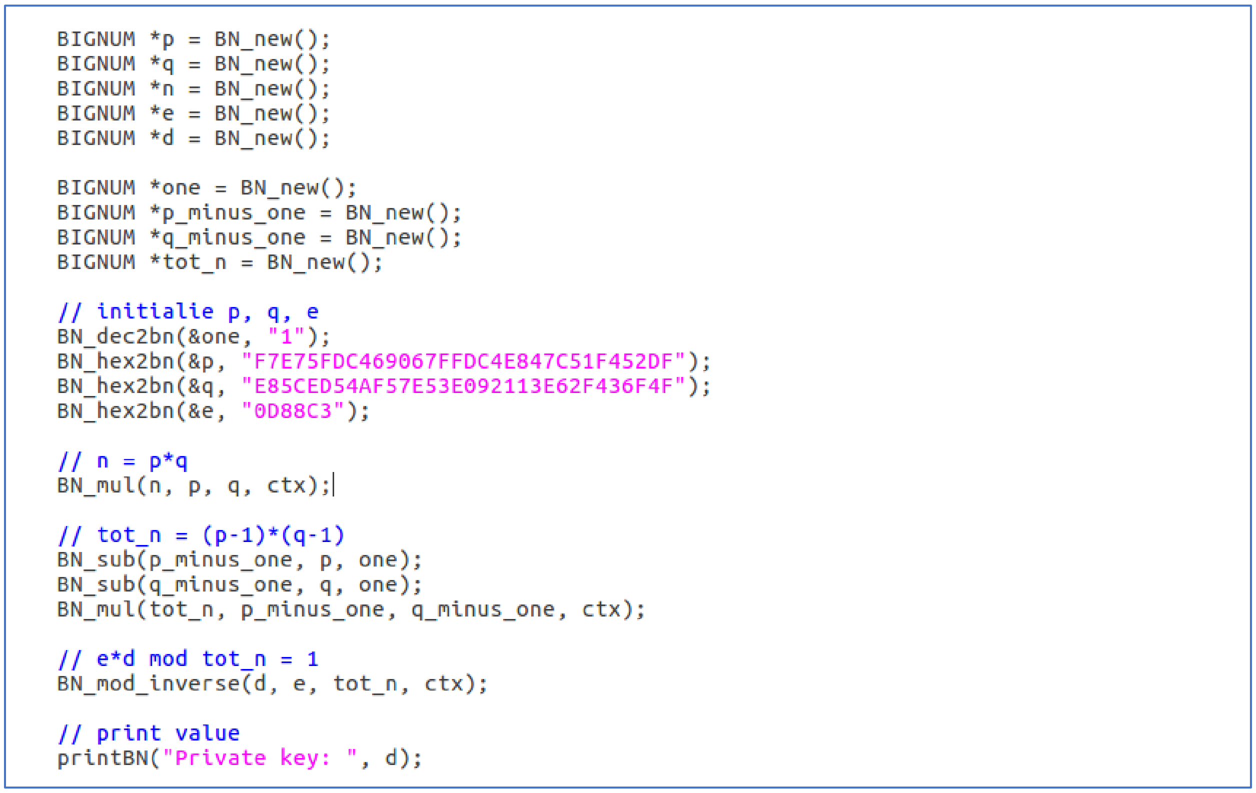
RSA Public-Key Encryption and Signature Lab

Lab 6

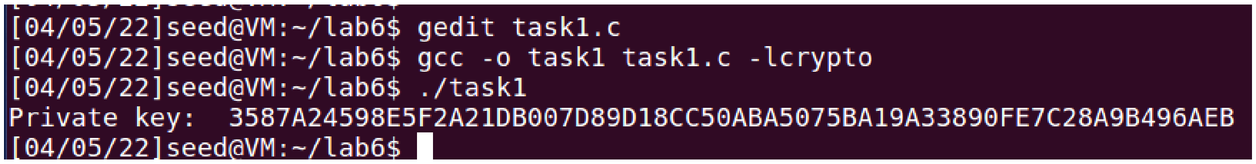
Himanshu Sharma - 202117006

Task 1: Deriving the Private Key

We need to derive the private key from public key (e,n), where n = p*q. This require Private key where e*d mod n = 1. So totient of n (n) = (p-1)*(q-1).



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

BIGNUM *one = BN_new();
BIGNUM *p_minus_one = BN_new();
BIGNUM *q_minus_one = BN_new();
BIGNUM *tot_n = BN_new();

// initialie p, q, e
BN_dec2bn(&one, "1");
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3");

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

// tot_n = (p-1)*(q-1)
BN_sub(p_minus_one, p, one);
BN_sub(q_minus_one, q, one);
BN_mul(tot_n, p_minus_one, q_minus_one, ctx);

// e*d mod tot_n = 1
BN_mod_inverse(d, e, tot_n, ctx);

// print value
printBN("Private key: ", d);
```

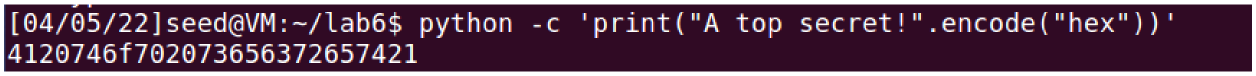


