CMSC 150 Lab 7
RSA Public-key Encryption and Signature

Antonino, Erica Mae
Layug, Mikaella Louise
Santos, Ethan Mark

Task 1: Deriving the Private Key

1. Create file named ‘task1.c’

- Given: Let p, q, and e be three prime numbers. Let n = p*q. We will use (e, n) as the public key.

```
p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3
```

- ***task1.c***

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128

void printBN(char *msg, BIGNUM * a)
{
   /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
   char * number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *phi = BN_new();
    BIGNUM *res = BN_new();
    BIGNUM *p1 = BN_new();
    BIGNUM *q1 = BN_new();

    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");

    BN_mul(n, p, q, ctx);
    printBN("public key e: ", e);
    printBN("public key n: ", n);

    BN_sub(p1, p, BN_value_one());
    BN_sub(q1, q, BN_value_one());
    BN_mul(phi, p1, q1, ctx);

    BN_gcd(res, phi, e, ctx);
    if (!BN_is_one(res))
    {
        printf("Error: e and φ(n) is not relatively prime \n ");
        exit(0);
    }

    BN_mod_inverse(d, e, phi, ctx);
    printBN("private key d: ", d);

```

```
    BN_clear_free(p);
    BN_clear_free(q);
    BN_clear_free(n);
    BN_clear_free(res);
    BN_clear_free(phi);
    BN_clear_free(e);
    BN_clear_free(d);
    BN_clear_free(p1);
    BN_clear_free(q1);

    return 0;
}
```

- Initialize the variables:
    - p, q, n, e, d
    - phi, res, pl, ql

```
BIGNUM *p = BN_new();
BIGNUM *q = BN_new();
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
```

```
BIGNUM *phi = BN_new();
BIGNUM *res = BN_new();
BIGNUM *p1 = BN_new();
BIGNUM *q1 = BN_new();
```

- Assign the variables to their respective values:

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

- Let n = p*q. We will use (e, n) as the public key.

```
BN_mul(n, p, q, ctx);
printBN("public key e", e);
printBN("public key n", n);
```

- phi(n) = (p-1)*(q-1)
- Then check if e and phi(n) is relatively prime

```
BN_sub(p1, p, BN_value_one());
BN_sub(q1, q, BN_value_one());
BN_mul(phi, p1, q1, ctx);

BN_gcd(res, phi, e, ctx);
if (!BN_is_one(res))
{
    printf("Error: e and φ(n) is not relatively prime \n ");
    exit(0);
}

BN_mod_inverse(d, e, phi, ctx);
printBN("private key d", d);
```

2. Compile task 1 then run

```
[12/05/24]seed@VM:~/.../Labsetup$ gcc task1.c -lcrypto -o task
[12/05/24]seed@VM:~/.../Labsetup$ task
public key e:  0D88C3
public key n:  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0A
A1968DBB143D1
private key d:  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890F
E7C28A9B496AEB
```

- **d:** 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

Task 2: Encrypting a Message

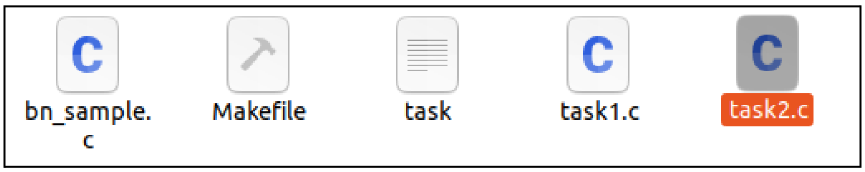
1. Convert the ASCII string “A top secret!” to hex string

```
[12/04/24]seed@VM:~/.../Labsetup$ python3 -c 'print("A top secret!
".encode("utf-8").hex())'
4120746f702073656372657421
```

- Given: The public keys are listed in the following (hexadecimals). Private key d is also given to help verify the encryption result.

