**Lab 10: RSA Encryption and Signature Lab**

**Task 1: Deriving the Private key**

In this task, they provided the hexadecimal values of p, q, and the exponent e. I need to compute the private key d. This involves modifying a sample program that utilizes the BIGNUM APIs to calculate d using the given values of p, q, and e. Additionally, the modulus n can be calculated by the equation n = p \* q. Here are the given values:

p = F7E75FDC469067FFDC4E847C51F452DF

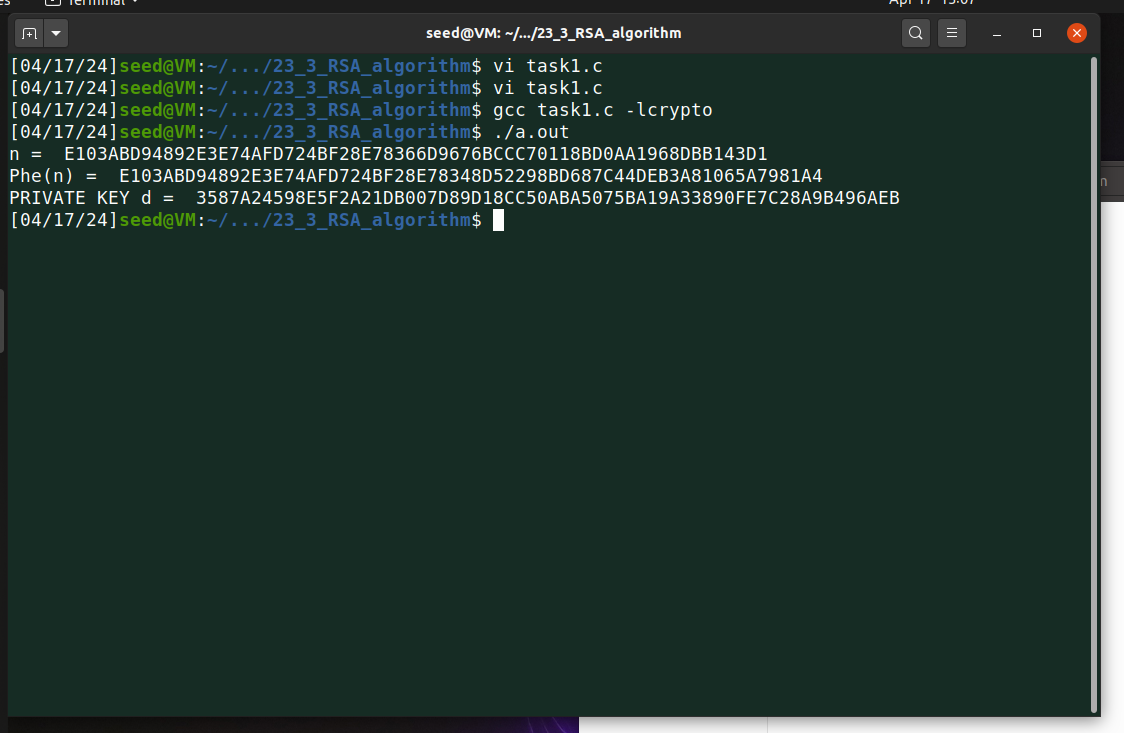
q = E85CED54AF57E53E092113E62F436F4F

e = 0D88C3

**Code:**

| /\* Task1: Finding private key d \*/ #include <stdio.h> #include <openssl/bn.h>  #define NBITS 256  void printBNhex(char \*msg, BIGNUM \* a) {  /\* Use BN\_bn2hex(a) for hex string\*/  char \* number\_str = BN\_bn2hex(a);  printf("%s %s\n", msg, number\_str);  OPENSSL\_free(number\_str); }  int main () {  //Declare variables  BN\_CTX \*ctx = BN\_CTX\_new();  BIGNUM \*p = BN\_new();  BIGNUM \*q = BN\_new();  BIGNUM \*e = BN\_new();  BIGNUM \*pheN = BN\_new();  BIGNUM \*n = BN\_new();  BIGNUM \*d = BN\_new();  BIGNUM \*one = BN\_new();    //initialize variables with given values  BN\_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");  BN\_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");  BN\_hex2bn(&e, "0D88C3");  BN\_dec2bn(&one, "1");   // n = p\*q  BN\_mul(n, p, q, ctx);  printBNhex("n = ", n);    //phe(n)=(p-1)\*(q-1)  BN\_sub(p, p, one);  BN\_sub(q, q, one);  BN\_mul(pheN, p, q, ctx);  printBNhex("Phe(n) = ", pheN);   //calculate d  BN\_mod\_inverse(d, e, pheN, ctx);   printBNhex("PRIVATE KEY d = ", d);   return 0; } |
| --- |

Compiled the above code using gcc and then ran.