```
[04/05/22]seed@VM:~/lab6$ gedit task1.c
[04/05/22]seed@VM:~/lab6$ gcc -o task1 task1.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task1
Private key:  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[04/05/22]seed@VM:~/lab6$ 
```

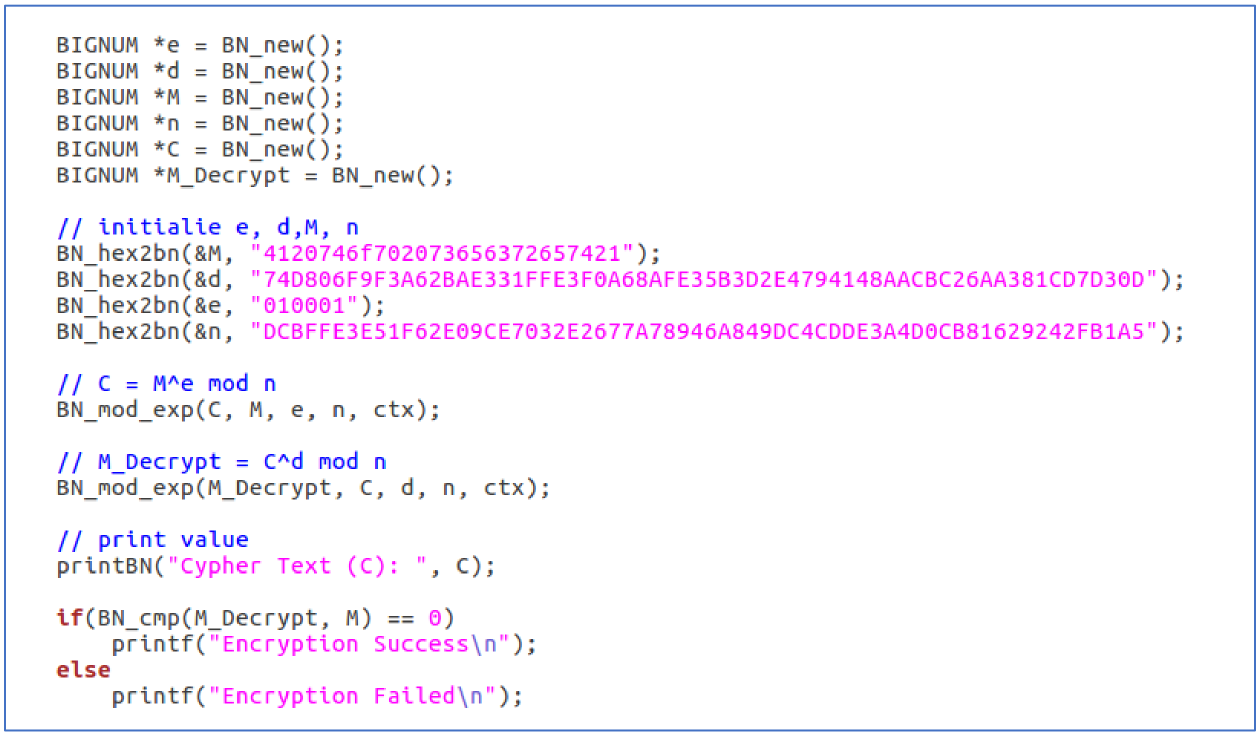
Task 2: Encrypting the message

We are tasked to encrypt “A top secret!” using RSA algorithm.

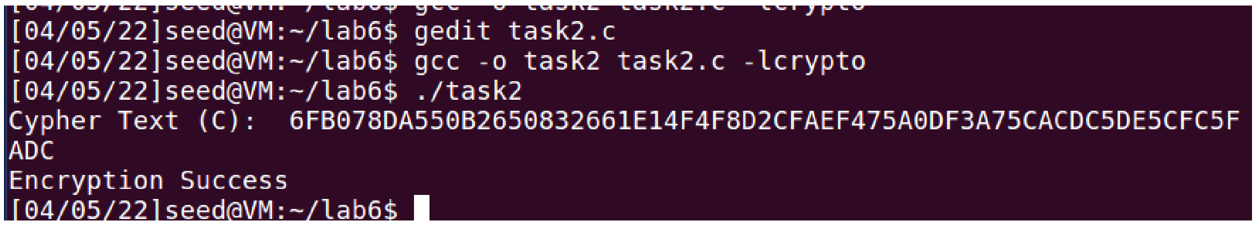
Using the following operation, $C = M^e \bmod n$. and in order to verify our encryption we are also given private key d. We tend to verify by comparing the original message and decrypted message $M = C^d \bmod n$.