```
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e = 010001 (this hex value equals to decimal 65537)
M = A top secret!
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
```

2. Create file named ‘task2.c’



- ***task2.c***

```
*task2.c
~/Downloads/Labsetup-2/Labsetup
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

void printBN(char *msg, BIGNUM * a)
{
   /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
   char * number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main ()
{
  BN_CTX *ctx = BN_CTX_new();

  BIGNUM *n = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *M = BN_new();
  BIGNUM *d = BN_new();

  BIGNUM *encrypt = BN_new();
  BIGNUM *decrypt = BN_new();

  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&e, "010001");
  BN_hex2bn(&M, "4120746f702073656372657421");
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

  BN_mod_exp(encrypt,M,e,n,ctx);
  BN_mod_exp(decrypt,encrypt,d,n,ctx);

  printBN("Encrypted Message: ", encrypt);
  printBN("Decrypted Message: ", decrypt);

  return 0;
}
```

- Initialize the variables:
    - n, e, M, d
    - decrypt
    - decrypt

```
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *M = BN_new();
BIGNUM *d = BN_new();
```

```
BIGNUM *encrypt = BN_new();
BIGNUM *decrypt = BN_new();
```

- Assign the variables to their respective values:

```
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&M, "4120746f702073656372657421");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

- Cncrypt using the formula x^e mod N and decrypt using y^d mod N

```
BN_mod_exp(encrypt,M,e,n,ctx);
BN_mod_exp(decrypt,encrypt,d,n,ctx);

printBN("Encrypted Message: ", encrypt);
printBN("Decrypted Message: ", decrypt);
```

3. Compile task 2 then run

```
[12/04/24]seed@VM:~/.../Labsetup$ gcc task2.c -lcrypto -o task
[12/04/24]seed@VM:~/.../Labsetup$ task
Encrypted Message:  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A7
5CACDC5DE5CFC5FADC
Decrypted Message:  4120746F702073656372657421
```

- **Encrypted Message:** 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
- **Decrypted Message:** 4120746F702073656372657421

4. Decrypted message is the same as the hexadecimal form of the ASCII message *'A top secret!'* from step 1:

```
[12/04/24]seed@VM:~/.../Labsetup$ python3 -c "print(bytes.fromhex(
'4120746f702073656372657421').decode('utf-8'))"
A top secret!
```

Task 3: Decrypting a Message

1. Create file named ‘task3.c’

- Given: public/private keys and ciphertext C

```
C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F
```

```
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
```

```
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
```

- ***task3.c***

```
task3.c
~/Downloads/Task1&2-Lab9-Labsetup/Lab9-Labsetup

#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

void printBN(char *msg, BIGNUM * a)
{
   /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
   char * number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main ()
{
  BN_CTX *ctx = BN_CTX_new();

  BIGNUM *n = BN_new();
  BIGNUM *d = BN_new();
  BIGNUM *C = BN_new();
  BIGNUM *decrypt = BN_new();

  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
  BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

  BN_mod_exp(decrypt, C, d, n, ctx);
  printBN("Decrypted message: ", decrypt);

  return 0;
}
```

- Initialize the variables:
  - n, d, C
  - ctx
  - Decrypt

```
BN_CTX *ctx = BN_CTX_new();

BIGNUM *n = BN_new();
BIGNUM *d = BN_new();
BIGNUM *C = BN_new();
BIGNUM *decrypt = BN_new();
```

- Assign the variables to their respective values:

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

- Decrypt using ciphertext^d mod n

```
BN_mod_exp(decrypt, C, d, n, ctx);
printBN("Decrypted message: ", decrypt);
```

3. Compile task 3 then run

```
[12/04/24]seed@VM:~/.../Labsetup$ gcc task3.c -lcrypto -o task
[12/04/24]seed@VM:~/.../Labsetup$ task
Decrypted message:  50617373776F726420697320646565 73
```

- **Decrypted message:** 50617373776F72642069732064656573

4. Use the following python command to convert the decrypted hex string back to to a plain ASCII string

```
[12/04/24]seed@VM:~/.../Labsetup$ python3 -c "print(bytes.fromhex(
'50617373776F72642069732064656573').decode('utf-8'))"
Password is dees
```

Task 4: Signing a Message

1. Create file named ‘task4.c’

- Given: public/private keys

```
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
```

```
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
```