And I got the private key: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

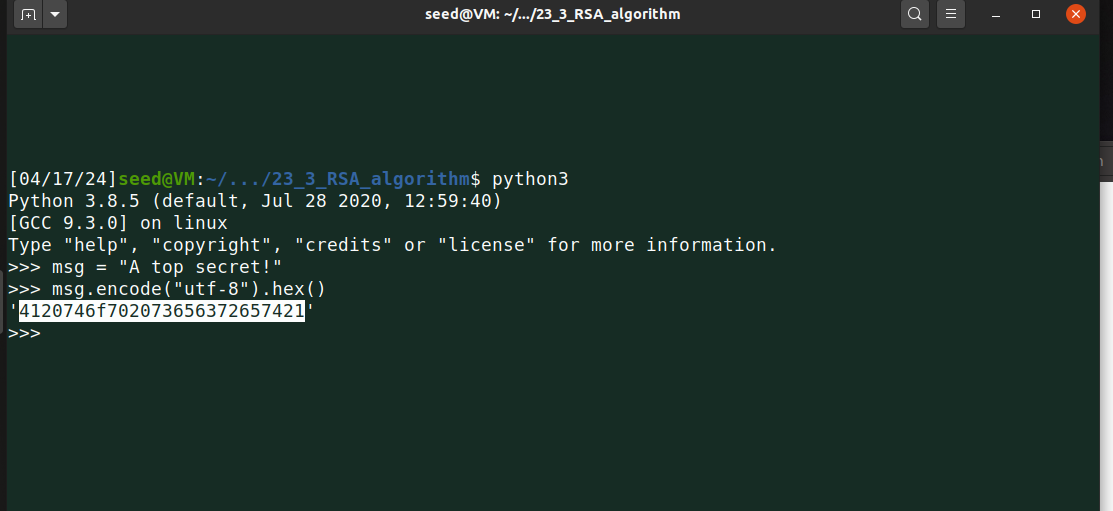
**Task 2: Encrypting a Message**

In this task, they gave the hexadecimal values of the modulus n, exponent e, and private key d. I am asked to encrypt a specific message by first converting it from an ASCII string to a hex string. Using these values along with a sample program, I will proceed to encrypt the message.

The given values of n, e, M, and a are

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

To convert a plain ASCII string to a hex string in Python, you can use the following command:

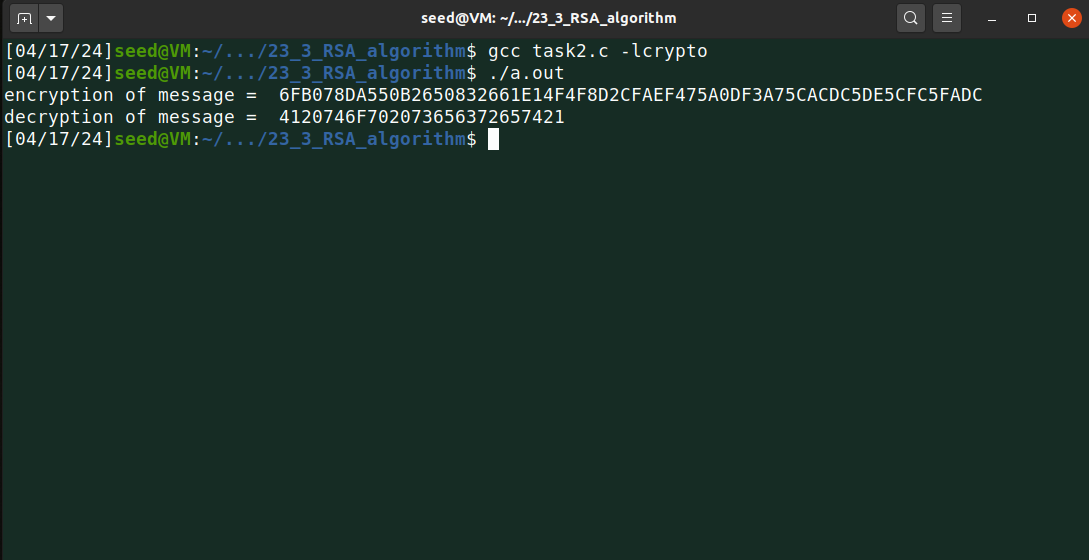


Hexadecimal value of msg is: 4120746f702073656372657421

I used this value in below code and then ran.

**Code:**

| /\* Task2: Finding private key d \*/ #include <stdio.h> #include <openssl/bn.h>  #define NBITS 256  void printBNhex(char \*msg, BIGNUM \* a) {  /\* Use BN\_bn2hex(a) for hex string\*/  char \* number\_str = BN\_bn2hex(a);  printf("%s %s\n", msg, number\_str);  OPENSSL\_free(number\_str); }  int main () {  //Declare variables  BN\_CTX \*ctx = BN\_CTX\_new();  BIGNUM \*e = BN\_new();  BIGNUM \*n = BN\_new();  BIGNUM \*M = BN\_new();  BIGNUM \*c = BN\_new();  BIGNUM \*d = BN\_new();   //Initialize variables with given values  BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");  BN\_hex2bn(&e, "010001");  BN\_hex2bn(&M, "4120746f702073656372657421");  BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");    //Encrypt -> c = M^e mod n   BN\_mod\_exp(c, M, e, n, ctx);  printBNhex("encryption of message = ", c);    //Decrypt -> M = c^d mod n  BN\_mod\_exp(M, c, d, n, ctx);  printBNhex("decryption of message = ", M);   return 0; } |
| --- |



Since I was also provided the private key, I implemented the decryption as well just to ensure that I got the original message back.

