PROJECT 2: RSA Public-Key Encryption and Signature

Objective: The objective of this project is to gain hands-on experience on the RSA algorithm.

Task 1: Deriving the Private Key:

As we know, we can derive a private key by using two large prime numbers (p, q) and a modulus (e). The first task uses the following given p, q and e:
p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3

To find the private key, we are doing below steps in our code by passing BIGNUM objects into BN() functions, as you can see generate_private_key() function below.

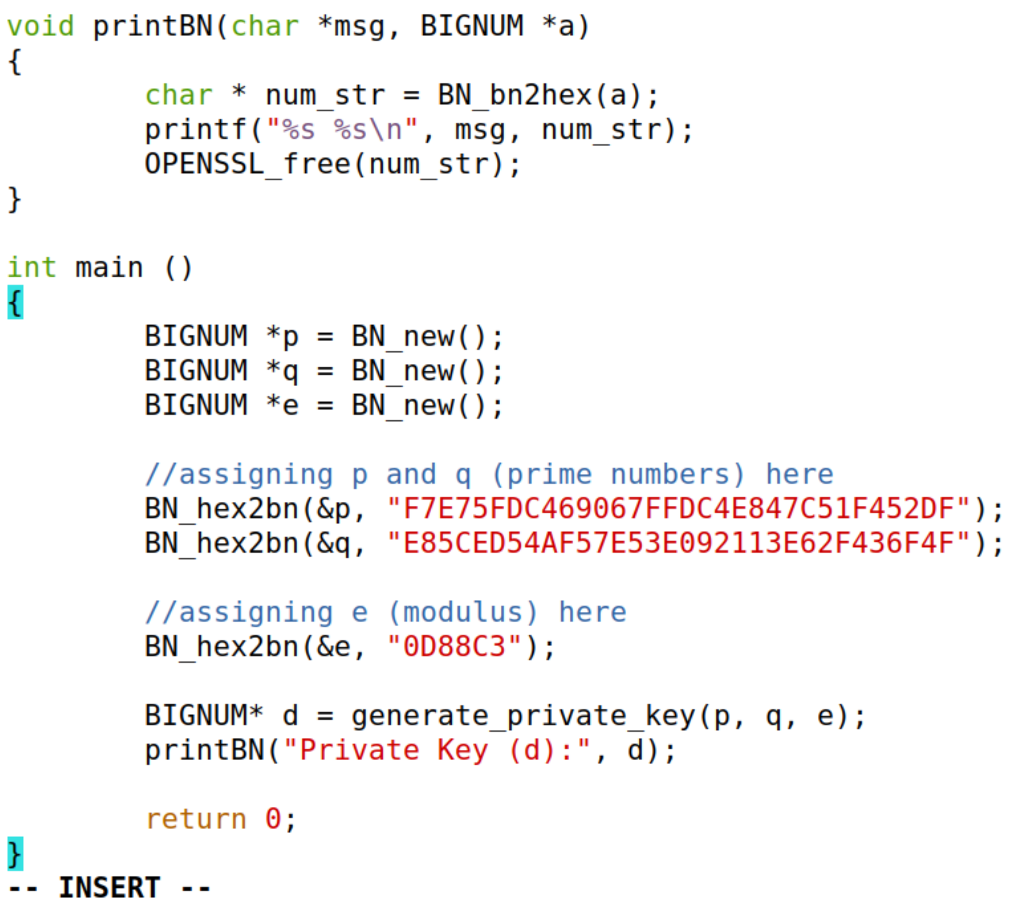
First, we are converting the given values from hexadecimal to binary using BN_hex2bn() function. Second, we are passing those binaries to BN() functions to compute below steps.

a. Calculate Ø(n) = (p-1) * (q-1)
b. Since e is given, private key (d) = (1/e) mod Ø(n)