2. Convert the 2 messages to hex string

- I owe you $2000.
- I owe you $3000.
  - *Note that the period ‘.’ is included in the conversion*

```
[12/04/24]seed@VM:~/.../Labsetup$ python3 -c 'print("I owe you $20
00.".encode("utf-8").hex())'
49206f776520796f752024323030302e
[12/04/24]seed@VM:~/.../Labsetup$ python3 -c 'print("I owe you $30
00.".encode("utf-8").hex())'
49206f776520796f752024333030302e
```

- ***task4.c***

```c
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

void printBN(char *msg, BIGNUM * a)
{
   /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
   char * number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main ()
{
  BN_CTX *ctx = BN_CTX_new();

  BIGNUM *n = BN_new();
  BIGNUM *d = BN_new();
  BIGNUM *M1 = BN_new();
  BIGNUM *M2 = BN_new();
  BIGNUM *s1 = BN_new();
  BIGNUM *s2 = BN_new();

  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
  BN_hex2bn(&M1, "49206f776520796f752024323030302e");
  BN_hex2bn(&M2, "49206f776520796f752024333030302e");

  BN_mod_exp(s1, M1, d, n, ctx);
  BN_mod_exp(s2, M2, d, n, ctx);
  printBN("Signature of 'I owe you $2000.': ", s1);
  printBN("Signature of 'I owe you $3000.': ", s2);

  return 0;
}
```

- Initialize the variables:
  - ctx, n, d
  - M1 & M2
  - s1 & s2

```c
BN_CTX *ctx = BN_CTX_new();

BIGNUM *n = BN_new();
BIGNUM *d = BN_new();
BIGNUM *M1 = BN_new();
BIGNUM *M2 = BN_new();
BIGNUM *s1 = BN_new();
BIGNUM *s2 = BN_new();
```

- Assign the variables to their respective values:

```c
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&M1, "49206f776520796f752024323030302e");
BN_hex2bn(&M2, "49206f776520796f752024333030302e");
```

- Sign the two messages

```
BN_mod_exp(s1, M1, d, n, ctx);
BN_mod_exp(s2, M2, d, n, ctx);
printBN("Signature of 'I owe you $2000.': ", s1);
printBN("Signature of 'I owe you $3000.': ", s2);
```

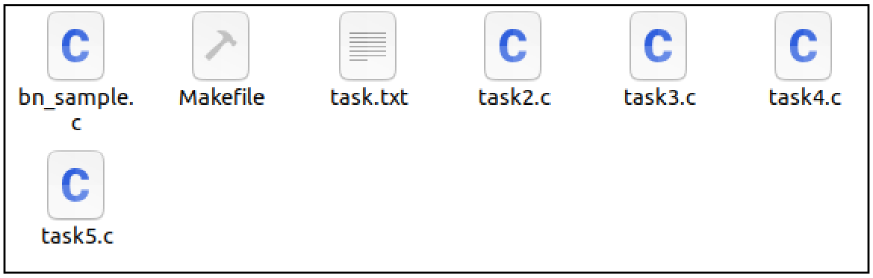
- Compile task 4 then run

```
[12/04/24]seed@VM:~/.../Labsetup$ gcc task4.c -lcrypto -o task
[12/04/24]seed@VM:~/.../Labsetup$ task
Signature of 'I owe you $2000.':  55A4E7F17F04CCFE2766E1EB32ADDBA8
90BBE92A6FBE2D785ED6E73CCB35E4CB
Signature of 'I owe you $3000.':  BCC20FB7568E5D48E434C387C06A6025
E90D29D848AF9C3EBAC0135D99305822
```

- Signature of 'I owe you $2000.': 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
- Signature of 'I owe you $3000.': BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822