**Task 3: Decrypting a Message**

In this task, they provided the hexadecimal values of the modulus n, exponent e, and ciphertext C. I am tasked with decrypting the given ciphertext using these values and a sample program. After decrypting, I will convert the hexadecimal string back into an ASCII string to reveal the original message.

n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

e = 010001 (this hex value equals to decimal 65537)

C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

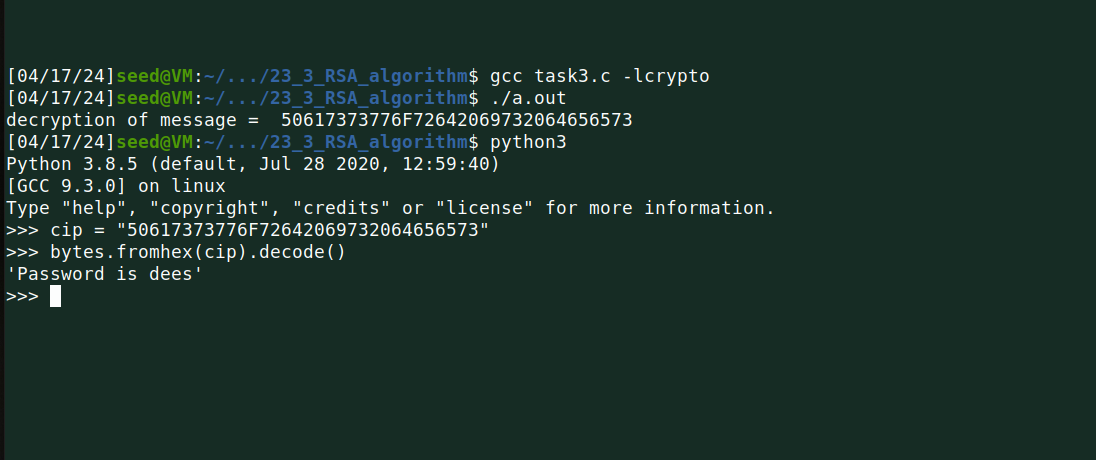
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

**Code:**

| /\* Task3 \*/ #include <stdio.h> #include <openssl/bn.h>  #define NBITS 256  void printBNhex(char \*msg, BIGNUM \* a) {  /\* Use BN\_bn2hex(a) for hex string\*/  char \* number\_str = BN\_bn2hex(a);  printf("%s %s\n", msg, number\_str);  OPENSSL\_free(number\_str); }  int main () {  //Declare variables  BN\_CTX \*ctx = BN\_CTX\_new();  BIGNUM \*e = BN\_new();  BIGNUM \*n = BN\_new();  BIGNUM \*M = BN\_new();  BIGNUM \*c = BN\_new();  BIGNUM \*d = BN\_new();   //Initialize variables with given values  BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");  BN\_hex2bn(&e, "010001");  BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");  BN\_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");     //Decrypt -> M = c^d mod n  BN\_mod\_exp(M, c, d, n, ctx);  printBNhex("decryption of message = ", M);   return 0; } |
| --- |

I compiled the code and then ran it, I got the decryption message.

After that, I used the Python code to convert it back to a plain ASCII string.



Thus, the ciphertext has been successfully decrypted, and the message was 'Password is dees'.

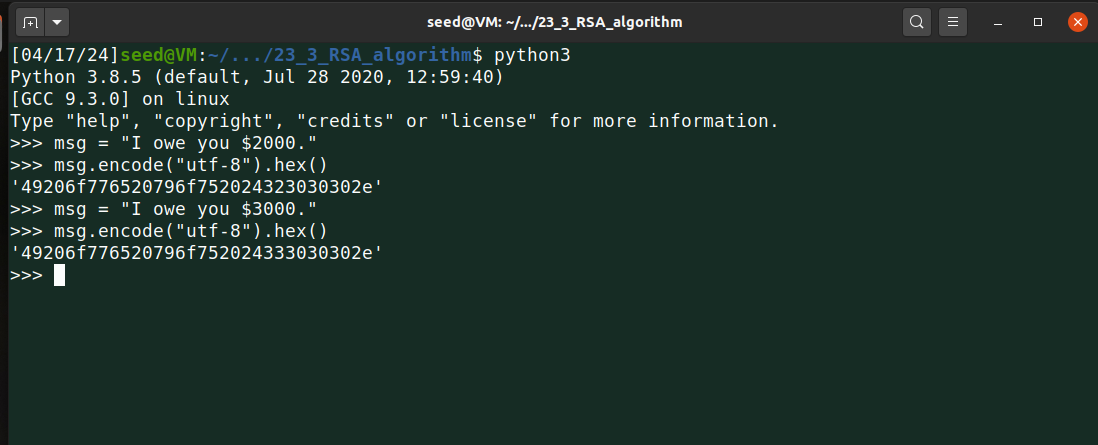
**Task 4: Signing a Message**

In this task, they gave the hexadecimal values of the modulus n, exponent e, private key d, and message M1. My objective is to generate a signature for the given message by directly signing it, rather than signing its hash value. Additionally, I will modify the message by one or two characters, sign the modified message, and then compare both signatures to observe any differences.

