fit@hcmus

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

MÃ HÓA ỨNG DỤNG

LAB 02

RSA ENCRYPTION AND SIGNATURE

Giảng viên: Trần Minh Triết

Lớp: 19TN

Người thực hiện:

Họ và tên	MSSV
Đặng Thái Duy	19120491
Hà Chí Hào	19120219

MỤC LỤC

I. Cấu trúc chung

Trong thư mục Source gồm có các file quan trọng sau đây:

- Thư mục Task chứa mã nguồn theo từng thư mục Task tương ứng
- Từng task sẽ có 1 file source.cpp chứa mã nguồn tương ứng, file main.sh dùng để run source.cpp đó và chạy để in ra kết quả
- Makefile: chứa lệnh để chạy các file main.sh tương ứng theo từng task. Để chạy task chỉ cần gõ lệnh sau: make <task_name>

II. Task

Các task dưới đây đều sử dụng các hàm tiện ích utils và luồng xử lý chung cho từng bài như sau:

- Hàm utils
  - $printBN(string\ msg, BIGNUM * a)$: dùng để in ra output gồm đoạn thông báo msg và kết quả tương ứng là $BN\_bn2hex(a)$
- Luồng xử lý chung:
  - Khởi tạo: Các biến dùng trong chương trình phân làm 2 loại

    Biến môi trường ctx: BN_CTX *ctx = BN_CTX_new(): Ta cần biến này vì một số hàm thư viện yêu cầu các biến tạm thời. Khi mà việc cấp phát bộ nhớ động để tạo BIGNUM khá tốn kém khi được sử dụng kết hợp với các lệnh con của chương trình được gọi lặp đi lặp lại, do đó cấu trúc BN_CTX được tạo ra để chứa các biến tạm thời BIGNUM được sử dụng bởi các hàm trong thư viện. Nên ta cần phải tạo một cấu trúc như thế và chuyển cho các hàm chức năng yêu cầu biến đó

    Các biến BIGNUM sử dụng để tính toán: Ví dụ BIGNUM *a = BN_new()

  - Tiếp theo là hướng xử lý riêng cho từng bài sẽ được liệt kê tương ứng theo từng Task dưới đây
  - Xóa các vùng nhớ cấp phát: Dùng hàm $BN\_clear\_free()$ để xóa các biến khởi tạo trên

1. Task 1: Deriving the Private Key

Task này cho trước 3 số p, q và e là 3 số nguyên tố. Dùng (e, n) đại diện cho public key. Yêu cầu của task này là cần tính toán ra giá trị của private key d.

Phần xử lý chính:

- Lưu các giá trị input p, q và e

```
    // set value
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");
```

- Tính giá trị của n và $\phi(n)$ dựa trên công thức: $\begin{cases} n = pq \\ \phi(n) = (p-1)(q-1) \end{cases}$

```
    // n = p * q
    BN_mul(n, p, q, ctx);

    // phi(n) = (p - 1) * (q - 1)
    BN_sub(p_minus_1, p, BN_value_one());
    BN_sub(q_minus_1, q, BN_value_one());
    BN_mul(phi, p_minus_1, q_minus_1, ctx);
```

- Kiểm tra e và $\phi(n)$ có nguyên tố cùng nhau hay không
  - Nếu không, báo lỗi và dừng chương trình
  - Nếu có, tính và in ra kết quả private key d dựa trên công thức $d = e^{-1} \ mod \ \phi(n)$

```
    // check if e and phi(n) is relatively prime
    BN_gcd(res, phi, e, ctx);
    if (!BN_is_one(res))
    {
        cout << "Error: e and phi(n) is not relatively prime\n";
        exit(0);
    }

    // d = e^-1 mod phi(n)
    BN_mod_inverse(d, e, phi, ctx);
    printBN("Private Key d =", d);
```

- Khi xây dựng xong source code ta sẽ chạy chương trình bằng lệnh sau

```
# compile source.c then run
g++ -o run source.cpp -lcrypto
./run
```