Task 5: Verifying a Signature

1. Create file named ‘task5.c’



2. Edit the task5.c code to accommodate the given values.

```
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

void printBN(char *msg, BIGNUM * a)
{
   /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
   char * number_str = BN_bn2hex(a);
   printf("%s %s\n", msg, number_str);
   OPENSSL_free(number_str);
}

int main ()
{
  BN_CTX *ctx = BN_CTX_new();

  BIGNUM *M = BN_new();
  BIGNUM *S = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *n = BN_new();
  BIGNUM *res = BN_new();z

  BN_hex2bn(&M, "");
  BN_hex2bn(&S, "");
  BN_hex2bn(&e, "");
  BN_hex2bn(&n, "");

  BN_mod_exp(res, S, e, n, ctx);
  printBN("Message: ", res);
  printBN("Result: ", res);

  return 0;
}
```

3. To make the given message compatible, we convert it to its hex value.

```
[12/05/24]seed@VM:~/.../lab7$ python3 -c 'print("Launch a missile.".encode("utf-8").hex())'
4c61756e63682061206d697373696c652e
```

4. We put the message and signature values into the code. We alsocreate an if else print statement.

```c
  BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
  BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005C-
AB026C0542CBDB6802F643D6F34902D9C7EC90CB0B2BCA36C47FA37165C000-
5CAB026C0542CBDB6802F");
  BN_hex2bn(&e, "010001");
  BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF12335951-
13AA51B450F18116115");

  BN_mod_exp(res, S, e, n, ctx);
  printBN("Message: ", M);
  printBN("Result: ", res);

  if(BN_cmp(res, M) == 0){
    print("Verification success.");
  }else{
    print("Verification failed.");
  }

  return 0;
}
```

6. When we compile and execute the file, we see that the verification is a success.

```
seed@VM: ~/.../lab7
[12/05/24]seed@VM:~/.../lab7$ gcc -o task5 task5.c -lcrypto
[12/05/24]seed@VM:~/.../lab7$ task5
Message:  4C61756E63682061206D697373696C652E
Result:  4C61756E63682061206D697373696C652E
Verification success.[12/05/24]seed@VM:~/.../lab7$
```

7. We replace the last 2F value of the signature with 3F which breaks the compatibility and shows us a failed verification.

```c
  BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
  BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
  BN_hex2bn(&e, "010001");
  BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
```

```
Verification success.[12/05/24]seed@VM:~/.../lab7$ gcc -o task5 task5.c -lcrypto
[12/05/24]seed@VM:~/.../lab7$ task5
Message:  4C61756E63682061206D697373696C652E
Result:  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
Verification failed.[12/05/24]seed@VM:~/.../lab7$
```

Task 6: Manually Verifying an X.509 Certificate

1. First step, we download a certificate from a real web server with the url ‘dpsm.cas.upm.edu.ph’.

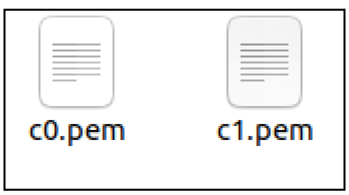
```
[12/05/24]seed@VM:~/.../lab7$ openssl s_client -connect dpsm.cas.upm.edu.ph:443 -showcerts
CONNECTED(00000003)
depth=1 C = US, O = Let's Encrypt, CN = R11
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = dpsm.cas.upm.edu.ph
verify return:1
---
```

