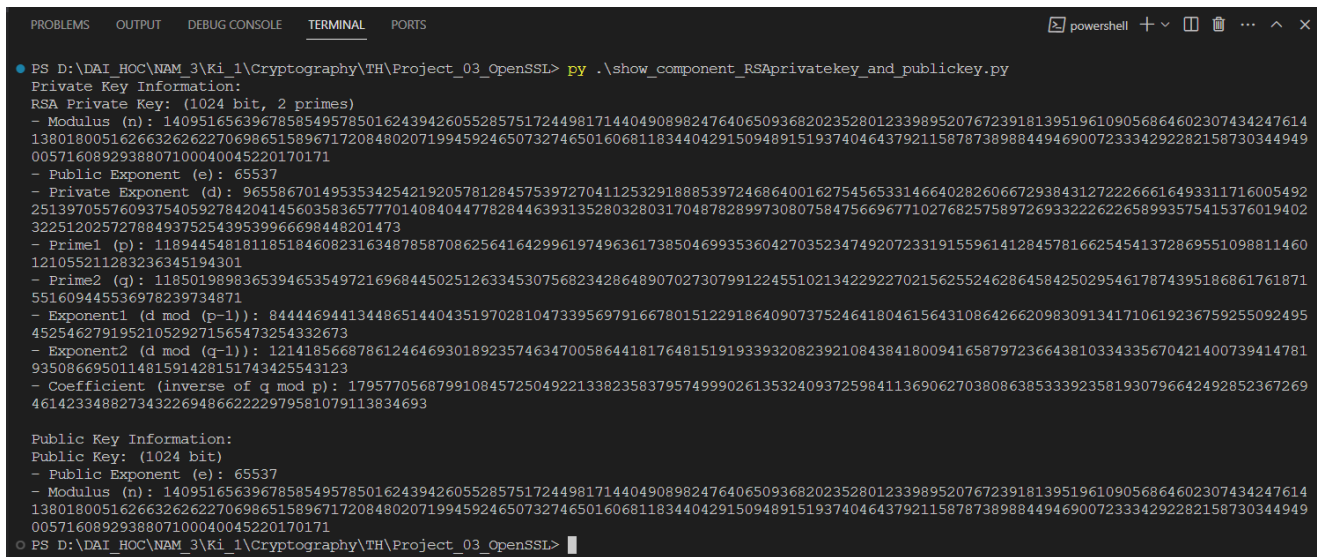
3.1.1 Quá trình mã hoá

OpenSSL mã hoá bằng hàm

```

1 static int rsa_oss1_public_encrypt(int flen, const unsigned char *from, unsigned
    char *to, RSA *rsa, int padding)

```



```
PS D:\DAI_HOC\NAM_3\Ki_1\Cryptography\TH\Project_03_OpenSSL> py .\show_component_RSAPrivatekey_and_publickey.py
Private Key Information:
RSA Private Key: (1024 bit, 2 primes)
- Modulus (n): 1409516563967858549578501624394260552857517244981714404908982476406509368202352801233989520767239181395196109056864602307434247614
1380180051626632626227069865158967172084802071994592465073274650160681183440429150948915193740464379211587873898844946900723334292282158730344949
0057160892938807100040045220170171
- Public Exponent (e): 65537
- Private Exponent (d): 9655867014953534254219205781284575397270411253291888539724686400162754565331466402826066729384312722266616493311716005492
25139705760937540592784204145603583657770140840447782844639313528032803170487828997308075847566967710276825758972693322622658993575415376019402
322512025727884937525439539966698448201473
- Prime1 (p): 11894454818118518460823163487858708625641642996197496361738504699353604270352347492072331915596141284578166254541372869551098811460
121055211283236345194301
- Prime2 (q): 11850198983653946535497216968445025126334530756823428648907027307991224551021342292270215625524628645842502954617874395186861761871
551609445536978239734871
- Exponent1 (d mod (p-1)): 844446944134865144043519702810473395697916678015122918640907375246418046156431086426620983091341710619236759255092495
452546279195210529271565473254332673
- Exponent2 (d mod (q-1)): 121418566878612464693018923574634700586441817648151913933208239210843841800941658797236643810334335670421400739414781
935086695011481591428151743425543123
- Coefficient (inverse of q mod p): 1795770568799108457250492213382358379574999026135324093725984113690627038086385333923581930796642492852367269
46142334882734322694866222979581079113834693

Public Key Information:
Public Key: (1024 bit)
- Public Exponent (e): 65537
- Modulus (n): 1409516563967858549578501624394260552857517244981714404908982476406509368202352801233989520767239181395196109056864602307434247614
1380180051626632626227069865158967172084802071994592465073274650160681183440429150948915193740464379211587873898844946900723334292282158730344949
0057160892938807100040045220170171
PS D:\DAI_HOC\NAM_3\Ki_1\Cryptography\TH\Project_03_OpenSSL>
```