→ Kết quả đạt được

```
[05/12/22]seed@VM:~/.../RSA Encryption and Signature$ make task1
cd Task/Task1; bash main.sh
Private Key d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

2. Task 2: Encryption a Message

Task này cho trước 2 số (e, n) là public key cùng với message cần mã hóa là M = "A top secret!". Trước hết ta chuyển đổi chuỗi ASCII sang chuỗi hex, sau đó chuyển tiếp sang BIGNUM bằng hàm BN_hex2bn(). Yêu cầu của task này là cần tính toán ra đoạn mã hóa của chuỗi message trên.

- Chuyển chuỗi ASCII sang chuỗi hex

```
# convert string to hex string
python -c 'print("Hex code for `A top secret!`: " + "A top secret!".encode("hex"))'
# -> result: "4120746f702073656372657421"
```

- Lưu các giá trị input n, e, M

```
    // set value
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_dec2bn(&e, "65537");

    // hex encode for " A top secret!"
    BN_hex2bn(&M, "4120746f702073656372657421");
```

- Mã hóa chuỗi hex trên dựa trên công thức $C = M^e \ mod \ n$ và in ra kết quả

```
    // encrypt M: M^e mod n
    BN_mod_exp(C, M, e, n, ctx);
    printBN("Encryption Output:", C);
```

- Khi xây dựng xong source code ta sẽ chạy chương trình bằng lệnh sau

```
# compile source.c then run
g++ -o run source.cpp -lcrypto
./run
```

→ Kết quả đạt được

```
[05/12/22]seed@VM:~/.../RSA Encryption and Signature$ make task2
cd Task/Task2; bash main.sh
Hex code for `A top secret!`: 4120746f702073656372657421
Encryption Output: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

3. Task 3: Decryption a Message

Task này cho trước public key và private key giống với Task 2. Yêu cầu của Task này là cần giải mã ciphertext C cho trước và chuyển đổi về chuỗi plaintext ASCII

- Lưu các giá trị input n, d và C:

```
    // set value
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
```

- Giải mã ciphertext C trên dựa trên công thức $M = C^d \ mod \ n$ và in ra kết quả

```
    // decrypt C: C^d mod n
    BN_mod_exp(M, C, d, n, ctx);
    printBN("Decryption Output:", M);
```

- Chạy chương trình để tìm ra kết quả giải mã của C và chuyển về dạng chuỗi ASCII

```
# compile source.c then run
g++ -o run source.cpp -lcrypto
./run

# decode the hex string into a plain ASCII string
python -c 'print("Decode Hex to ASCII string: "
+"50617373776F726420697320064656573".decode("hex"))'
# -> result: "Password is dees"
```

→ Kết quả đạt được:

```
[05/12/22]seed@VM:~/.../RSA Encryption and Signature$ make task3
cd Task/Task3; bash main.sh
Decryption Output: 50617373776F726420697320064656573
Decode Hex to ASCII string: Password is dees
```

4. Task 4: Signing a Message

Task này cho trước public key và private key giống với Task 2. Yêu cầu của task này là tạo ra chữ ký cho 2 đoạn thông điệp "I own you $2000" và "I own you $300". Sau đó so sánh 2 chữ ký này và mô tả kết quả nhận được.

- Trước hết, ta chuyển đổi 2 chuỗi ASCII sang 2 chuỗi hex tương ứng

```
# get the hex strings of 2 strings
python -c 'print("Hex code for `I owe you $2000`: " + "I owe you $2000".encode("hex"))'
python -c 'print("Hex code for `I owe you $3000`: " + "I owe you $2000".encode("hex"))'
```

→ Kết quả:

```
Hex code for `I owe you $2000`: 49206f776520796f75202432303030
Hex code for `I owe you $3000`: 49206f776520796f75202432303030
```

- Lưu các giá trị input và 2 chuỗi hex vừa mã hóa

```
    // set value
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // hex encode for "I owe you $2000"
    BN_hex2bn(&M1, "49206f776520796f75202432303030");

    // hex encode for "I owe you $3000"
    BN_hex2bn(&M2, "49206f776520796f75202433303030");
```