We obtain two certificates, shown below.

```
Certificate chain
 0 s:CN = dpsm.cas.upm.edu.ph
   i:C = US, O = Let's Encrypt, CN = R11
-----BEGIN CERTIFICATE-----
MIIE+DCCA+CgAwIBAgISA51310MSMhTl7rehyaN+EXIbMA0GCSqGSIb3DQEBCwUA
MDMxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQwwCgYDVQQD
EwNSMTEwHhcNMjQxMTIyMTgxMjIyWhcNMjUwMjIwMTgxMjIxWjAeMRwwGgYDVQQD
ExNkcHNtLmNhcy51cG0uZWR1LnBoMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAvgcbbyi1roxLLsAVv/FRaqx6wQlICeiUCSeZgNQzuxi0VR+r/Fj+guQg
pdqS7IgbW/ahLflh4tDGsclEVkM7nEhhP3kZ+b85mhx3453oBFwM6QqzFgfN7aXE
GBMvjn6tOlHA7IGlGtL/0NH8f/iDWhWvUKHlotVQFZiLY8ddbNZNIw7e4f5rcMCn
4H+ToKGUx015JiI8UpSDd7VfVj8G+9OiO0bQZfx4nMMN+e2JJ+EXPj075h3sn8KC
0Ro3R69Ketieo/K+pshBtA7t016EH6X+/4iqDaK6xghbpcEtQ08sCA2Q8eUcGPtx
d9ycxt0kWbOcU5nGLlhJnikOzRj/3QIDAQABo4ICGTCCAhUwDgYDVR0PAQH/BAQD
AgWgMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAMBgNVHRMBAf8EAjAA
MB0GA1UdDgQWBBS8sfH9mLwjzRFy/7LVZWWhMtV+lzAfBgNVHSMEGDAWgBTFz0ak
6vTDwHpslcQtsF6SLybjuTBXBggrBgEFBQcBAQRLMEkwIgYIKwYBBQUHMAGGFmh0
dHA6Ly9yMTEuby5sZW5jci5vcmcwIwYIKwYBBQUHMAKGF2h0dHA6Ly9yMTEuaS5s
ZW5jci5vcmcvMB4GA1UdEQQXMBWCE2Rwc20uY2FzLnVwbS5lZHUucGgwEwYDVR0g
BAwwCjAIBgZngQwBAgEwggEGBgorBgEEAdZ5AgQCBIH3BIH0APIAdwCi4wrkRe+9
rZt+OO1HZ3dT14JbhJTXK14bLMS5UKRH5wAAAZNVSMVqAAAEAwBIMEYCIQCGv8tz
qlTqllf/EUt1KDANkrnHOPVuDxznPoUNobyuAQIhAN9i6K15QkwS3LjMC51WvtM2
YbFKmSUU47AxIJKskdpHAHcAzxFW7tUufK/zh1vZaS6b6RpxZ0qwF+ysAdJbd87M
OwgAAAGTVUjFkwAABAMASDBGAiEA9R6E/68HR7MzKCJMC5BSfTS0foM8K50V6+hU
EwVCzpkCIQDkT34ym7bf0Y2x+FISjqqLIOu/VkagDWeXWoOqSRfyyjANBgkqhkiG
9w0BAQsFAAOCAQEAp93u8MaZI9UXXnKvJuWKDlxAgAGJUXt+N1k5MUt40Ae9RnP4
2pIAKy0g9PUDnT5/h1N0n+AWhyd/ffxFipHd62vG3KlSnWn1T5r4bVgL0JFjjKL8
RThZugaZVYkhfX0ki/Ac5OvGxO1xG4ss5zzniDECemJgTAfRlxKupokvHBjB5hho
K0rVN1aD5TA70VKwDuxzOHy8A8k4dn0cpmWPMs2haP3rlRKmBCJ5GgNcvJa+b0gq
jtmYhksuC4Rojou/SI2bLEn0HLGst2ODtp3lZWS2TBCVc/TzvLeyeYrPa3DQTjQd
zElXNcBksUGQH18dJKyYsQ4l5zOHdcBHhruWug==
-----END CERTIFICATE-----
```

```
 1 s:C = US, O = Let's Encrypt, CN = R11
   i:C = US, O = Internet Security Research Group, CN = ISRG Root X1
-----BEGIN CERTIFICATE-----
MIIFBjCCAu6gAwIBAgIRAIp9PhPWLzDvI4a9KQdrNPgwDQYJKoZIhvcNAQELBQAw
TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh
cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMjQwMzEzMDAwMDAw
WhcNMjcwMzEyMjM1OTU5WjAzMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNTGV0J3Mg
RW5jcnlwdDEMMAoGA1UEAxMDUjExMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAuoe8XBsAOcvKCs3UZxD5ATylTqVhyybKUvsVAbe5KPUoHu0nsyQYOWcJ
DAjs4DqwO3cOvfPlOVRBDE6uQdaZdN5R2+97/1i9qLcT9t4x1fJyyXJqC4N0lZxG
AGQUmfOx2SLZzaiSqhwmej/+71gFewiVgdtxD4774zEJuwm+UE1fj5F2PVqdnoPy
6cRms+EGZkNIGIBloDcYmpuEMpexsr3E+BUAnSeI++JjF5ZsmydnS8TbKF5pwnnw
SVzgJFDhxLyhBax7QG0AtMJBP6dYuC/FXJuluwme8f7rsIU5/agK70XEeOtlKsLP
Xzze41xNG/cLJyuqC0J3U095ah2H2QIDAQABo4H4MIH1MA4GA1UdDwEB/wQEAwIB
hjAdBgNVHSUEFjAUBggrBgEFBQcDAgYIKwYBBQUHAwEwEgYDVR0TAQH/BAgwBgEB
/wIBADAdBgNVHQ4EFgQUxc9GpOr0w8B6bJXELbBeki8m47kwHwYDVR0jBBgwFoAU
ebRZ5nu25eQBc4AIiMgaWPbpm24wMgYIKwYBBQUHAQEEJjAkMCIGCCsGAQUFBzAC
hhZodHRwOi8veDEuaS5sZW5jci5vcmcvMBMGA1UdIAQMMAowCAYGZ4EMAQIBMCcG
A1UdHwQgMB4wHKAaoBiGFmh0dHA6Ly94MS5jLmxlbmNyLm9yZy8wDQYJKoZIhvcN
AQELBQADggIBAE7iiV0KAxyQOND1H/lxXPjDj7I3iHpvsCUf7b632IYGjukJhM1y
v4Hz/MrPU0jtvfZpQtSlET41yBOykh0FX+ou1Nj4ScOt9ZmWnO8m2OG0JAtIIE38
01S0qcYhyOE2G/93ZCkXufBL713qzXnQv5C/viOykNpKqUgxdKlEC+Hi9i2DcaR1
e9KUwQUZRhy5j/PEdEglKg3l9dtD4tuTm7kZtB8v32oOjzHTYw+7KdzdZiw/sBtn
UfhBPORNuay4pJxmY/WrhSMdzFO2q3Gu3MUBcdo27goYKjL9CTF8j/Zz55yctUoV
aneCWs/ajUX+HypkBTA+c8LGDLnWO2NKq0YD/pnARkAnYGPfUDoHR9gVSp/qRx+Z
WghiDLZsMwhN1zjtSC0uBWiugF3vTNzYIEFfaPG7Ws3jDrAMMYebQ95JQ+HIBD/R
PBuHRTBpqKlyDnkSHDHYPiNX3adPoPAcgdF3H2/W0rmoswMWgTlLn1Wu0mrks7/q
pdWfS6PJ1jty80r2VKsM/Dj3YIDfbjXKdaFU5C+8bhfJGqU3taKauuz0wHVGT3eo
6FlWkWYtbt4pgdamlwVeZEW+LM7qZEJEsMNPrfC03APKmZsJgpWCDWOKZvkZcvjV
uYkQ4omYCTX5ohy+knMjdOmdH9c7SpqEWBDC86fiNex+O0XOMEZSa8DA
-----END CERTIFICATE-----
```