Given values are:

* n =DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
* e = 010001 (this hex value equals to decimal 65537)
* M1 = I owe you $2000.
* M2 = I owe you $3000.
* d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

Then converted messages into hexadecimal using Python:



So, the hex-encoded values of M1 and M2 are:

M1 = 49206f776520796f752024323030302e

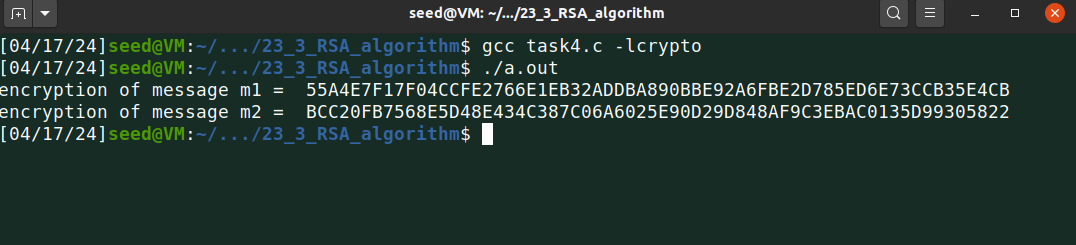
M2 = 49206f776520796f752024333030302e

**Code:**

| /\* Task4: Signing a Message \*/ #include <stdio.h> #include <openssl/bn.h>  #define NBITS 256  void printBNhex(char \*msg, BIGNUM \* a) {  /\* Use BN\_bn2hex(a) for hex string\*/  char \* number\_str = BN\_bn2hex(a);  printf("%s %s\n", msg, number\_str);  OPENSSL\_free(number\_str); }  int main () {  //Declare variables  BN\_CTX \*ctx = BN\_CTX\_new();  BIGNUM \*e = BN\_new();  BIGNUM \*n = BN\_new();  BIGNUM \*M1 = BN\_new();  BIGNUM \*M2 = BN\_new();  BIGNUM \*c = BN\_new();  BIGNUM \*d = BN\_new();   //Initialize variables with given values  BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");  BN\_hex2bn(&e, "010001");  BN\_hex2bn(&M1, "49206f776520796f752024323030302e");  BN\_hex2bn(&M2, "49206f776520796f752024333030302e");  BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");    //Encrypt -> c = M^d mod n   BN\_mod\_exp(c, M1, d, n, ctx);  printBNhex("encryption of message m1 = ", c);    //Encrypt -> c = M^d mod n   BN\_mod\_exp(c, M2, d, n, ctx);  printBNhex("encryption of message m2 = ", c);    return 0; } |
| --- |

I compiled and ran the code.

It's clear that even a minor change, such as altering just one digit in the messages, results in vastly different signatures. This illustrates how sensitive the signature generation process is to any variations in the input, where even the smallest change can lead to significantly different outcomes.



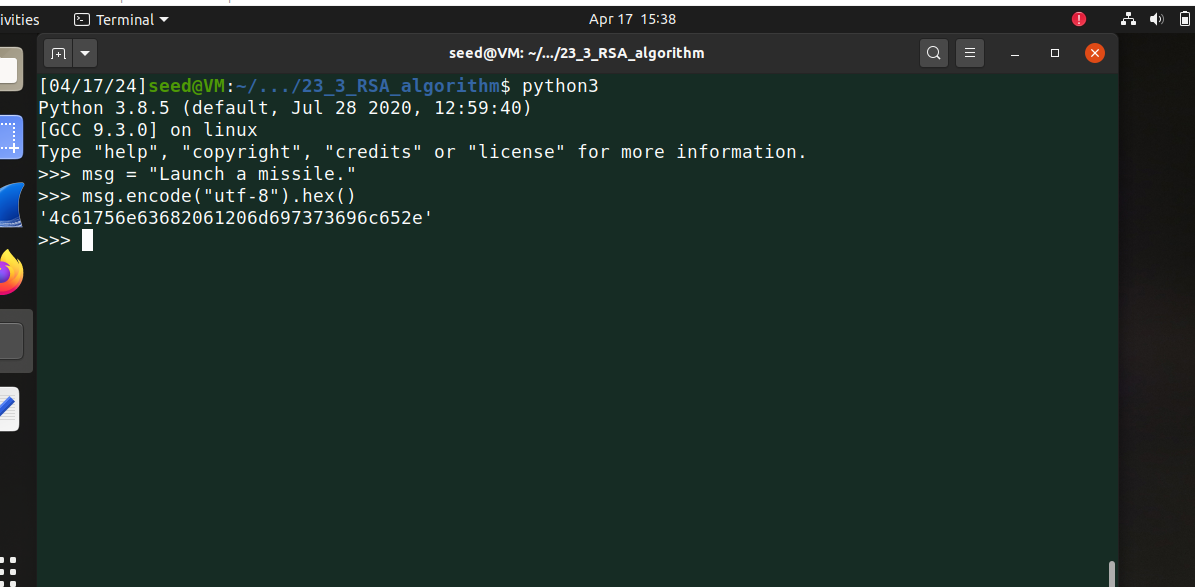
**Task 5: Verifying a Signature**

In this task, they provided the hexadecimal values of the modulus n, exponent e, a signature S1, and a message M1. My goal is to verify whether the provided signature correctly corresponds to the given message. Additionally, I am tasked with altering the message by one bit and then describing the impact of this modification on the verification process.

The given values of M, n, e, signature S1, along with the modified signature S2, are:

* M = Launch a missile.
* S1 = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
* S2 = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F
* e = 010001 (this hex value equals to decimal 65537)
* n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115