Public Key: {e, n}, where n=p*q;
Private Key: {d, n}

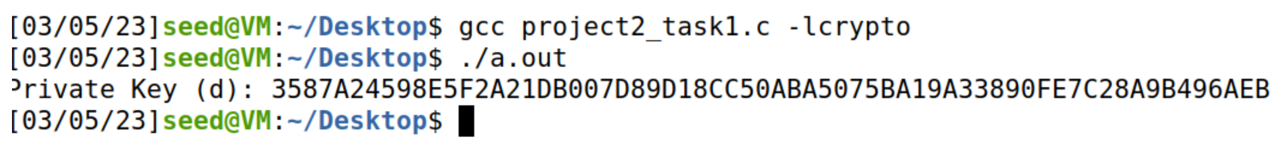
Task-1: Code

```
//TASK1

#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 512

BIGNUM* generate_private_key(BIGNUM* p, BIGNUM* q, BIGNUM* e)
{
        //In this function, we are accepting two large prime numbers
        //and computing a private key.
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM* sub_op1 = BN_new();
        BIGNUM* sub_op2 = BN_new();
        BIGNUM* one = BN_new();
        BIGNUM* mul_op = BN_new();
        BIGNUM* mod_inv_op = BN_new();

        BN_dec2bn(&one, "1");
        BN_sub(sub_op1, p, one);
        BN_sub(sub_op2, q, one);
        BN_mul(mul_op, sub_op1, sub_op2, ctx);

        BN_mod_inverse(mod_inv_op, e, mul_op, ctx);
        BN_CTX_free(ctx);
        return mod_inv_op;
}
```



```
void printBN(char *msg, BIGNUM *a)
{
        char * num_str = BN_bn2hex(a);
        printf("%s %s\n", msg, num_str);
        OPENSSL_free(num_str);
}

int main ()
{
        BIGNUM *p = BN_new();
        BIGNUM *q = BN_new();
        BIGNUM *e = BN_new();

        //assigning p and q (prime numbers) here
        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

        //assigning e (modulus) here
        BN_hex2bn(&e, "0D88C3");

        BIGNUM* d = generate_private_key(p, q, e);
        printBN("Private Key (d):", d);

        return 0;
}
-- INSERT --
```

<u>Output</u>: private key (d)

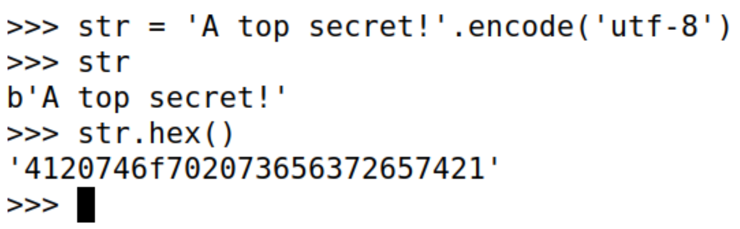


```
[03/05/23]seed@VM:~/Desktop$ gcc project2_task1.c -lcrypto
[03/05/23]seed@VM:~/Desktop$ ./a.out
Private Key (d): 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[03/05/23]seed@VM:~/Desktop$ 
```

Task 2: Encrypting a message:

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

First, we need to convert the given message M = "A top secret!" to hexadecimal using Python.

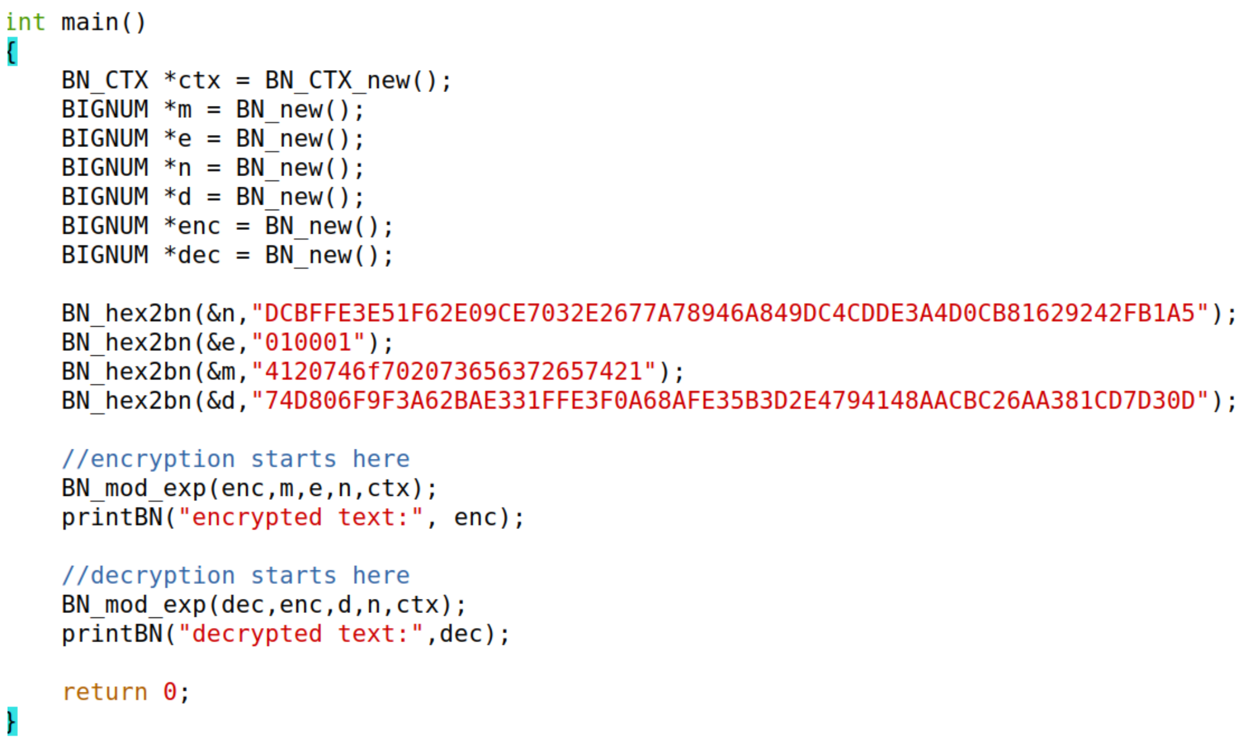


```
>>> str = 'A top secret!'.encode('utf-8')
>>> str
b'A top secret!'
>>> str.hex()
'4120746f702073656372657421'
>>> 
```