Hình 3: Kết quả sau khi chạy chương trình

trong đó:

- `int flen`: độ dài của dữ liệu đầu vào.
- `const unsigned char *from`: là con trỏ đến dữ liệu đầu vào.
- `unsigned char *to`: là con trỏ đến dữ liệu đầu ra.
- `RSA *rsa`: là cấu trúc chứa thông tin của khoá RSA.
- `int padding`: chỉ định loại padding cho quá trình mã hoá.

Đầu tiên OpenSSL sẽ kiểm tra Modulus (n) và Exponent (e) có hợp lệ hay không, trả về lỗi nếu không hợp lệ. Sau đó, OpenSSL sẽ tiến hành padding dữ liệu dựa vào tham số `int padding`. Sẽ có 3 chế độ padding:

- **RSA_PKCS1_PADDING**: Thêm padding PKCS#1 v1.5 type 2 vào message trước khi mã hoá RSA (**OpenSSL mặc định là padding này nếu ta không lựa chọn gì**). Cấu trúc tổng quát của message sau khi được thêm padding như sau:
 1. Byte đầu tiên (0x00): Đánh dấu bắt đầu của bản mã.
 2. Byte thứ hai (0x02): Chỉ ra rằng đây là loại padding PKCS#1 v1.5 type 2.
 3. Padding ngẫu nhiên khác zero: Một chuỗi các byte ngẫu nhiên không phải là zero, dài `BN_num_bytes(rsa → n) - 3 - flen` byte. `BN_num_bytes` là một marco tính số byte cần thiết để lưu n .
 4. Byte phân cách (0x00): Một byte zero dùng làm dấu phân cách giữa padding và dữ liệu thực sự.
 5. Dữ liệu thực sự (`from`): Dữ liệu gốc cần mã hoá, dài `flen` byte.

- **RSA_PKCS1_OAEP_PADDING**: Thêm padding OAEP (Optimal Asymmetric Encryption Padding) vào message trước khi mã hoá RSA. Cấu trúc tổng quát của message sau khi được thêm padding như sau:
 1. Byte đầu tiên (0x00): Đánh dấu bắt đầu.
 2. Seed: Một chuỗi ngẫu nhiên gồm 20 byte.
 3. DB XOR Mask: trong đó DB (Data Block) bao gồm: 0x01 (byte đánh dấu kết thúc của padding) và dữ liệu gốc. Mask được tạo ra bằng cách sử dụng một hàm gọi là "Mask Generation Function" (MGF), MGF thường là một biến thể của hàm băm (hàm băm sử dụng trong trường hợp này là SHA-1), và nó sử dụng một giá trị ngẫu nhiên (gọi là seed) để tạo ra một chuỗi byte dài, hàm này trả về chuỗi dbMask. Sau đó ta tiến hành XOR từng byte tương ứng của DB với dbMask.
- **RSA_NO_PADDING**: không thêm bất kì padding gì mà chỉ chép nội dung từ dữ liệu nguồn (**from**) sang bộ đệm đích (**to**).

Sau đó sẽ tiến hành mã hoá trên message đã được padding bởi công thức: $c = m^e \bmod n$. Mã giả được trình bày ở Algorithm 1.

3.1.2 Quá trình giải mã

OpenSSL giải mã bằng hàm

```
1 static int rsa_ossl_private_decrypt(int flen, const unsigned char *from,
   unsigned char *to, RSA *rsa, int padding)
```

với các tham số tương tự như quá trình mã hoá được mô tả ở phần 3.1.1. Quy trình giải mã như sau:

- Đầu tiên sẽ đi kiểm tra các dữ liệu đầu vào và chuyển chúng về dạng số nguyên lớn.
- Sau đó OpenSSL sẽ xử lý blinding để chống lại Side-Channel Attacks nếu cờ **RSA_FLAG_NO_BLINDING** không được thiết lập, tức là blinding được yêu cầu. OpenSSL sẽ lấy cấu trúc Blinding phù hợp bằng hàm `blinding = rsa_get_blinding(rsa, &local_blinding, ctx)`, biến `local_blinding` sẽ được thiết lập để chỉ ra liệu blinding có phải là cục bộ (chỉ dùng cho luồng hiện tại) hay không. Nếu không lấy được cấu trúc blinding (`blinding == NULL`), hàm sẽ báo lỗi và thoát.
- Sau đó, OpenSSL tiến hành giải mã bằng 1 trong 2 phương pháp. Nếu cờ **RSA_FLAG_EXT_PKEY** không được bật, tức là sẽ không có các thành phần của khoá private ($p, q, dmp1, dmql, iqmp$), thì sẽ tiến hành giải mã $c^d = m \bmod n$. Ngược lại nếu cờ **RSA_FLAG_EXT_PKEY** được bật, OpenSSL sẽ giải mã tối ưu hóa dựa trên CRT - định lý đồng dư Trung Hoa. Phương pháp này nhanh hơn so với giải mã thông thường vì nó chia nhỏ phép toán thành hai phần nhỏ hơn, mỗi phần được thực hiện trên một số nguyên lớn nhỏ hơn (p và q). Phương pháp này tìm $m = c^d \bmod n$ như sau:

- $m_1 = c^{dmq1} \bmod q$
- $r_1 = c^{dmp1} \bmod p$
- $h = (r_1 - m_1) \times iqmp \bmod p$
- $m = m_1 + h \times q$

Algorithm 1: RSA Encryption

Data: RSA Public-key, message m , int padding
Result: return cipher c

```

if  $BN\_num\_bits(n) > OPENSSL\_RSA\_MAX\_MODULUS\_BITS$  then
     $ERR\_LIB\_RSA \leftarrow RSA\_R\_MODULUS\_TOO\_LARGE$ 
    return -1;
end
if  $BN\_compare(n, e) \leq 0$  then
     $ERR\_LIB\_RSA \leftarrow RSA\_R\_BAD\_E\_VALUE$ 
    return -1;
end
if  $BN\_num\_bits(n) > OPENSSL\_RSA\_SMALL\_MODULUS\_BITS$  then
    if  $e > OPENSSL\_RSA\_MAX\_PUBEXP\_BITS$  then
         $ERR\_LIB\_RSA \leftarrow RSA\_R\_BAD\_E\_VALUE$ 
        return -1;
    end
end
switch  $padding$  do
    case  $RSA\_PKCS1\_PADDING$  do
         $message = rsa\_padding\_add\_PKCS1\_type\_2(message)$ 
    end
    case  $RSA\_PKCS1\_OAEP\_PADDING$  do
         $message = rsa\_padding\_add\_PKCS1\_OAEP\_mgf1(message)$ 
    end
    case  $RSA\_NO\_PADDING$  do
         $message = RSA\_padding\_add\_none(message)$ 
    end
    otherwise do
         $ERR\_LIB\_RSA \leftarrow RSA\_R\_UNKNOWN\_PADDING\_TYPE$ 
    end
end
 $c \leftarrow rsa\_bn\_mod\_exp(message, e, n);$  /* Mod exp by Montgomery reduction */
return  $c$ ;

```

- Sau đó, thực hiện quá trình Blinding invert trên message thu được từ bước trên, nếu blinding được sử dụng.
- Cuối cùng loại bỏ padding để được message ban đầu (trong quá trình loại bỏ padding, nếu là $RSA_PKCS1_PADDING$ sẽ sinh ra khoá dẫn xuất (Key Derivation Key - KDK) để tăng cường an ninh trong xử lý padding và ngăn chặn việc tiết lộ thông tin không mong muốn thông qua lỗi padding).

Mã giả được trình bày ở Algorithm 2. Và các hàm nhân, mũ trong quá trình mã hoá và giải mã đều dùng thuật toán Montgomery reduction.

3.2 Chi tiết về mã nguồn

Từ mô tả về quá trình mã hoá và giải mã ở phần 3.1, mã nguồn của chương trình được em tổ chức như sau:

- Ngôn ngữ sử dụng: Python.
- Các thư viện cần cài đặt: cryptography (có thể dùng lệnh `$ pip install cryptography` hoặc xem cách cài đặt cụ thể tại [đây](#)).
- Cách biên dịch, cách chạy: dùng lệnh `$ py Source/Cau2/enc.py` để chạy chương trình mã hoá, `$ py Source/Cau2/enc.py` để chạy chương trình giải mã.
- Kết quả sau khi chạy: chương trình mã hoá sẽ trả về file `cipher.enc` và chương trình giải mã sẽ trả về file `plain_text.txt`, xem thêm ở video demo.

```

1 def encryption(plaintext_file, cipher_file, public_key_path):
2     # Read the public key content from the file
3     with open(public_key_path, 'rb') as key_file:
4         public_key = serialization.load_pem_public_key(
5             key_file.read(),
6             backend=default_backend()
7         )
8
9     # Read the data from the plaintext file
10    with open(plaintext_file, 'rb') as file:
11        plain_text = file.read()
12
13    # Encrypt using public key and PKCS#1 v1.5 padding
14    # because OpenSSL uses this padding by default
15    cipher = public_key.encrypt(
16        plain_text,
17        padding.PKCS1v15()
18    )
19
20    # Write the encrypted data to the cipher file
21    with open(cipher_file, 'wb') as file:
22        file.write(cipher)
23
24 def decryption(cipher_file, plain_file, private_key_path):
25     # Read the private key content from the file
26     with open(private_key_path, 'rb') as key_file:
27         private_key = serialization.load_pem_private_key(
28             key_file.read(),
29             password=None,
30             backend=default_backend()
31         )
32
33     # read data from cipher file
34     with open(cipher_file, 'rb') as file:
35         cipher = file.read()
36
37     # decrypt using private key and PKCS#1 v1.5 padding
38     # because OpenSSL uses this padding by default

```

```

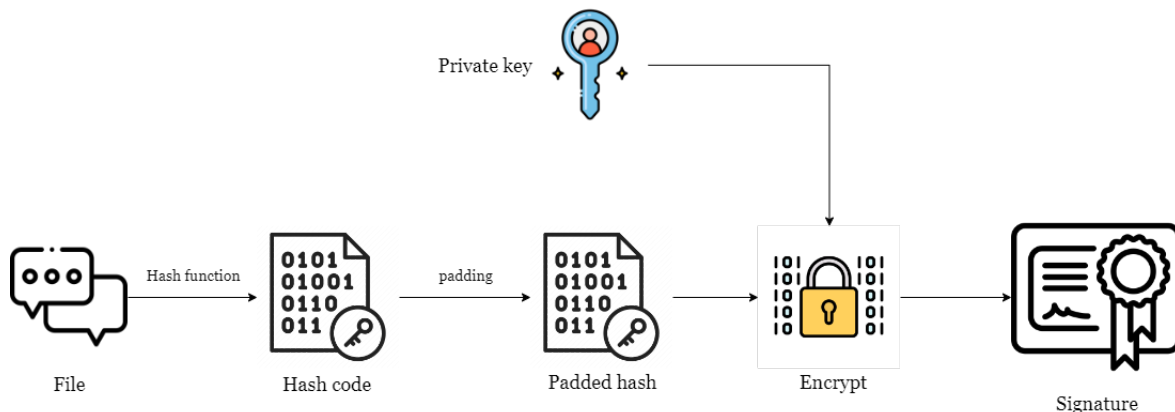
39 plain_text = private_key.decrypt(
40     cipher,
41     padding.PKCS1v15()
42 )
43
44 # write the decrypted data to the plain file
45 with open(plain_file, 'wb') as file:
46     file.write(plain_text)
    
```

3.3 Video demo

Video demo được lưu ở thư mục Demo với tên 'Cau2_enc.mp4' và Cau2_dec.mp4 hoặc xem tại [enc](#) và [dec](#).

4 Câu 3

4.1 Quá trình kí sử dụng private key



Hình 4: Quá trình kí một file của OpenSSL (tham khảo ở [đây](#)).