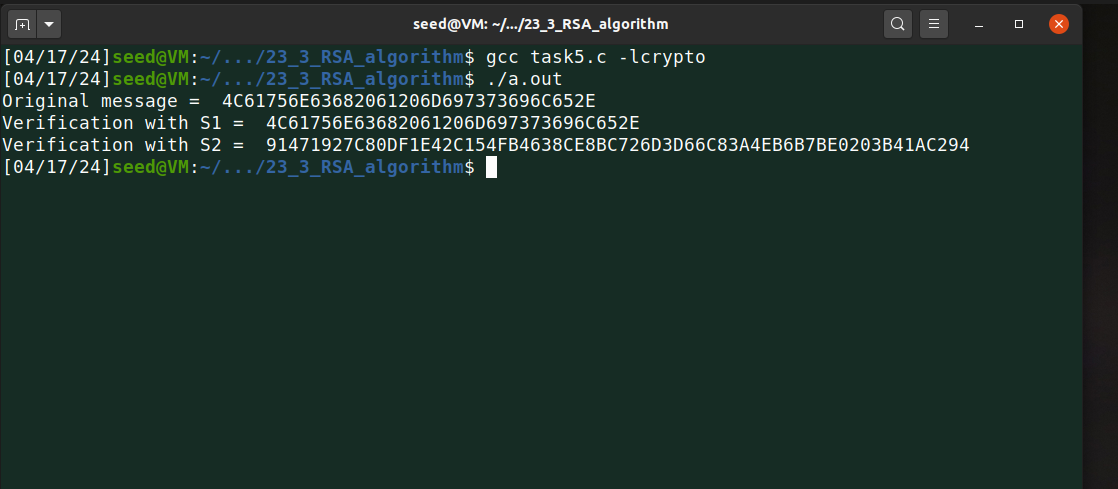
Now, I converted the message into hexadecimal using Python:



**Code:**

| /\* bn\_sample.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 \*e = BN\_new();  BIGNUM \*M = BN\_new();  BIGNUM \*S1 = BN\_new();  BIGNUM \*S2 = BN\_new();  BIGNUM \*res1 = BN\_new();  BIGNUM \*res2 = BN\_new();    // Initialize e, n, M, S1, S2  BN\_hex2bn(&e, "010001");  BN\_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");  BN\_hex2bn(&S1, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");  BN\_hex2bn(&S2, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");  BN\_hex2bn(&M, "4c61756e63682061206d697373696c652e");    // Verifying both the given signed results, S1 and S2 to see which one is right  printBN("Original message = ", M);    BN\_mod\_exp(res1, S1, e, n, ctx);  printBN("Verification with S1 = ", res1);   BN\_mod\_exp(res2, S2, e, n, ctx);  printBN("Verification with S2 = ", res2);    return 0; } |
| --- |

I ran the code.



Therefore, S1 is confirmed as the correct signature because it successfully reproduces the original message value. Additionally, modifying the signature by even a single bit results in a completely different generated message, demonstrating the sensitivity of the signature to any alterations.

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

In this task, I have to download a real X.509 certificate from paypal.com and get its issuer’s

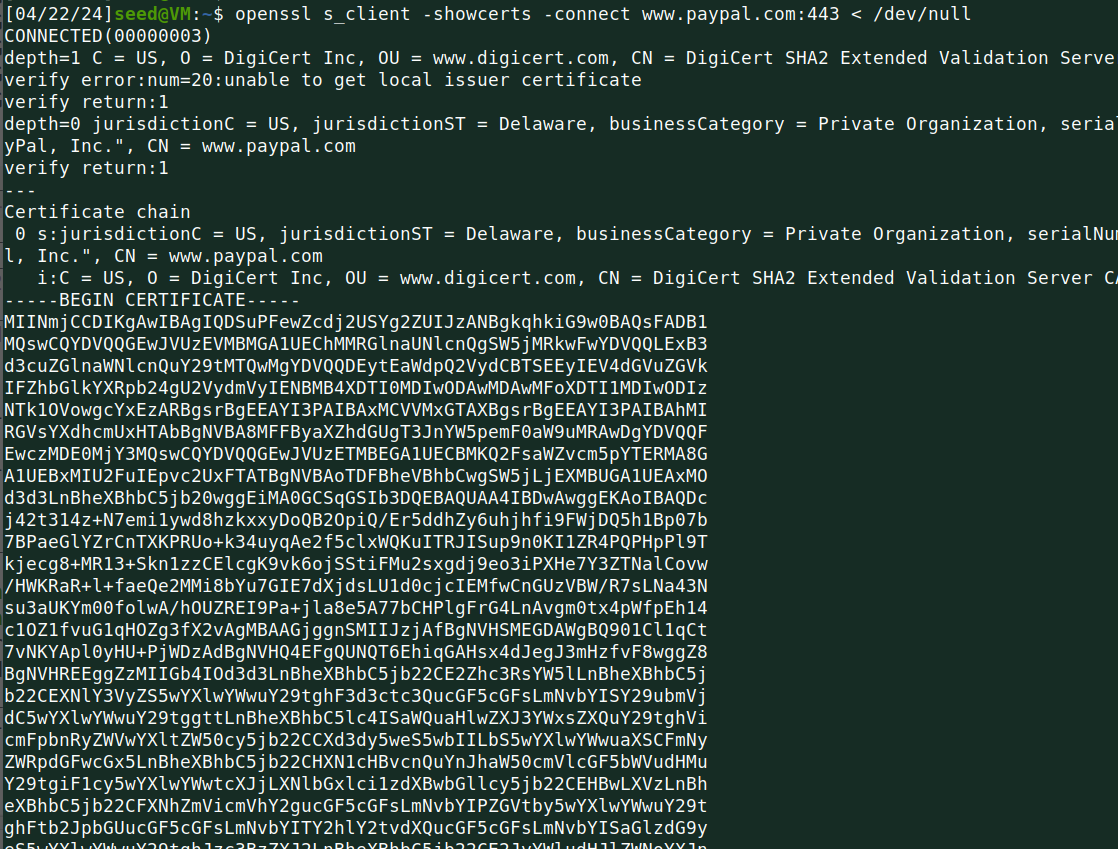
public key and then use the public key to manually verify the signature on the certificate using

the program.