Then after getting the hexadecimal, we need to convert all hexadecimal variables into binary using BN_hex2bn() function and pass M, e and n to BN_mod_exp() function for encryption (C=M^e mod n) as you see in the below code.

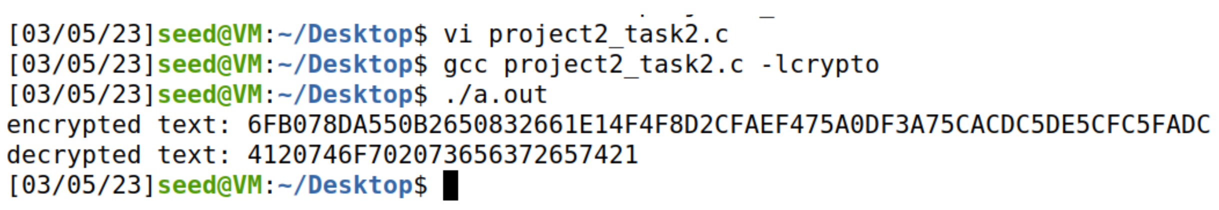
And since, we are also provided the variable ‘d’, so we decrypted the ciphertext (M=C^d mod n) by using BN_mod_exp() function again to verify our encryption result.

Task-2 code:



```
int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *m = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

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

    //encryption starts here
    BN_mod_exp(enc,m,e,n,ctx);
    printBN("encrypted text:", enc);

    //decryption starts here
    BN_mod_exp(dec,enc,d,n,ctx);
    printBN("decrypted text:",dec);

    return 0;
}
```

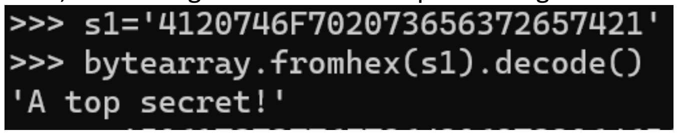
Compiling and running the above task2.c code:



```
[03/05/23]seed@VM:~/Desktop$ vi project2_task2.c
[03/05/23]seed@VM:~/Desktop$ gcc project2_task2.c -lcrypto
[03/05/23]seed@VM:~/Desktop$ ./a.out
encrypted text: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
decrypted text: 4120746F702073656372657421
[03/05/23]seed@VM:~/Desktop$
```

As you can, the decrypted text matches with the hexadecimal and the message.

Now, converting hexadecimal to a plain string:



```
>>> s1='4120746F702073656372657421'
>>> bytearray.fromhex(s1).decode()
'A top secret!'
```