```
[04/05/22]seed@VM:~/lab6$ python -c 'print("A top secret!".encode("hex"))'
4120746f702073656372657421
```



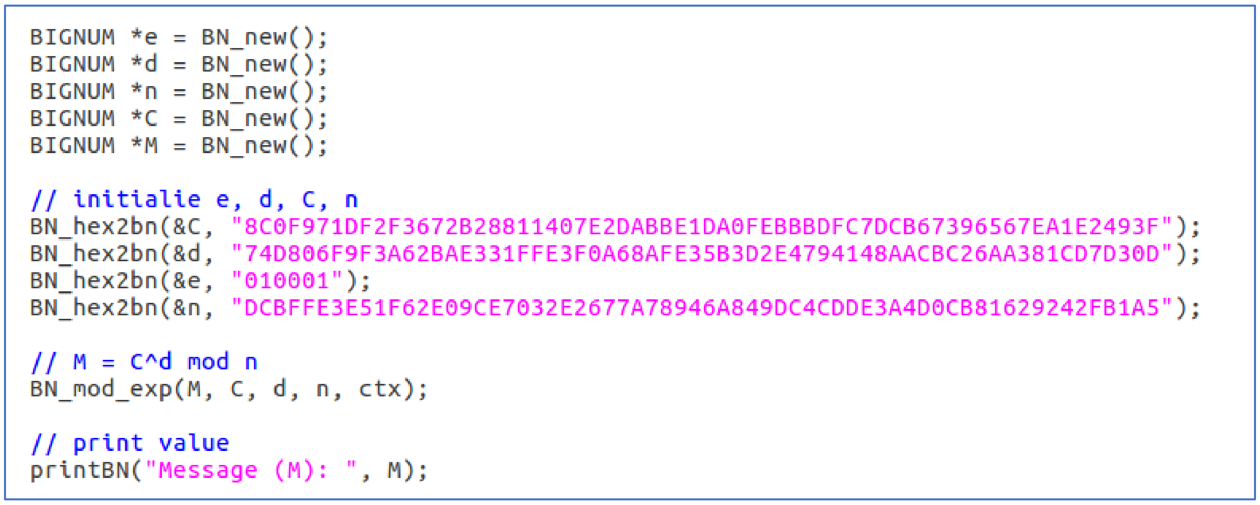
```
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *C = BN_new();
    BIGNUM *M_Decrypt = BN_new();

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

    // C = M^e mod n
    BN_mod_exp(C, M, e, n, ctx);

    // M_Decrypt = C^d mod n
    BN_mod_exp(M_Decrypt, C, d, n, ctx);

    // print value
    printBN("Cypher Text (C): ", C);

    if(BN_cmp(M_Decrypt, M) == 0)
        printf("Encryption Success\n");
    else
        printf("Encryption Failed\n");
```



```
[04/05/22]seed@VM:~/lab6$ gedit task2.c
[04/05/22]seed@VM:~/lab6$ gcc -o task2 task2.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task2
Cypher Text (C):  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5F
ADC
Encryption Success
[04/05/22]seed@VM:~/lab6$
```

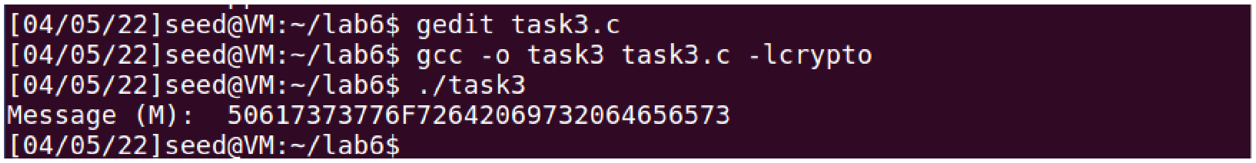
Task 3: Decrypting a Message

Using the same keys from the previous task, we will decrypt the cyphertext. The value taken are same as Task 2



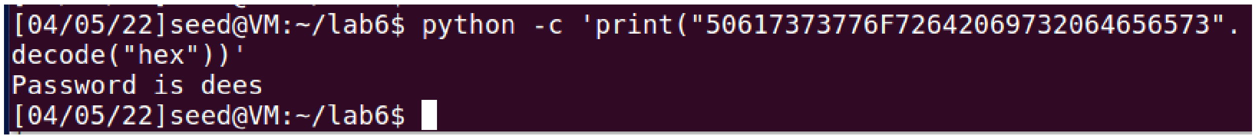
```
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
BIGNUM *n = BN_new();
BIGNUM *C = BN_new();
BIGNUM *M = BN_new();

// initialie e, d, C, n
BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&e, "010001");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

// M = C^d mod n
BN_mod_exp(M, C, d, n, ctx);

// print value
printBN("Message (M): ", M);
```

After executing the program we obtained the original message back in hexadecimal format.