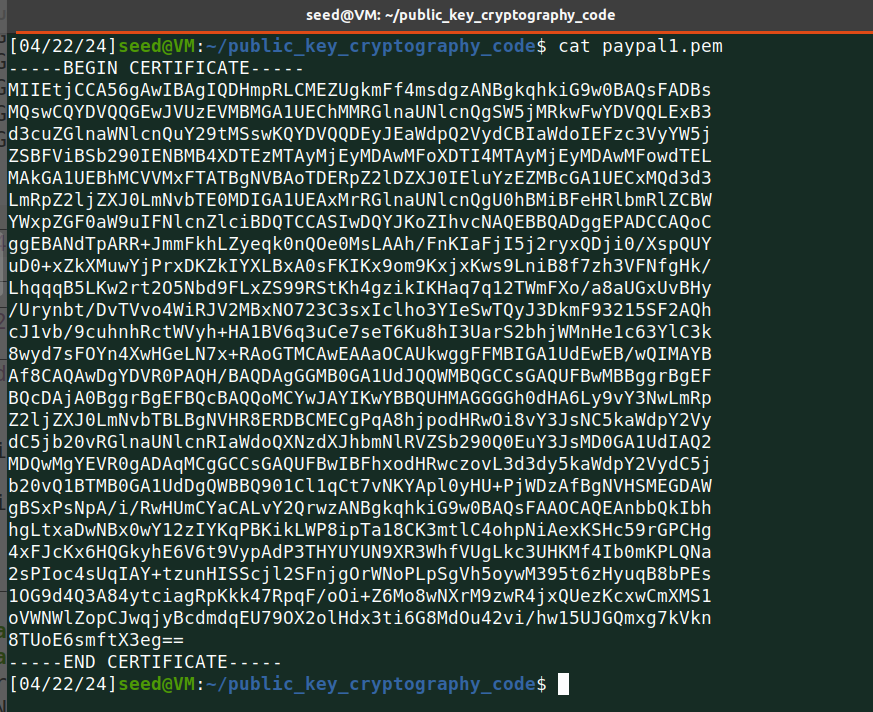
**Task 6.1: Download a certificate from a real web server’s**

In this sub- task, I have the command to download the X.509 certificate from the PayPal web

server.

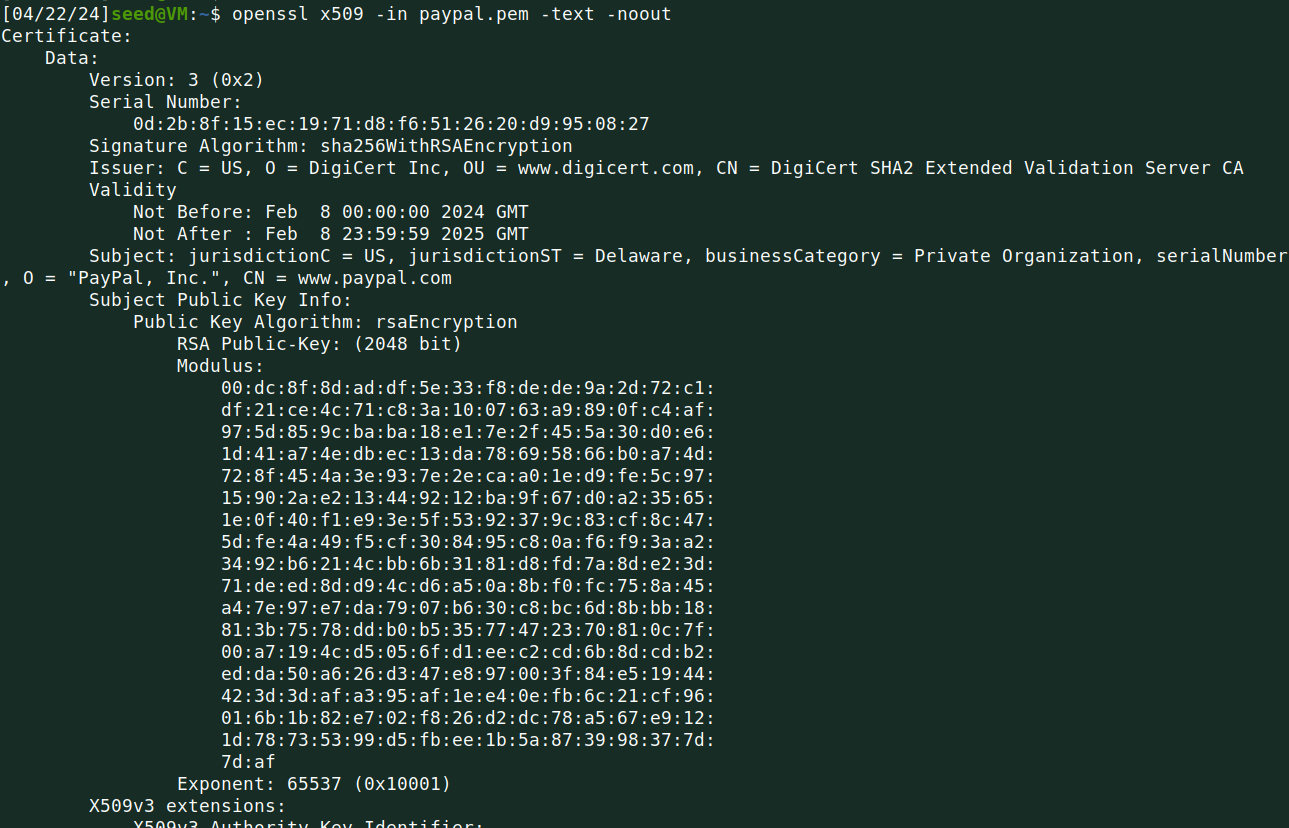


As illustrated above, I began by downloading the certificate from PayPal. Next, I downloaded and copied two certificates: one from PayPal (the organization itself) and another that is a CA-signed certificate. These are then stored in two separate files named paypal.pem and paypal1.pem.