- Giải mã các đoạn thông điệp ra ciphertext dựa trên công thức $C = M^d \ mod \ n$ và in ra kết quả

```
    // encrypt M: M^d mod n
    BN_mod_exp(C1, M1, d, n, ctx);
    BN_mod_exp(C2, M2, d, n, ctx);
    printBN("M1 Signature Output:", C1);
    printBN("M2 Signature Output:", C2);
```

- Chạy chương trình bằng lệnh sau để in ra kết quả chữ ký của 2 đoạn message

```
# compile source.c then run
g++ -o run source.cpp -lcrypto
./run
```

→ Kết quả đạt được:

```
M1 Signature Output: 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
M2 Signature Output: 04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB
```

Nhận xét: Từ kết quả quan sát được, ta thấy tuy hai đoạn thông điệp ban đầu chỉ khác nhau đúng 1 ký tự nhưng khi ra được chữ ký của 2 bên thì kết quả khác nhau hoàn toàn. Từ đó cho thấy chữ ký rất xác thực và bảo mật khi mà dù chỉ sửa đổi thông điệp 1 phần cũng sẽ khiến tạo ra chữ ký khác hẳn hoàn toàn so với ban đầu

5. Task 5: Verifying a Signature

Theo đề bài chúng ta đã có:

- Nội dung: `M = Launch a missile.`
- Chữ ký của Alice: `S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F`
- Public key của Alice: `(e = 6553710, n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115)`

Để kiểm tra chữ ký có hợp lệ hay không, ta chỉ cần tính m_r (ảnh của R với R là ánh xạ 1-1 từ M vào M_S) bằng công thức $m_r = s^e mod\ n$, và lấy m_r so sánh với M, nếu chúng bằng nhau thì chữ ký hợp lệ.

Vì M là một văn bản thông thường nên ta phải chuyển về dạng hex trước khi đưa vào thư viện bignum của openssl. Ta có thể làm vậy với C++ bằng cách chuyển từng kí tự trong string thành số thập phân ASCII tương ứng, rồi in ra dưới dạng hex có độ dài 2 và có số 0 ở đầu nếu cần (leading zeroes) vào một stringstream. Làm vậy với mỗi kí tự, thì từ stringstream đó ta lấy string kết quả, đó chính là văn bản ban đầu đã được chuyển thành hex.

Ta lần lượt chuyển các dữ kiện ở trên thành cấu trúc dữ liệu BIGNUM* với các hàm BN_hex2bn, BN_dec2bn:

```
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_dec2bn(&e, "65537");
BN_hex2bn(&M, string_to_hex("Launch a missile.").c_str());
BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
```

Sau đó ta sẽ tính $m_r = s^e mod\ n$ bằng hàm BN_mod_exp:

```
BN_mod_exp(mr, S, e, n, ctx);
```

Có được m_r, ta chỉ việc so sánh m_r và M bằng hàm BN_cmp:

```
if (BN_cmp(mr, M) == 0)
{
    std::cout << "Chu ky hop le \n";
}
else
{
    std::cout << "Chu ky khong hop le \n";
}
```

Kết quả sẽ ra:

```
M = 4C61756E63682061206D697373696C652E

mr = 4C61756E63682061206D697373696C652E

Chu ky hop le
```

Sau khi thực hiện chương trình ta cần phải dọn bộ nhớ bằng hàm BN_clear_free với từng biến BIGNUM*.

Khi chữ ký bị corrupt byte cuối 2F thành 3F, thì kết quả sẽ ra:

```
M = 4C61756E63682061206D697373696C652E

mr =
91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294

Chu ky khong hop le
```

6. Task 6: Manually Verifying an X.509 Certificate

Với task này em sẽ thử với web của trường hcmus.edu.vn, trước tiên em sẽ down về các certificates của trang web bằng lệnh có sẵn trong file task của SEED lab:

```
openssl s_client -connect hcmus.edu.vn:443 -showcerts
```