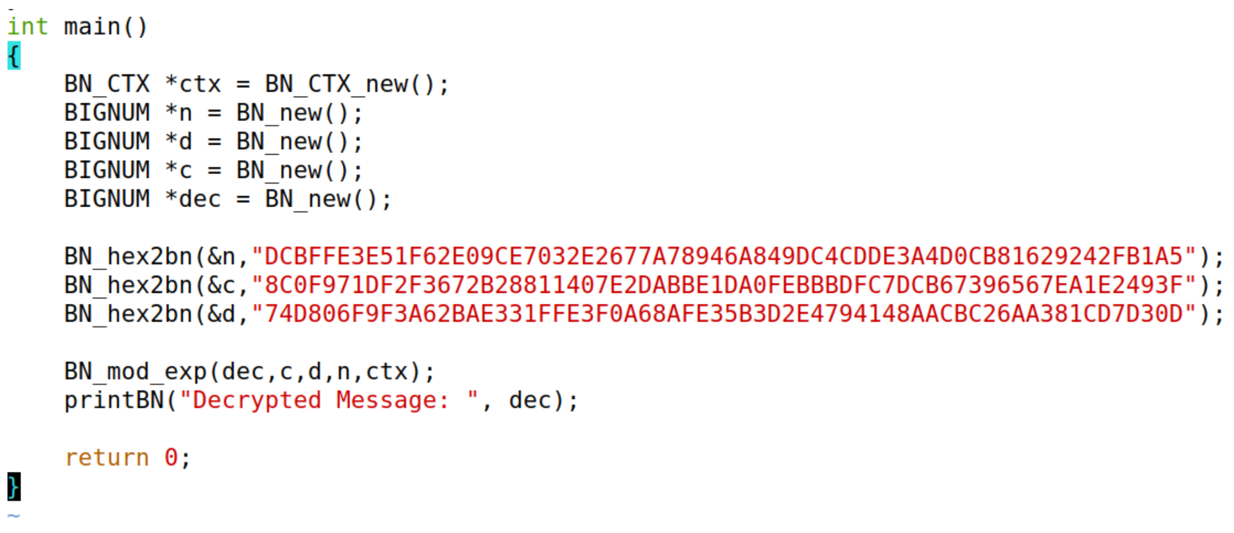
So, we can see the decrypted message is matching. Hence, we verified our encryption result.

Task 3: Decrypting a message:

Given,
C= 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F,
Public Key and Private Key are same as in Task-2.

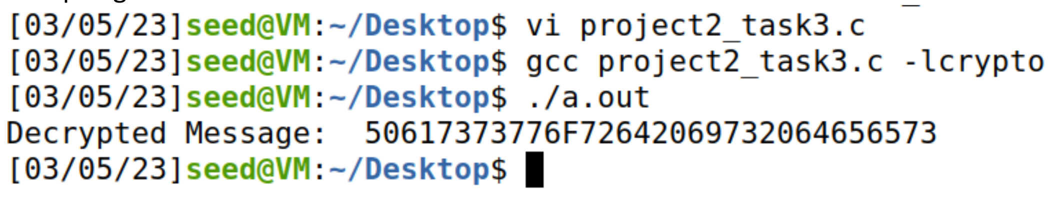
In this task, we need to decrypt the above ciphertext and convert it back to a plain ASCII string. So, we will again use BN_mod_exp() function to perform the decryption (M=C^d mod n) but before that we will convert the given values from hexadecimal to binary, using BN_hex2bn() function.

Task-3 Code:



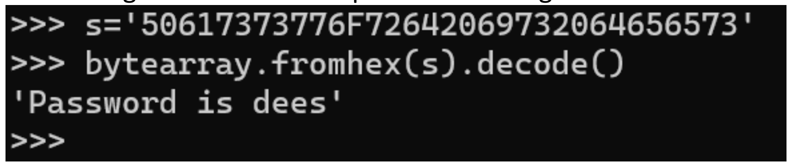
```
int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *c = BN_new();
    BIGNUM *dec = BN_new();

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

    BN_mod_exp(dec,c,d,n,ctx);
    printBN("Decrypted Message: ", dec);

    return 0;
}
~
```

Compiling and run the above task3.c code:



```
[03/05/23]seed@VM:~/Desktop$ vi project2_task3.c
[03/05/23]seed@VM:~/Desktop$ gcc project2_task3.c -lcrypto
[03/05/23]seed@VM:~/Desktop$ ./a.out
Decrypted Message:  50617373776F72642069732064656573
[03/05/23]seed@VM:~/Desktop$ 
```

Converting hexadecimal to a plain ASCII string.



```
>>> s='50617373776F72642069732064656573'
>>> bytearray.fromhex(s).decode()
'Password is dees'
>>>
```

So, the message in string is "**Password is dees**".

Task 4: Signing a message:

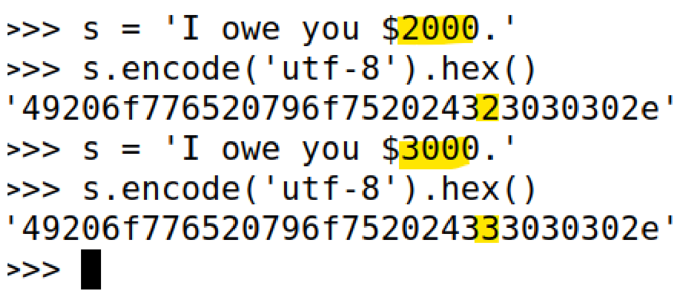
Given,

M = I owe you $2000.