```
[04/05/22]seed@VM:~/lab6$ gedit task3.c
[04/05/22]seed@VM:~/lab6$ gcc -o task3 task3.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task3
Message (M):  50617373776F72642069732064656573
[04/05/22]seed@VM:~/lab6$
```

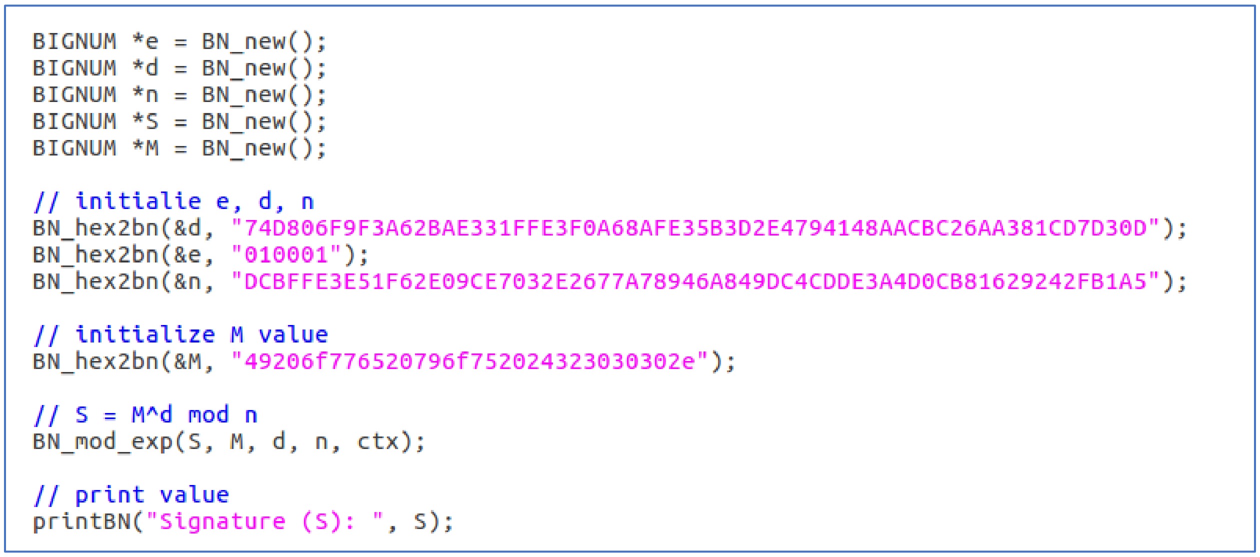
We will convert obtained hexadecimal values to character format. To read the content of the message.



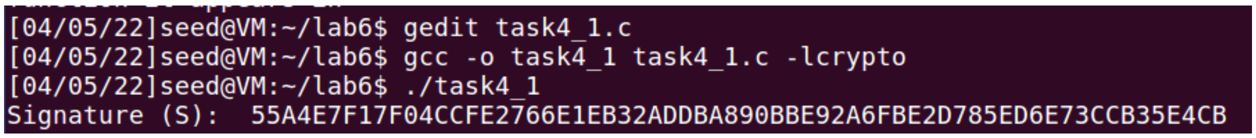
```
[04/05/22]seed@VM:~/lab6$ python -c 'print("50617373776F72642069732064656573".
decode("hex"))'
Password is dees
[04/05/22]seed@VM:~/lab6$
```

Task 4: Signing a Message

In this task we are given two different messages which differ only by single character. Both of them will be encrypted with same encryption scheme.

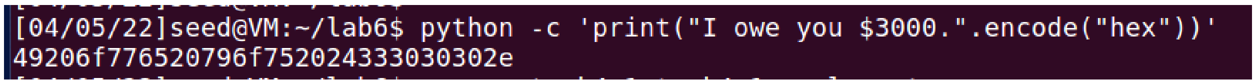


```
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
BIGNUM *n = BN_new();
BIGNUM *S = BN_new();
BIGNUM *M = BN_new();

// initialie e, d, n
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&e, "010001");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

// initialize M value
BN_hex2bn(&M, "49206f776520796f752024323030302e");

// S = M^d mod n
BN_mod_exp(S, M, d, n, ctx);

// print value
printBN("Signature (S): ", S);
```



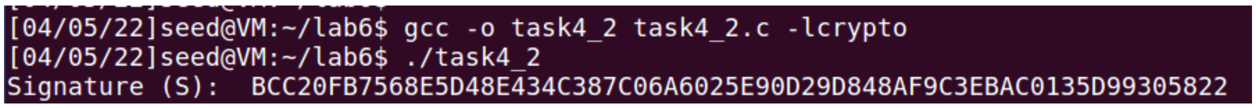
```
[04/05/22]seed@VM:~/lab6$ gedit task4_1.c
[04/05/22]seed@VM:~/lab6$ gcc -o task4_1 task4_1.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task4_1
Signature (S):  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
```

When encoding both the string to hexadecimal format the output remains the same despite original message were different.



```
[04/05/22]seed@VM:~/lab6$ python -c 'print("I owe you $3000.".encode("hex"))'
49206f776520796f752024333030302e
```

When the source code is executed for both the messages the output (signature) generated is different. This refers to the collision resistance property of the algorithm.

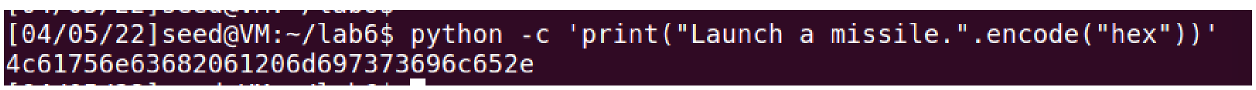