**Task 6.2: Extract the public key (e, n) from the issuer’s certificate**

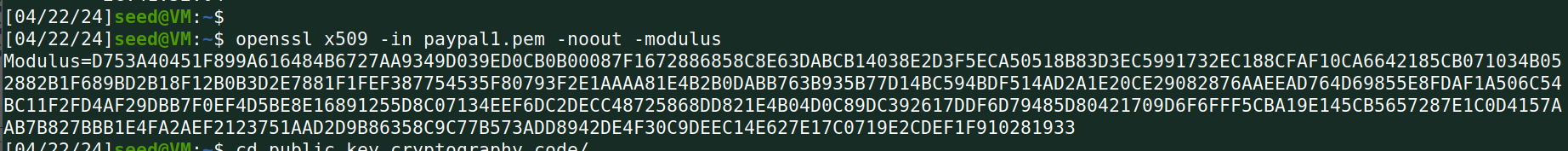
In this subtask, My aim is to extract the public key components (e, n) from the issuer's certificate. OpenSSL offers commands that allow for the extraction of specific attributes from X.509 certificates. Here, I will use the modulus to extract the value of n.



The hexadecimal value of the exponent, e is 65537 as you can see from the above

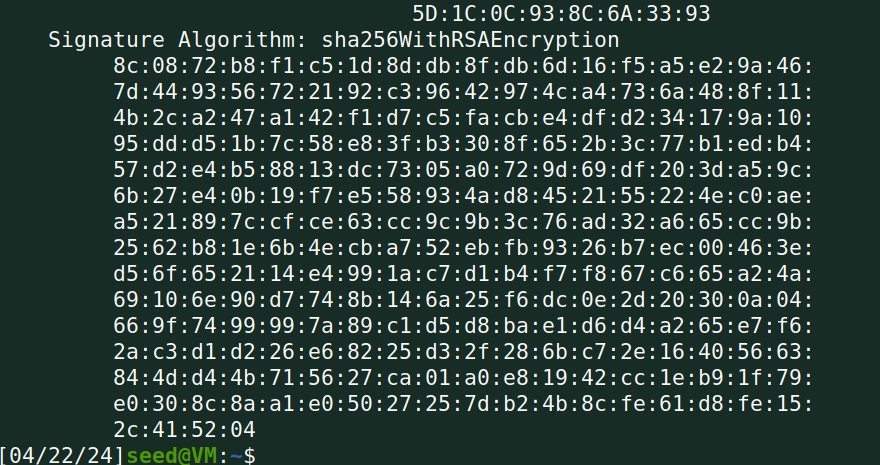
screenshots

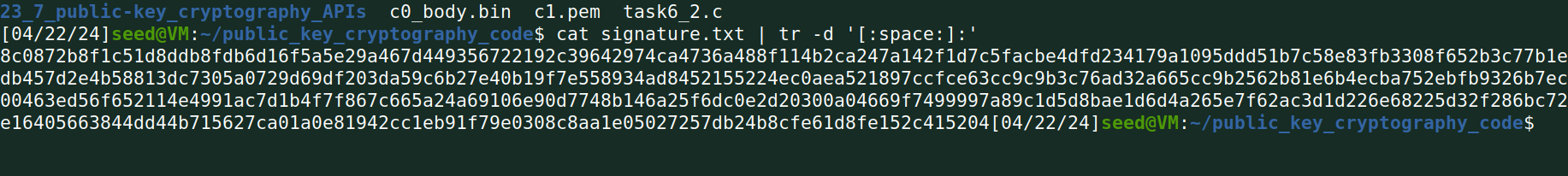
And below is the modulus.