The Public /private keys used in this task are same as the ones used in Task-2.

To perform this task, we will just do encryption using the givens.

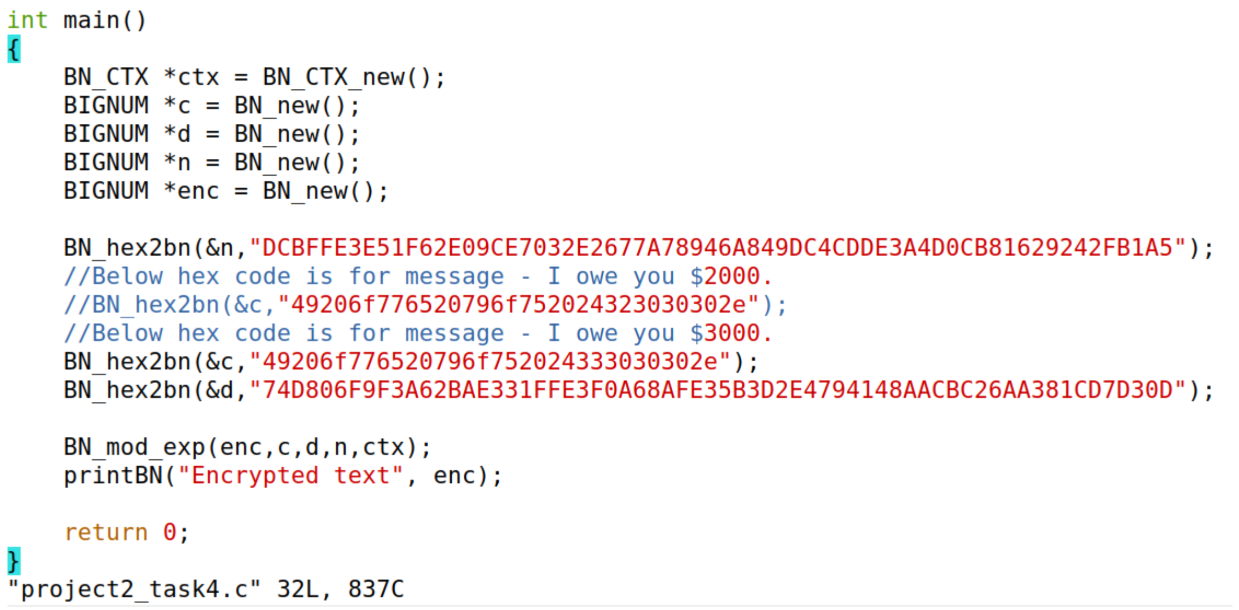
First, we convert the messages into hexadecimal. There are two text messages shown below with only one difference (see highlighted text).



```
>>> s = 'I owe you $2000.'
>>> s.encode('utf-8').hex()
'49206f776520796f75202432303030302e'
>>> s = 'I owe you $3000.'
>>> s.encode('utf-8').hex()
'49206f776520796f75202433303030302e'
>>> 
```

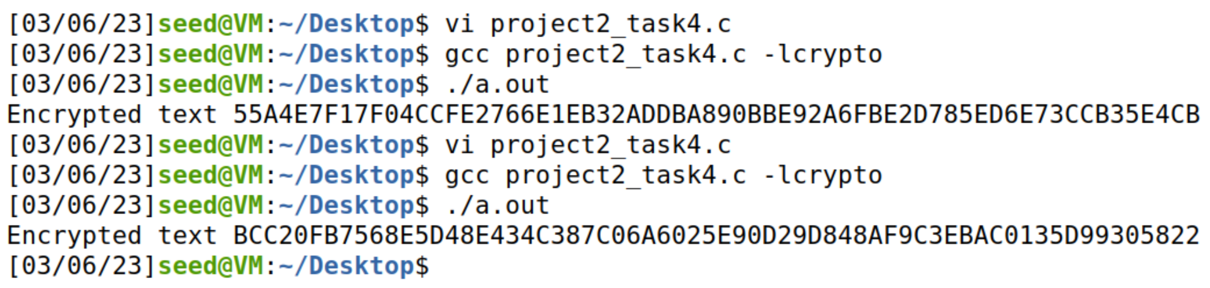
Now, we run the encrypt the above messages, one at a time. Please refer below snapshot:

Task-4 code:



```
int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *c = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *enc = BN_new();

    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    //Below hex code is for message - I owe you $2000.
    //BN_hex2bn(&c,"49206f776520796f752024323030302e");
    //Below hex code is for message - I owe you $3000.
    BN_hex2bn(&c,"49206f776520796f752024333030302e");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    BN_mod_exp(enc,c,d,n,ctx);
    printBN("Encrypted text", enc);

    return 0;
}
"project2_task4.c" 32L, 837C
```

Compiling and running the above task4.c code:



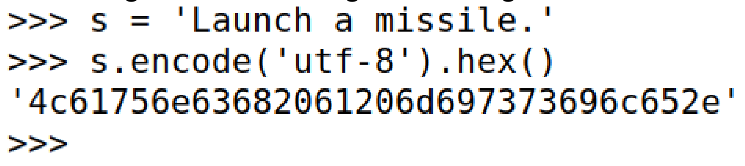
```
[03/06/23]seed@VM:~/Desktop$ vi project2_task4.c
[03/06/23]seed@VM:~/Desktop$ gcc project2_task4.c -lcrypto
[03/06/23]seed@VM:~/Desktop$ ./a.out
Encrypted text 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
[03/06/23]seed@VM:~/Desktop$ vi project2_task4.c
[03/06/23]seed@VM:~/Desktop$ gcc project2_task4.c -lcrypto
[03/06/23]seed@VM:~/Desktop$ ./a.out
Encrypted text BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
[03/06/23]seed@VM:~/Desktop$
```

From the above output, we can conclude, even with only one minor change in text message, the whole signature gets changed. Both signatures are totally different from each other and unique.

Task 5: Verifying a Signature:

Given,
Bob receives a message 'M' from Alice, with her signature 'S'.
M = Launch a missile.
S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
e = 010001 (this hex value equals to decimal 65537)
n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115

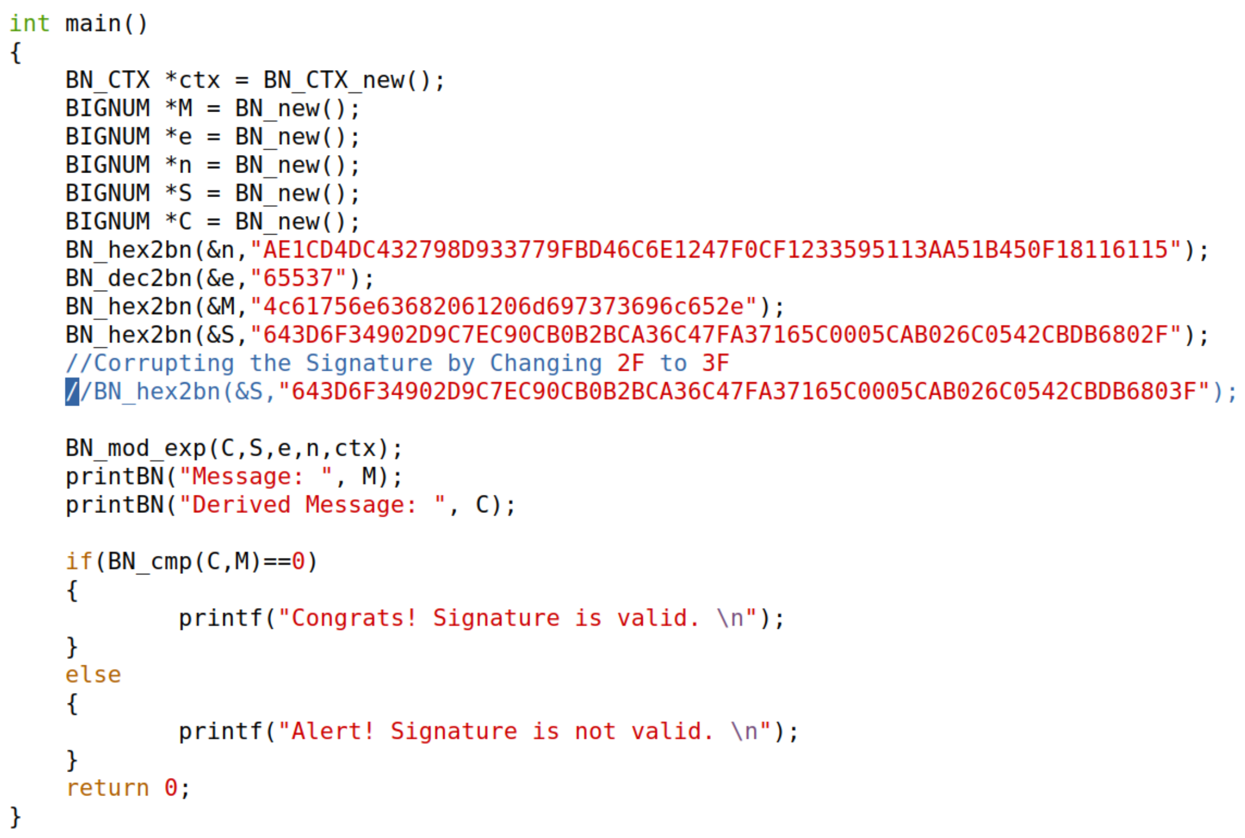
Converting the above message from String to Hexadecimal.



```
>>> s = 'Launch a missile.'
>>> s.encode('utf-8').hex()
'4c61756e63682061206d697373696c652e'
>>>
```