```
[04/05/22]seed@VM:~/lab6$ gcc -o task4_2 task4_2.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task4_2
Signature (S):  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
```

Task 5: Verify a signature

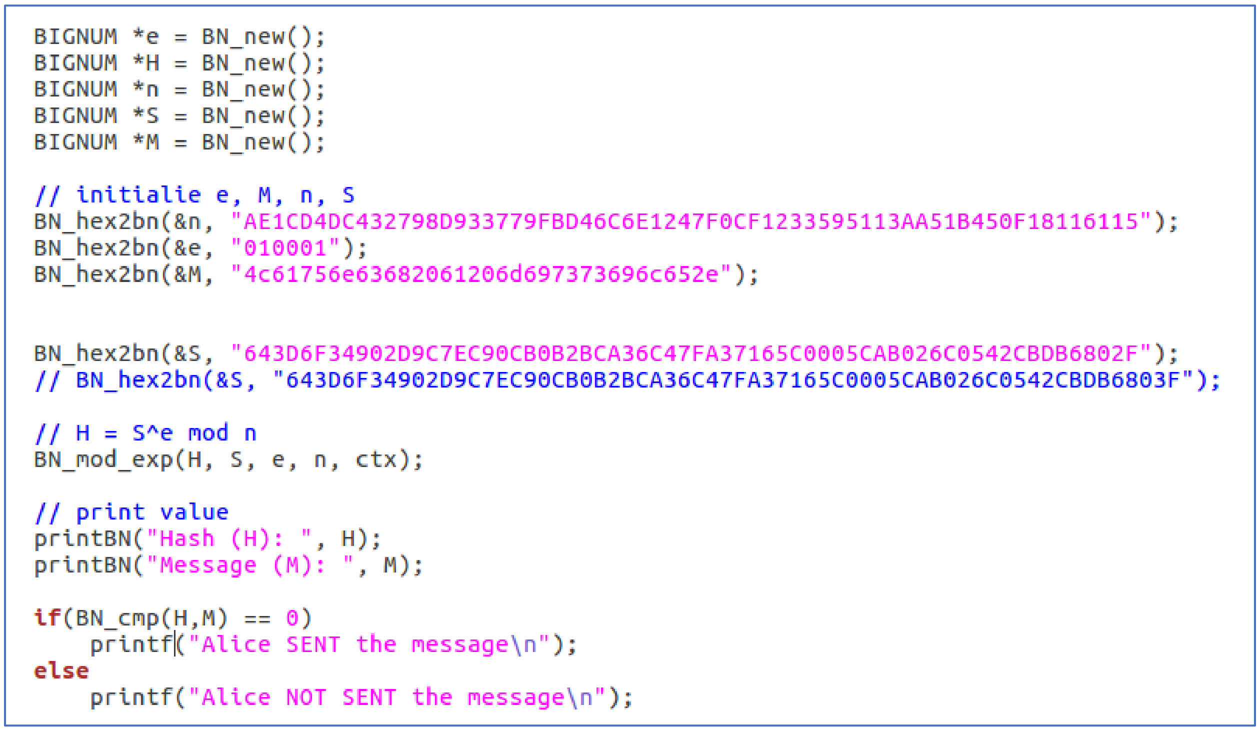
We need to verify the signature S of message from Alice.To verify it, the message was sent by the Alice if M = S^e mod n, where (e,n) is Alice’s public key.

We convert original message “Launch a missile.” to the HEX value for encryption.

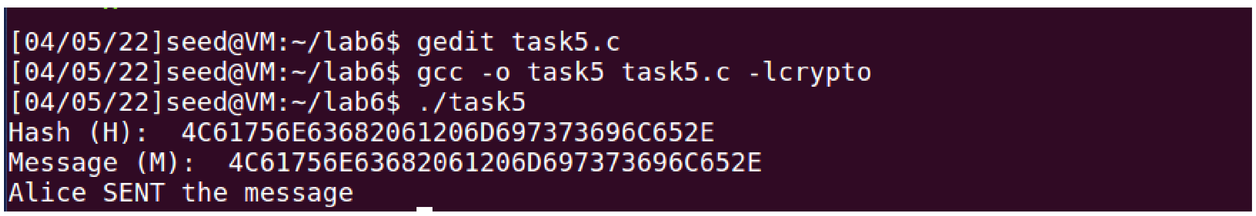


```
[04/05/22]seed@VM:~/lab6$ python -c 'print("Launch a missile.".encode("hex"))'
4c61756e63682061206d697373696c652e
```

Source code:

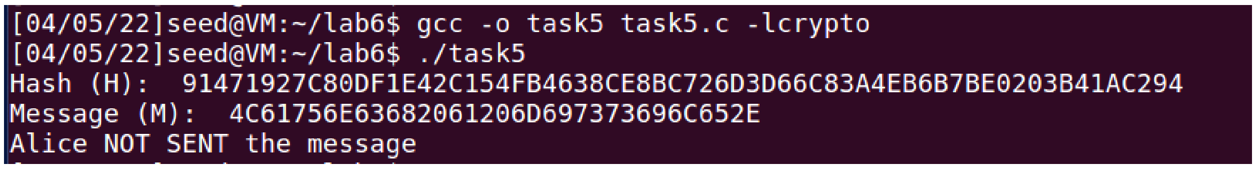