We then save these certificates into two files, c0.pem and c1.pem.



2. Extract the public key (e, n) from the issuer's certificate. We do this by using the modulus command from the module. "openssl x509 -in c0.pem -noout -modulus" and "openssl x509 -in c1.pem -noout -modulus" to extract the modulus n.

```
[12/05/24]seed@VM:~/.../lab7$ openssl x509 -in c1.pem -noout -modulus
Modulus=BA87BC5C1B0039CBCA0ACDD46710F9013CA54EA561CB26CA52FB1501B7B928F5281EED27B324183967090C08ECE03A
B03B770EBDF3E53954410C4EAE41D69974DE51DBEF7BFF58BDA8B713F6DE31D5F272C9726A0B8374959C4600641499F3B1D922
D9CDA892AA1C267A3FFEEF58057B089581DB710F8EFBE33109BB09BE504D5F8F91763D5A9D9E83F2E9C466B3E1066643481880
65A037189A9B843297B1B2BDC4F815009D2788FBE26317966C9B27674BC4DB285E69C279F0495CE02450E1C4BCA105AC7B406D
00B4C2413FA758B82FC55C9BA5BB099EF1FEEBB08539FDA80AEF45C478EB652AC2CF5F3CDEE35C4D1BF70B272BAA0B4277534F
796A1D87D9
```