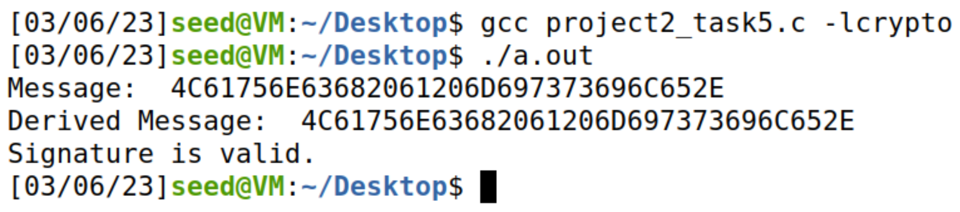
Using Alice's Signature to compute the original message. (C=S^e mod n) and then using BN_cmp() to perform comparison between the original message and the decrypted message.

Task-5 code:



```
int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *M = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *S = BN_new();
    BIGNUM *C = BN_new();
    BN_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_dec2bn(&e,"65537");
    BN_hex2bn(&M,"4c61756e63682061206d697373696c652e");
    BN_hex2bn(&S,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    //Corrupting the Signature by Changing 2F to 3F
    //BN_hex2bn(&S,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

    BN_mod_exp(C,S,e,n,ctx);
    printBN("Message: ", M);
    printBN("Derived Message: ", C);

    if(BN_cmp(C,M)==0)
    {
            printf("Congrats! Signature is valid. \n");
    }
    else
    {
            printf("Alert! Signature is not valid. \n");
    }
    return 0;
}
```

Compiling and running the above task5.c code:

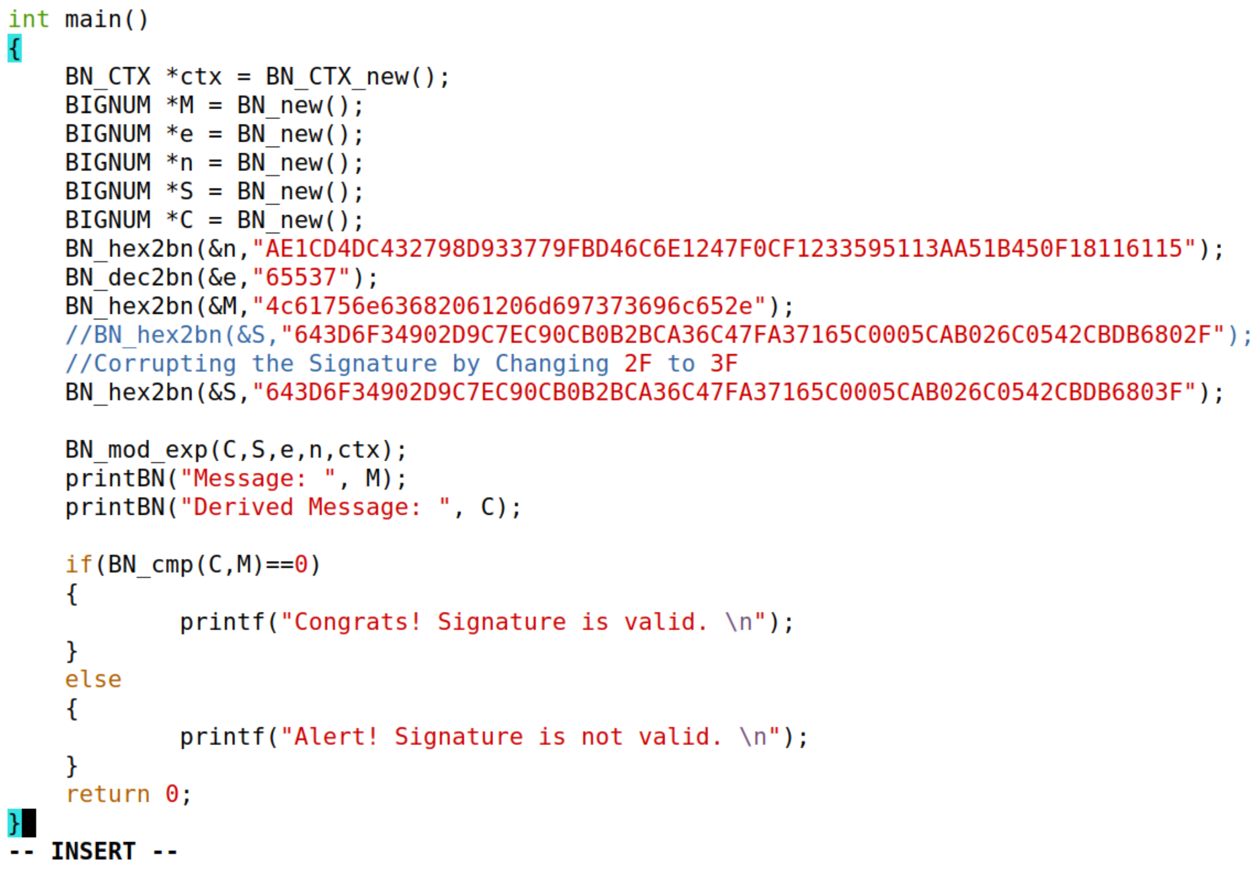


```
[03/06/23]seed@VM:~/Desktop$ gcc project2_task5.c -lcrypto
[03/06/23]seed@VM:~/Desktop$ ./a.out
Message:  4C61756E63682061206D697373696C652E
Derived Message:  4C61756E63682061206D697373696C652E
Signature is valid.
[03/06/23]seed@VM:~/Desktop$ 
```