```
BIGNUM *e = BN_new();
BIGNUM *H = BN_new();
BIGNUM *n = BN_new();
BIGNUM *S = BN_new();
BIGNUM *M = BN_new();

// initialie e, M, n, S
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&e, "010001");
BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");


BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
// BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

// H = S^e mod n
BN_mod_exp(H, S, e, n, ctx);

// print value
printBN("Hash (H): ", H);
printBN("Message (M): ", M);

if(BN_cmp(H,M) == 0)
    printf("Alice SENT the message\n");
else
    printf("Alice NOT SENT the message\n");
```



```
[04/05/22]seed@VM:~/lab6$ gedit task5.c
[04/05/22]seed@VM:~/lab6$ gcc -o task5 task5.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task5
Hash (H):  4C61756E63682061206D697373696C652E
Message (M):  4C61756E63682061206D697373696C652E
Alice SENT the message
```

Changing a single byte of signature from Alice, corrupted the signature. Mismatch between expected message and original message, state no guarantee of message sender authenticity.

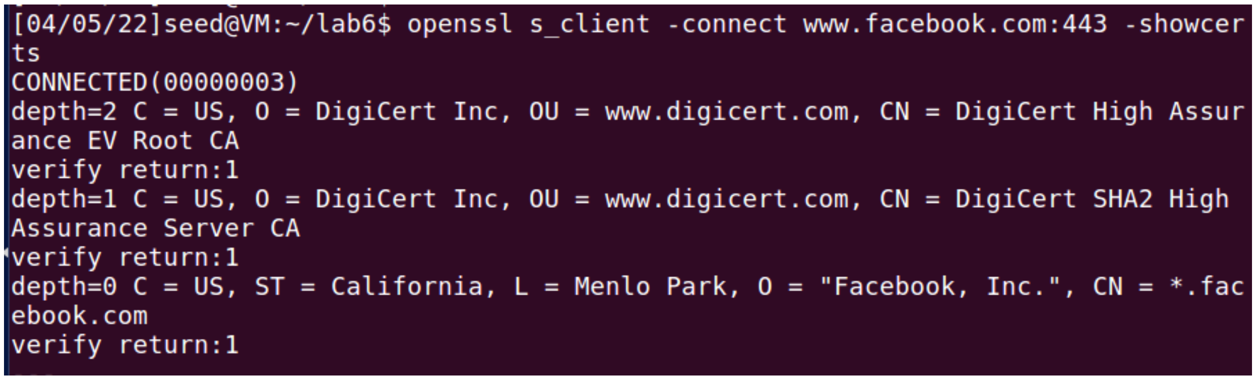