Kết quả câu lệnh:

```
Certificate chain
 0 s:C = VN, L = Ho Chi Minh, O = University of Science - Vietnam National University, CN =
*.hcmus.edu.vn
   i:C = GB, ST = Greater Manchester, L = Salford, O = Sectigo Limited, CN = Sectigo RSA
Organization Validation Secure Server CA
-----BEGIN CERTIFICATE-----
MIIG/TCCBeWgAwIBAgIQLoVLFvYaO90lu60yKPpMzjANBgkqhkiG9w0BAQsFADCB
…
BWidjE3tfxY02I/zH/mtStk=
-----END CERTIFICATE-----
 1 s:C = GB, ST = Greater Manchester, L = Salford, O = Sectigo Limited, CN = Sectigo RSA
Organization Validation Secure Server CA
   i:C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN = USERTrust RSA
Certification Authority
-----BEGIN CERTIFICATE-----
MIIGGTCCBAGgAwIBAgIQE31TnKp8MamkM3AZaIR6jTANBgkqhkiG9w0BAQwFADCB
…
MntHWpdLgtJmwsQt6j8k9Kf5qLnjatkYYaA7jBU=
-----END CERTIFICATE-----
 2 s:C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN = USERTrust RSA
Certification Authority
   i:C = GB, ST = Greater Manchester, L = Salford, O = Comodo CA Limited, CN = AAA Certificate
Services
-----BEGIN CERTIFICATE-----
MIIFgTCCBGmgAwIBAgIQOXJEOvkit1HX02wQ3TE1lTANBgkqhkiG9w0BAQwFADB7
…
vGp4z7h/jnZymQyd/teRCBaho1+V
-----END CERTIFICATE-----
---
Server certificate
subject=C = VN, L = Ho Chi Minh, O = University of Science - Vietnam National University, CN =
*.hcmus.edu.vn

issuer=C = GB, ST = Greater Manchester, L = Salford, O = Sectigo Limited, CN = Sectigo RSA
Organization Validation Secure Server CA

---
```

Phần ở trên một certificate sẽ cho ta biết thông tin của subject (phần sau s:) và issuer (phần sau i:) của certificate đó. Với certificate đầu, subject là server hcmus.edu.vn, là người đang giữ certificate, issuer là Sectigo RSA Organization Validation Secure Server CA, là người cấp certificate đó. Và certificate thứ 2 là của CA đó, task này ta chỉ quan tâm tới 2 certificate đầu vì ta chỉ đang xác nhận xem chữ ký trên certificate của server có phải là của CA hay không. Để có thể kiểm chứng được, ta phải:

- Lấy public key (n, e) từ certificate của CA
- Lấy chữ ký S trong certificate của server
- Lấy phần thân (nội dung M) trong certificate của server
- Tính $m_r = s^e \ mod \ n$, và so sánh nó với M, nếu giống nhau thì chữ ký S trong certificate của server chính là của CA.

Trước khi kiểm chứng, em sẽ phải lấy 2 certificate đầu ra từ nội dung được download về, thay vì copy paste ra như file task của SEED Lab nói, em sẽ tự động hóa nó bằng cách viết một chương trình C++ nhỏ trong file extractCertificates.cpp để tự dò và lấy ra 2 certificate ấy.

Ý tưởng đơn giản chỉ là đọc từng dòng cho tới khi gặp string có chứa "BEGIN CERTIFICATE", sau đó thì sẽ bắt đầu vừa đọc từng dòng vừa thêm vào một biến lưu lại những dòng đã đọc cho tới khi gặp string có chứa "END CERTIFICATE" thì dừng. Lúc này ta đã có certificate đầu, ta chỉ cần làm vậy thêm 1 lần nữa thì sẽ có certificate thứ hai, và em sẽ lưu 2 certificate đó lần lượt vào c0.pem và c1.pem.