As you can see in the above output, C & M matches. Hence, it confirms that **it is Alice's signature**.

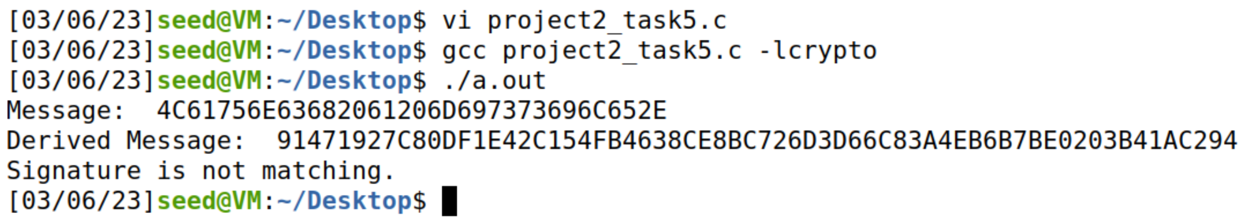
Now, we are corrupting the Signature by changing the last byte from **2F** to **3F** (please refer comments) in the code.

Task-5 code:



```
int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *M = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *S = BN_new();
    BIGNUM *C = BN_new();
    BN_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_dec2bn(&e,"65537");
    BN_hex2bn(&M,"4c61756e63682061206d697373696c652e");
    //BN_hex2bn(&S,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    //Corrupting the Signature by Changing 2F to 3F
    BN_hex2bn(&S,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

    BN_mod_exp(C,S,e,n,ctx);
    printBN("Message: ", M);
    printBN("Derived Message: ", C);

    if(BN_cmp(C,M)==0)
    {
            printf("Congrats! Signature is valid. \n");
    }
    else
    {
            printf("Alert! Signature is not valid. \n");
    }
    return 0;
}
-- INSERT --
```

Compiling the updated code and running it again:



```
[03/06/23]seed@VM:~/Desktop$ vi project2_task5.c
[03/06/23]seed@VM:~/Desktop$ gcc project2_task5.c -lcrypto
[03/06/23]seed@VM:~/Desktop$ ./a.out
Message:  4C61756E63682061206D697373696C652E
Derived Message:  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
Signature is not matching.
[03/06/23]seed@VM:~/Desktop$ 
```

As expected, after corrupting the signature, C & M validation fails. Hence, the **Signature is not valid** after the change.

References:

I. https://www.openssl.org/docs/man1.1.1/man3/BN_CTX_new.html
II. bn(3): multiprecision integer arithmetics - Linux man page (die.net)