```
[04/05/22]seed@VM:~/lab6$ gcc -o task5 task5.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task5
Hash (H):  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
Message (M):  4C61756E63682061206D697373696C652E
Alice NOT SENT the message
```

Task 6: Manually Verifying an X.509 Certificate

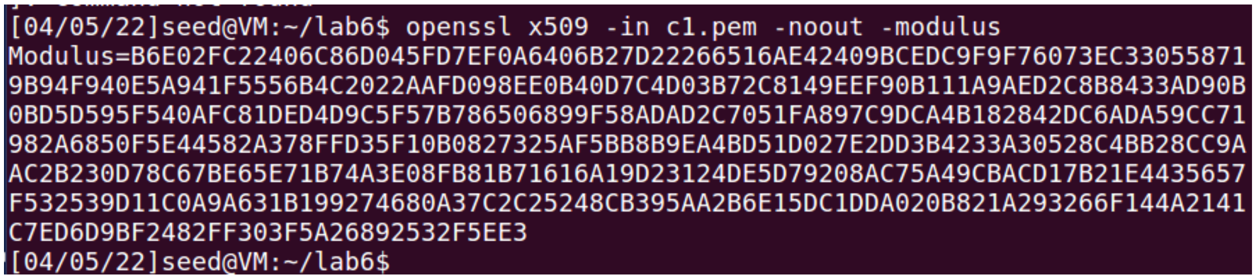
Download a certificate from the real server. In this case, I have used URL: Facebook.com.

Then we extracted the two-certificate obtained in two different files (c0.pem and c1.pem).



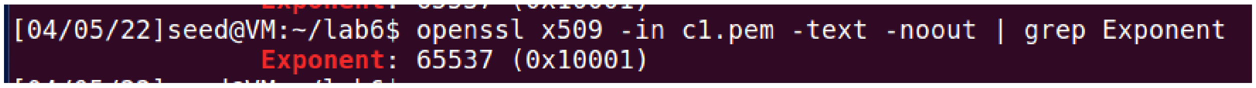
```
[04/05/22]seed@VM:~/lab6$ openssl s_client -connect www.facebook.com:443 -showcer
ts
CONNECTED(00000003)
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assur
ance EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High
Assurance Server CA
verify return:1
depth=0 C = US, ST = California, L = Menlo Park, O = "Facebook, Inc.", CN = *.fac
ebook.com
verify return:1
```

Here are we trying to extract the modulus n from the x509 certificate



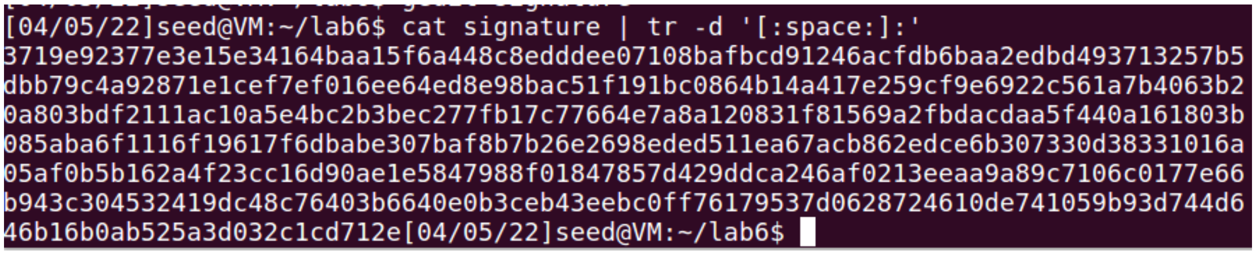
```
[04/05/22]seed@VM:~/lab6$ openssl x509 -in c1.pem -noout -modulus
Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC33055871
9B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90B
0BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC71
982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528C4BB28CC9A
AC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E4435657
F532539D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2141
C7ED6D9BF2482FF303F5A26892532F5EE3
[04/05/22]seed@VM:~/lab6$
```

Print all attribute and finding the exponent.



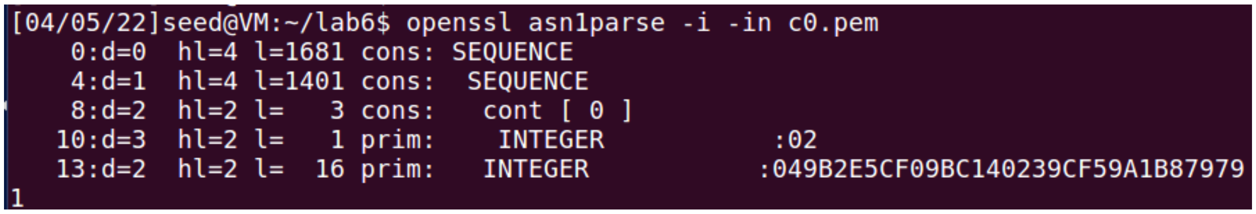
```
[04/05/22]seed@VM:~/lab6$ openssl x509 -in c1.pem -text -noout | grep Exponent
                Exponent: 65537 (0x10001)
```

To extract the signature from the certificate, we executed "***openssl x509 -in c0.pem -text -noout***" and formatted the output by trimming spaces and colons.

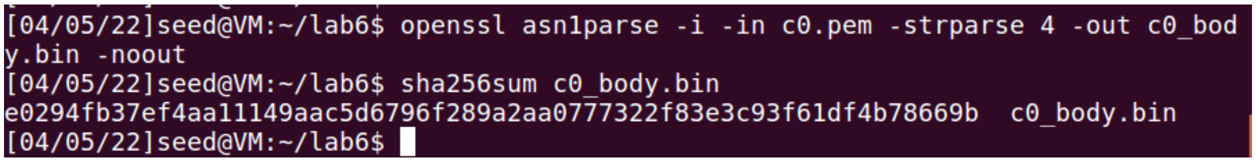