6.1 Lấy public key (n, e) từ certificate của CA

Trước tiên em sẽ extract giá trị n từ certificate, thì file task của SEED Lab cũng có hướng dẫn là dùng lệnh:

```
openssl x509 -in c1.pem -noout -modulus
```

Kết quả của câu lệnh:

```
Modulus=9C930246454A524892FC578DF92DEA53…AFFD7C011CDB
```

Với giá trị e, thì openssl không có lệnh nào cụ thể, nhưng ta có thể in ra mọi trường trong certificate, trong đó bao gồm cả e (Exponent):

```
openssl x509 -in c1.pem -text -noout
```

Kết quả của câu lệnh:

```
Certificate:
    Data:
        …
        Signature Algorithm: sha384WithRSAEncryption
        Issuer: C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN = USERTrust RSA
Certification Authority
        …
        Subject: C = GB, ST = Greater Manchester, L = Salford, O = Sectigo Limited, CN = Sectigo RSA
Organization Validation Secure Server CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:9c:93:02:46:45:4a:52:48:92:fc:57:8d:f9:2d:
                    …
                    1c:db
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            …

    Signature Algorithm: sha384WithRSAEncryption
         4e:13:40:96:c9:c3:e6:6e:5b:c0:e3:ba:f4:17:e1:ae:09:1f:
         …
         6a:d9:18:61:a0:3b:8c:15
```

Như vậy e = 65537

Vì mục đích tự động hoá nên em đã viết thêm file extractPublicKey.cpp để có thể lấy giá trị n và e ra từ 2 output trên. Với giá trị n thì em chỉ cần bỏ đi chuỗi "Modulus=" ở đầu, còn với e thì em phải đọc từng dòng cho tới khi gặp dòng có chuỗi "Exponent" thì dừng, và tìm số đầu tiên trong chuỗi đó.

6.2 Lấy chữ ký S trong certificate của server

Cũng như giá trị e của public key, openssl không có lệnh nào cụ thể để extract chữ ký từ certificate, mà phải in ra hết các trường trong certificate và tìm trong đó:

```
openssl x509 -in c0.pem -text -noout
```

Kết quả của câu lệnh:

```
Certificate:
    Data:
        …
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = GB, ST = Greater Manchester, L = Salford, O = Sectigo Limited, CN = Sectigo RSA
Organization Validation Secure Server CA
        …
        Subject: C = VN, L = Ho Chi Minh, O = University of Science - Vietnam National University, CN =
*.hcmus.edu.vn
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:be:d8:6f:28:bb:a9:b9:de:fb:77:77:c9:d1:2c:
                    …
                    ae:b1
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            …
    Signature Algorithm: sha256WithRSAEncryption
         62:b3:76:7d:2d:6e:76:41:01:37:33:95:61:0b:d0:43:6e:0c:
         …
         f9:ad:4a:d9
```

Về tự động hoá việc lấy ra certificate, vì có 2 chuỗi "Signature Algorithm" trong nội dung trên, nên file extractSignature.cpp của em sẽ phải đọc cho hết nội dung để tìm ra chuỗi "Signature Algorithm" cuối cùng, rồi để con trỏ file ở đó, rồi sau đó mới bắt đầu đọc signature, đọc cho tới khi string hiện tại không còn dấu 2 chấm trong string thì dừng. Lúc đó em sẽ có được signature.

Vì signature hiện tại còn có các dấu : và dấu xuống hàng, nên ta phải bỏ đi chúng, file task của SEED Lab cũng có hướng dẫn câu lệnh:

```
cat signature | tr -d '[:space:]:'
```

Là in chữ ký (chữ ký lúc này được lưu trong file signature) ra terminal, nhưng thay vì ra terminal thì sẽ được để thành input cho câu lệnh sau dấu |, là xoá đi mọi dấu : và khoảng trắng.

6.3 Lấy phần thân (nội dung M) trong certificate của server

Một Certificate Authority (CA) tạo ra chữ ký cho 1 server sẽ có bước đầu tiên là tính hash của certificate, và rồi ký lên cái hash đó. Cho nên khi kiểm chứng chữ ký, ta cũng phải hash certificate, nhưng vì hash được CA tạo ra trước khi dùng để tạo chữ ký, ta phải loại bỏ phần chữ ký ra ngoài trước khi hash.
Để làm được điều đó, ta phải parse certificate ra cấu trúc ASN.1 (Abstract Syntax Notation One), vì các certificate dạng X.509 được encode bằng ASN.1. Khi parse xong ta có thể dễ dàng chắc lọc ra các dữ liệu từ certificate, để parse được, ta có thể dùng lệnh sau:

```
openssl asn1parse -i -in c0.pem
```

Trong đó -i có nghĩa là thục dòng output theo độ sâu của cấu trúc, và -in chỉ đơn giản là tên file làm input.