Then we use the command below to extract the exponent, which is the same as in task 5.

```
[12/05/24]seed@VM:~/.../lab7$ openssl x509 -in c1.pem -text -noout
```

```
    4d:1b:17:0b:27:2b:aa:0
    87:d9
 Exponent: 65537 (0x10001)
extensions:
09v3 Key Usage: critical
```

3. Extract the signature from the server's certificate. We use the ‘openssl x509 -in c0.pem -text -noout’ command for this.

```
[12/05/24]seed@VM:~/.../lab7$ openssl x509 -in c0.pem -text -noout
Certificate:
```

```
Signature Algorithm: sha256WithRSAEncryption
     a7:dd:ee:f0:c6:99:23:d5:17:5e:72:af:26:e5:8a:0e:5c:40:
     80:01:89:51:7b:7e:37:59:39:31:4b:78:d0:07:bd:46:73:f8:
     da:92:00:2b:2d:20:f4:f5:03:9d:3e:7f:87:53:74:9f:e0:16:
     87:27:7f:7d:fc:45:8a:91:dd:eb:6b:c6:dc:a9:52:9d:69:f5:
     4f:9a:f8:6d:58:0b:d0:91:63:8c:a2:fc:45:38:59:ba:06:99:
     55:89:21:7d:7d:24:8b:f0:1c:e4:eb:c6:c4:ed:71:1b:8b:2c:
     e7:3c:e7:88:31:02:7a:62:60:4c:07:d1:97:12:ae:a6:89:2f:
     1c:18:c1:e6:18:68:2b:4a:d5:37:56:83:e5:30:3b:d1:52:b0:
     0e:ec:73:38:7c:bc:03:c9:38:76:7d:1c:a6:65:8f:32:cd:a1:
     68:fd:eb:95:12:a6:04:22:79:1a:03:5c:bc:96:be:6f:48:2a:
     8e:d9:98:86:4b:2e:0b:84:68:8e:8b:bf:48:8d:9b:2c:49:f4:
     1c:b1:ac:b7:63:83:b6:9d:e5:65:64:b6:4c:10:95:73:f4:f3:
     bc:b7:b2:79:8a:cf:6b:70:d0:4e:34:1d:cc:49:57:35:c0:64:
     b1:41:90:1f:5f:1d:24:ac:98:b1:0e:25:e7:33:87:75:c0:47:
     86:bb:96:ba
```

Let us save this block of data in a ‘signature’ file.

To then remove the spaces and colons in order to be able to use the value, we use the ‘cat’ program below.

```
[12/05/24]seed@VM:~/.../lab7$ cat signature | tr -d '[:space:]:'
a7ddeef0c69923d5175e72af26e58a0e5c40800189517b7e375939314b78d007
bd4673f8da92002b2d20f4f5039d3e7f8753749fe01687277f7dfc458a91ddeb
6bc6dca9529d69f54f9af86d580bd091638ca2fc453859ba06995589217d7d24
8bf01ce4ebc6c4ed711b8b2ce73ce78831027a62604c07d19712aea6892f1c18
c1e618682b4ad5375683e5303bd152b00eec73387cbc03c938767d1ca6658f32
```

4. Extract the body of the server's certificate.

```
[12/05/24]seed@VM:~/.../lab7$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1272 cons: SEQUENCE
    4:d=1  hl=4 l= 992 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=  18 prim:   INTEGER           :039D77D743123214E5EEB7A1C9A37E11721B
   33:d=2  hl=2 l=  13 cons:   SEQUENCE
   35:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   46:d=3  hl=2 l=   0 prim:    NULL
   48:d=2  hl=2 l=  51 cons:   SEQUENCE
   50:d=3  hl=2 l=  11 cons:    SET
   52:d=4  hl=2 l=   9 cons:     SEQUENCE
   54:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   59:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   63:d=3  hl=2 l=  22 cons:    SET
   65:d=4  hl=2 l=  20 cons:     SEQUENCE
   67:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   72:d=5  hl=2 l=  13 prim:      PRINTABLESTRING   :Let's Encrypt
```

```
[12/05/24]seed@VM:~/.../lab7$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[12/05/24]seed@VM:~/.../lab7$ sha256sum c0_body.bin
9808ee2bf2576b0a74190cf1324cdb6d5f50f290f25166eab3ca4154b8197e7b  c0_body.bin
```