```
[04/05/22]seed@VM:~/lab6$ cat signature | tr -d '[:space:]:'
3719e92377e3e15e34164baa15f6a448c8edddee07108bafbcd91246acfdb6baa2edbd493713257b5
dbb79c4a92871e1cef7ef016ee64ed8e98bac51f191bc0864b14a417e259cf9e6922c561a7b4063b2
0a803bdf2111ac10a5e4bc2b3bec277fb17c77664e7a8a120831f81569a2fbdacdaa5f440a161803b
085aba6f1116f19617f6dbabe307baf8b7b26e2698eded511ea67acb862edce6b307330d38331016a
05af0b5b162a4f23cc16d90ae1e5847988f01847857d429ddca246af0213eeaa9a89c7106c0177e66
b943c304532419dc48c76403b6640e0b3ceb43eebc0ff76179537d0628724610de741059b93d744d6
46b16b0ab525a3d032c1cd712e[04/05/22]seed@VM:~/lab6$
```

Below two screenshot shows how to extract the body of certificate which is used to generate the hash. And get the field from c0.pem and then calculated sha256sum.

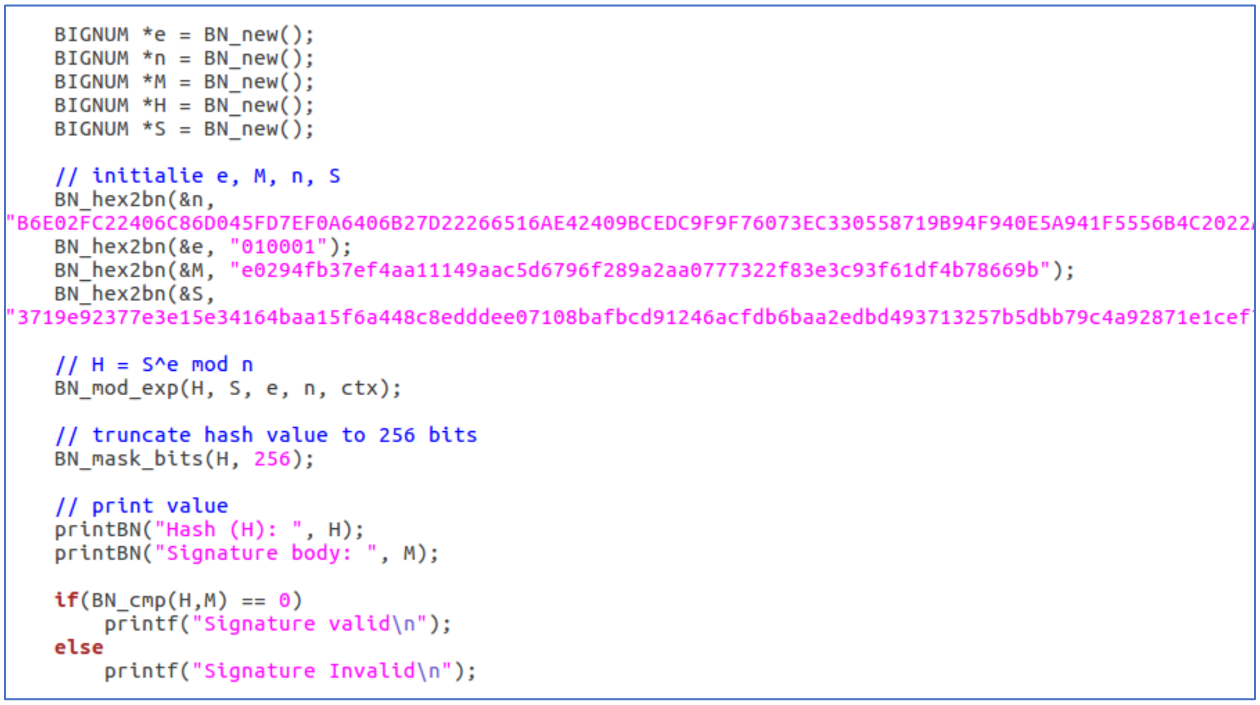


```
[04/05/22]seed@VM:~/lab6$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1681 cons: SEQUENCE
    4:d=1  hl=4 l=1401 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER           :049B2E5CF09BC140239CF59A1B87979
1
```



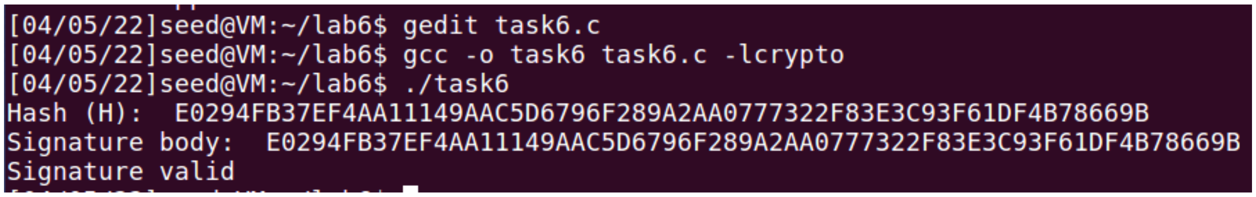
```
[04/05/22]seed@VM:~/lab6$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_bod
y.bin -noout
[04/05/22]seed@VM:~/lab6$ sha256sum c0_body.bin
e0294fb37ef4aa11149aac5d6796f289a2aa0777322f83e3c93f61df4b78669b  c0_body.bin
[04/05/22]seed@VM:~/lab6$ 
```

The above generated values (modulus, private key, hash, public key) are collected and stored in the given source code at appropriate places.



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

    // initialie e, M, n, S
    BN_hex2bn(&n,
"B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330558719B94F940E5A941F5556B4C2022
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&M, "e0294fb37ef4aa11149aac5d6796f289a2aa0777322f83e3c93f61df4b78669b");
    BN_hex2bn(&S,
"3719e92377e3e15e34164baa15f6a448c8edddee07108bafbcd91246acfdb6baa2edbd493713257b5dbb79c4a92871e1cef

    // H = S^e mod n
    BN_mod_exp(H, S, e, n, ctx);

    // truncate hash value to 256 bits
    BN_mask_bits(H, 256);

    // print value
    printBN("Hash (H): ", H);
    printBN("Signature body: ", M);

    if(BN_cmp(H,M) == 0)
        printf("Signature valid\n");
    else
        printf("Signature Invalid\n");
```

Executing the above program, we can state that the downloaded certificate is **valid**.



```
[04/05/22]seed@VM:~/lab6$ gedit task6.c
[04/05/22]seed@VM:~/lab6$ gcc -o task6 task6.c -lcrypto
[04/05/22]seed@VM:~/lab6$ ./task6
Hash (H):  E0294FB37EF4AA11149AAC5D6796F289A2AA0777322F83E3C93F61DF4B78669B
Signature body:  E0294FB37EF4AA11149AAC5D6796F289A2AA0777322F83E3C93F61DF4B78669B
Signature valid
```