Kết quả câu lệnh:

```
    0:d=0  hl=4 l=1789 cons: SEQUENCE
    4:d=1  hl=4 l=1509 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER           :2E854B16F61A3BDD25BBAD3228FA4CCE
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
    …
 1517:d=1  hl=2 l=  13 cons:  SEQUENCE
 1519:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1530:d=2  hl=2 l=   0 prim:   NULL
 1532:d=1  hl=4 l= 257 prim:  BIT STRING
```

Với các certificate dạng X.509 thì phần thân của certificate bắt đầu từ offset 4, nhưng phần kết thúc thì ta phải xem xét giá trị d ở từng dòng, có nghĩa là depth. Phần thân bắt đầu từ 1 dòng có d=1 thì tất cả những dòng có d > 1 bên dưới nó đều là bên trong nó, cho tới khi trước offset 1517 thì d=1 trở lại, đó chính là phần chữ ký. Để có thể extract đúng phần thân, ta có thể xử dụng -strparse offset, có tác dụng: “Parse the contents octets of the ASN.1 object starting at **offset**. This option can be used multiple times to "drill down" into a nested structure.” (https://www.openssl.org/docs/man1.1.1/man1/openssl-asn1parse.html)

```
openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
```

Sau khi lấy được phần thân và để vào trong file c0_body.bin, bây giờ ta có thể tiến hành hash nó bằng thuật toán SHA-256 được ghi chú trong certificate, bằng lệnh:

```
sha256sum c0_body.bin
```

Kết quả: 5b41c136c7d265b08cbd8c137ebf63684aa211f6b1c06ba3c7bf5e9801b5e190

6.4 Kiểm chứng certificate

Sau khi có hết public key (e, n) của CA, chữ ký S của CA và nội dung M, ta bây giờ có thể tiến hành kiểm chứng bằng phương pháp y hệt Task 5. Em viết hết phần kiểm chứng trong file verifyCert.cpp, với public key (e, n) và chữ ký S em chỉ việc lấy lại kết quả từ các file cpp ở trên, còn với phần thân của certificate được hash, thì ta phải thêm padding cho nó.

Encryption dạng SHA256withRSA sử dụng PKCS#1 v1.5 để pad cho hash. Dựa vào phương pháp trong https://datatracker.ietf.org/doc/html/rfc3447#section-9.2, đầu tiên, tuỳ theo thuật toán hash, mà mình để DER encoding của thuật toán hash đó vào đằng trước nội dung đã hash, đối với SHA-256, thì nó là:

`30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20`

https://datatracker.ietf.org/doc/html/rfc3447 trang 42

Gọi DER encoding đó là A, nội dung đã hash của mình là H, thì bây giờ ta đang có string AH. Sau đó ta sẽ để thêm 0x00 0x01 PS 0x00 vào đằng trước AH, với PS là 1 chuỗi các 0xFF có độ dài bằng độ dài cần để cho độ dài cả string 0x00 0x01 PS 0x00 AH được như độ dài mong muốn (256).
Cả quá trình trên đã được thực hiện trong hàm getPaddedHashedMessage(BIGNUM*& M) trong file verifyCert.cpp của em.

Sau khi có hết mọi thứ cần trong file verifyCert.cpp, em chỉ cần sử dụng code tính $m_r = s^e \, mod \, n$ y hệt như task 5. Và lấy m_r so sánh với M, nếu giống nhau thì chữ ký hợp lệ và ngược lại. Kết quả của chương trình là:

```
Padded hashed server certificate body:
01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D0609
608648016503040201050004205B41C136C7D265B08CBD8C137EBF63684AA211F6B1C06BA3C7BF5E9801B5E190

mr:
01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D0609
608648016503040201050004205B41C136C7D265B08CBD8C137EBF63684AA211F6B1C06BA3C7BF5E9801B5E190

Chu ky hop le
```