Because we need 256 bytes to compare the values, we need to pad the hash we obtained from the previous step. We use the python code below to pad it.

```
prefix = "0001"
hash = "9808EE2BF2576B0A74190CF1324CDB6D5F50F290F25166EAB3CA4154B8197E7B"
A = "30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20".replace(' ','')
total_len = 256
pad_len = total_len - 1 - (len(A)+len(prefix)+len(hash))//2
prefix + "FF" * pad_len + "00" + A + hash
```

```
[12/05/24]seed@VM:~/.../lab7$ python3 padder.py
0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFF003031300D0609608648016503040201050004209808EE2BF2576B0A74190CF1324CDB6D5F50F2
90F25166EAB3CA4154B8197E7B
```

5. We verify the signature using the code in the previous task, ‘task5.c’. The code is edited to have the values obtained from the steps performed in this task.

```c
BIGNUM *M = BN_new();
BIGNUM *S = BN_new();
BIGNUM *e = BN_new();
BIGNUM *n = BN_new();
BIGNUM *res = BN_new();

BN_hex2bn(&M, "0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF-
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF-
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF-
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D060960864-
8016503040201050004209808EE2BF2576B0A74190CF1324CDB6D5F50F290F25166EAB3CA4154B8197E7B");
BN_hex2bn(&S, "a7ddeef0c69923d5175e72af26e58a0e5c40800189517b7e375939314b78d007bd4673f8da92002b2d20f4f5039d3e-
7f8753749fe01687277f7dfc458a91ddeb6bc6dca9529d69f54f9af86d580bd091638ca2fc453859ba06995589217d7d248bf01ce4ebc6c-
4ed711b8b2ce73ce78831027a62604c07d19712aea6892f1c18c1e618682b4ad5375683e5303bd152b00eec73387cbc03c938767d1ca665-
8f32");
BN_hex2bn(&e, "010001");
BN_hex2bn(&n, "BA87BC5C1B0039CBCA0ACDD46710F9013CA54EA561CB26CA52FB1501B7B928F5281EED27B324183967090C08ECE03A-
B03B770EBDF3E53954410C4EAE41D69974DE51DBEF7BFF58BDA8B713F6DE31D5F272C9726A0B8374959C4600641499F3B1D922D9CDA892A-
A1C267A3FFEEF58057B089581DB710F8EFBE33109BB09BE504D5F8F91763D5A9D9E83F2E9C466B3E106664348188065A037189A9B843297-
B1B2BDC4F815009D2788FBE26317966C9B27674BC4DB285E69C279F0495CE02450E1C4BCA105AC7B406D00B4C2413FA758B82FC55C9BA5B-
B099EF1FEEBB08539FDA80AEF45C478EB652AC2CF5F3CDEE35C4D1BF70B272BAA0B4277534F796A1D87D9");

BN_mod_exp(res, S, e, n, ctx);
printBN("Message: ", M);
printBN("Result: ", res);

if(BN_cmp(res, M) == 0)
{
  printf("Verification success.");
}else
{
  printf("Verification failed.");
}
```

6. When we compile the code and run it, we see that the signature is verified.

```
[12/05/24]seed@VM:~/.../lab7$ task5
Message:  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF003031300D0609608648016503040201050004209808EE2BF2576B0A74190CF1
324CDB6D5F50F290F25166EAB3CA4154B8197E7B
Result:  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFF003031300D0609608648016503040201050004209808EE2BF2576B0A74190CF13
24CDB6D5F50F290F25166EAB3CA4154B8197E7B
```