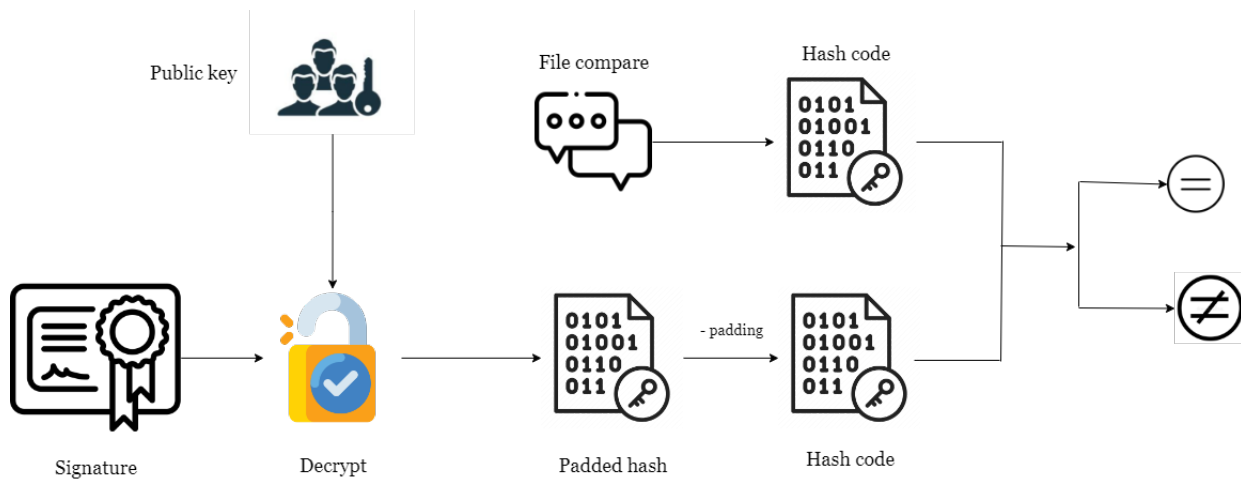
Quá trình kí của OpenSSL được mô tả bằng hình 4.

- Đầu tiên message sẽ được băm bởi một hàm băm. Hàm băm thường được dùng là SHA1, SHA256, SHA512, MD5 hoặc có thể kết hợp nhiều hàm băm lại với nhau để tăng cường bảo mật. Ở đây OpenSSL nếu không có tùy chọn thì sẽ **KHÔNG** băm message.
- Tiếp theo sẽ padding dữ liệu vừa được băm, nếu không có tùy chọn gì thì OpenSSL mặc định padding theo PKCS#1v1.5, cấu trúc của message sau khi padding theo kiểu này tương tự như cấu trúc RSA_PKCS1_PADDING trong quá trình mã hoá được trình bày ở phần 3.1.1, nhưng byte thứ hai thay vì là (0x02) thì sẽ là (0x01).
- Cuối cùng sẽ đem đi mã hoá bằng 2 cách như sau:
 - Nếu khoá ở dạng (n, d) thì trả về $s = m^d \bmod n$, với m là message đã được xử lý qua hai bước trên.

- Nếu khoá ở dạng $(p, q, dmp1, dmq1, iqmp)$ và có các giá trị bổ sung (r_i, d_i, t_i) thì sẽ dùng thuật toán sau:

- * $s_1 = m^{dmp1} \bmod p$ và $s_2 = m^{dmq1} \bmod q$.
- * Với $u > 2$ thì $s_i = m^{d_i} \bmod r_i, i = 3, \dots, u$.
- * $h = (s_1 - s_2) * iqmp \bmod p$.
- * Với $u > 2, R = r_1$ và for $i = 3$ đến u :
 - $R = R * r(i - 1)$
 - $h = (s_i - s) * t_i \bmod r_i$.
 - $s = s + R * h$
- * return s .

4.2 Quá trình xác nhận sử dụng public key



Hình 5: Quá trình xác thực sử dụng public key (tham khảo ở [đây](#)).

Quá trình xác thực của OpenSSL được mô tả qua hình 5.

- Đầu tiên, OpenSSL sẽ lấy n, e từ public key. Sau đó lấy message bởi công thức $m = s^e \bmod n$ với s từ file signature cần verify, quá trình này trả về message ở dạng padded hash.
- Sau đó ta loại bỏ padding đã sử dụng để thu được hash code.
- Cuối cùng ta sẽ so sánh hash code của file signature cần verify và hash code của file message gốc, nếu giống nhau thì trả về kết quả true, và ngược lại.

4.3 Chi tiết về mã nguồn

Từ mô tả quá trình kí và xác nhận đã nêu ở trên thì em tổ chức chương trình ký đọc tệp chứa khóa bí mật <priv.pem> và tệp chứa tin nhắn cần ký <mess>, xuất ra tệp chứa chữ ký <sign> và chương trình xác thực đọc tệp chứa khóa công khai <pub.pem>, tệp chứa tin nhắn <mess>, và tệp chứa chữ ký <sign> như sau:

- Ngôn ngữ sử dụng: Python.
- Các thư viện cần cài đặt: PyCryptodome (có thể dùng lệnh `$ pip install pycryptodome` hoặc xem cách cài đặt cụ thể tại [đây](#)).
- Cách biên dịch, cách chạy: dùng lệnh `$ py Source/Cau3/sign.py` để chạy chương trình ký, `$ py Source/Cau3/verify.py` để chạy chương trình xác thực.
- Kết quả sau khi chạy: chương trình ký sẽ trả về file **signature** và chương trình giải mã sẽ trả về kết quả giải mã thành công hay không (valid or invalid), cụ thể ở video demo.

```

1 # read data input
2 ...
3 def pkcs1_v1_5_encode(message, key_size):
4     # Determine the length of the key in bytes
5     k = key_size // 8
6
7     # Check if the message is not too long
8     if len(message) > k - 11:
9         raise ValueError("Message too long")
10
11     # The padding string PS, consisting of FF bytes
12     PS = b'\xFF' * (k - len(message) - 3)
13
14     # Construct the encoded message EM
15     EM = b'\x00\x01' + PS + b'\x00' + message
16     return EM
17
18 def sign_message_without_hash(private_key, message):
19     # Encode the message with PKCS#1 v1.5 padding
20     encoded_message = pkcs1_v1_5_encode(message, private_key.size_in_bits())
21
22     # Convert encoded message to an integer
23     m = bytes_to_long(encoded_message)
24
25     # Sign the message (compute m^d mod n)
26     s = pow(m, private_key.d, private_key.n)
27
28     # Convert the signature to a byte sequence
29     signature = long_to_bytes(s)
30     return signature
31
32 def pkcs1_v1_5_decode(encoded_message, key_size):
33     k = key_size // 8
34     if len(encoded_message) != k or not (encoded_message.startswith(b'\x00\x01')
35     and b'\x00' in encoded_message[2:]):
36         raise ValueError("Invalid encoded message")
37     index = encoded_message.find(b'\x00', 2)
38     return encoded_message[index + 1:]
39
40 def verify_message_without_hash(public_key, message, signature):
41     s = bytes_to_long(signature)
42     m = pow(s, public_key.e, public_key.n)
43     encoded_message = long_to_bytes(m, public_key.size_in_bytes())
44     try:

```

```
44         decoded_message = pkcs1_v1_5_decode(encoded_message, public_key.  
size_in_bits())  
45         return decoded_message == message  
46     except ValueError:  
47         return False
```

4.4 Video demo

Video demo được lưu ở thư mục Demo với tên ‘Cau3_sign.mp4’ và Cau3_verify.mp4 hoặc xem tại [sign](#) và [verify](#).

Algorithm 2: RSA Decryption

```

Data: RSA Private-key, cipher c, int padding
Result: return message m
// Check the input parameters
...
// Handle blinding
if !(rsa → flags & RSA_FLAG_NO_BLINDING) then
    | blinding ← rsa_get_blinding(rsa, local_blinding, ctx);
end
if blinding != NULL then
    | if !local_blinding AND ((unblind = BN_CTX_get(ctx)) == NULL) then
    |   ERR_LIB_RSA ← ERR_R_BN_LIB
    | end
    | if !rsa_blinding_convert(blinding, f, unblind, ctx) then
    |   goto err;
    | end
end
// do the decrypt
if rsa → flags & RSA_FLAG_EXT_PKEY then
    | rsa → meth → rsa_mod_exp(c, rsa);
    | /* decrypt by CRT theorem */
else
    | rsa → meth → bn_mod_exp(c, d, e);
    | /* decrypt directly */
end
// invert blinding if it's use
if blinding then
    | if !rsa_blinding_invert(blinding, unblind) then
    |   go err;
    | end
end
// remove padding
if padding == RSA_PKCS1_PADDING then
    | derive_kdk(flen, from, rsa, buf, num, kdk)
end
switch padding do
    case RSA_PKCS1_PADDING do
    | message = rsa_padding_check_PKCS1_type_2(message, kdk)
    end
    case RSA_PKCS1_OAEP_PADDING do
    | message = rsa_padding_check_PKCS1_OAEP_mgf1(message)
    end
    case RSA_NO_PADDING do
    | message = memcpy(to, buf)
    end
    otherwise do
    | ERR_LIB_RSA ← RSA_R_UNKNOWN_PADDING_TYPE
    end
end
return message;

```

5 Tài liệu tham khảo

- [1] [ChatGPT - Open AI](#)
- [2] [OpenSSL Cookbook 3rd Edition](#)
- [3] [Certificate Binary Posters \(Part One\)](#)
- [4] [OpenSSL repository](#)
- [5] [Documentation - RSA_private_decrypt\(\)](#)
- [6] [Documentation - RSA_public_encrypt\(\)](#)
- [7] [RSA sign and verify using Openssl : Behind the scene](#)
- [8] [PKCS 1: RSA Cryptography Specifications Version 2.2](#)