**Task 6.3 – Extract the signature from the server’s certificate**

In the task, I will extract the signature from the server’s certificate

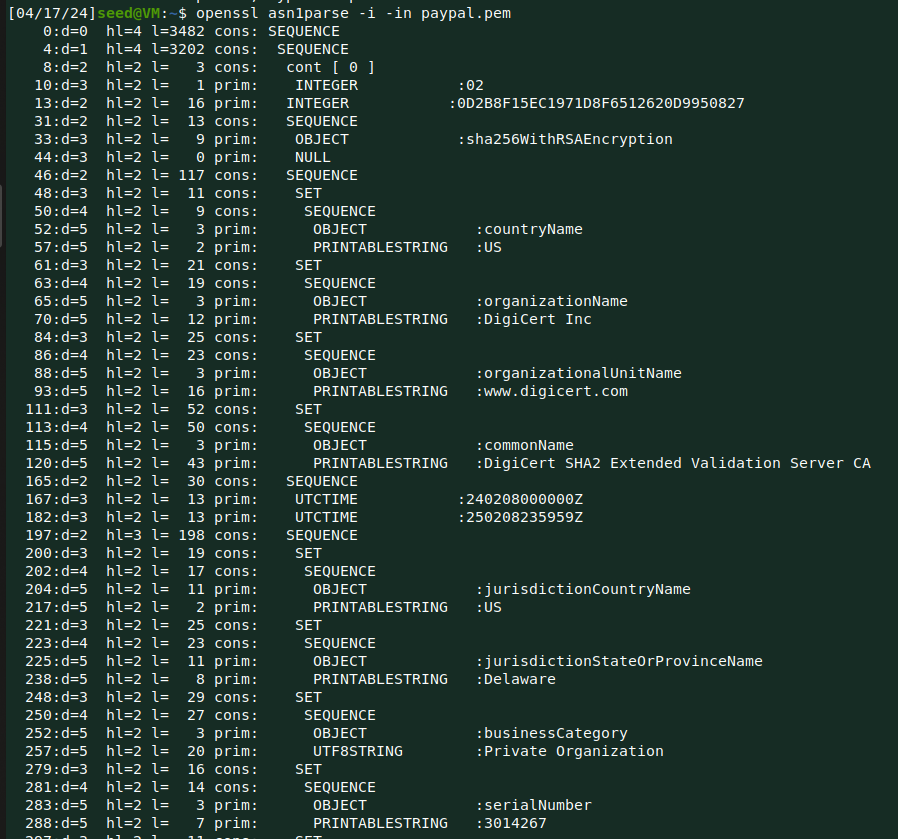


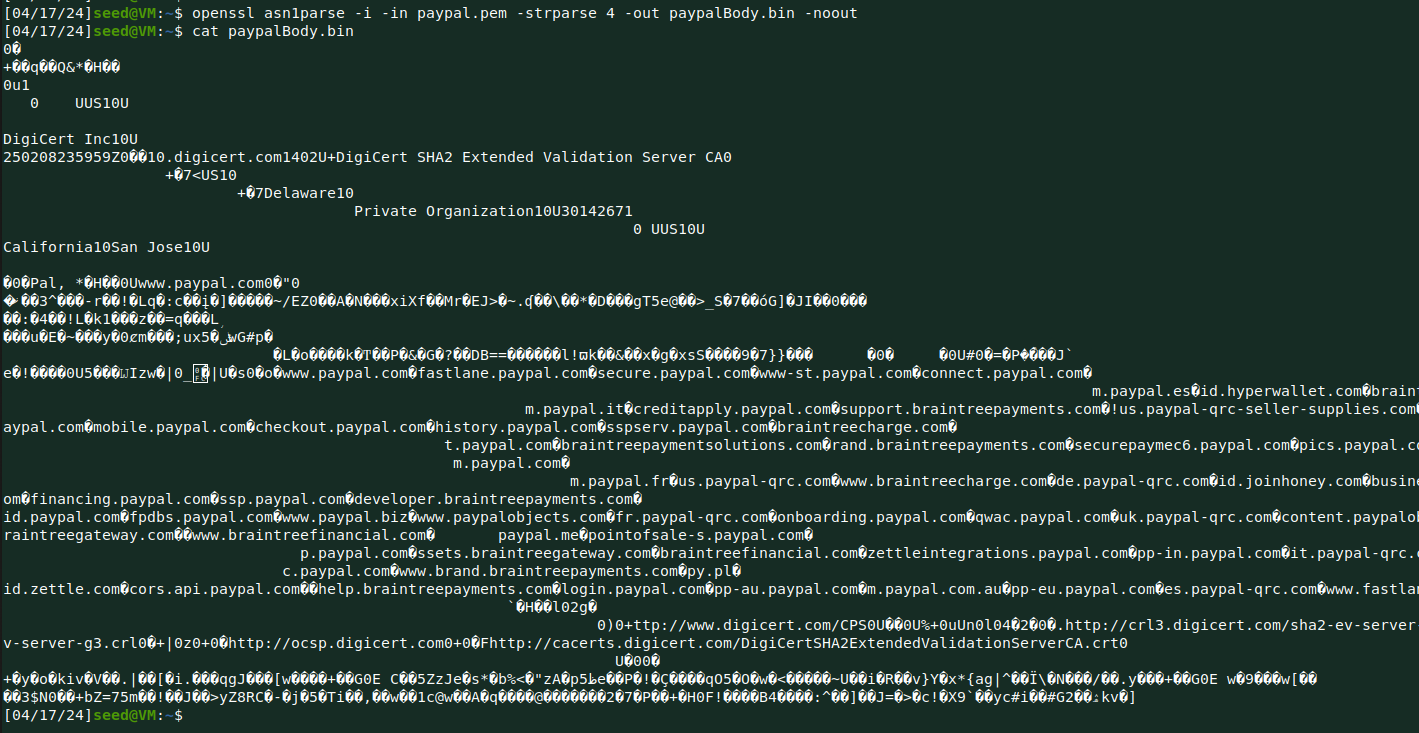


As shown in the screenshot above, we removed the extra spaces and colons from the file.

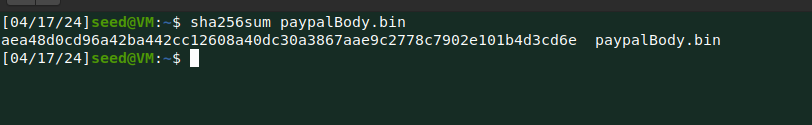
**Task 6.4 – Extract the body of the server’s certificate**

In this subtask, I have to extract the body of the server’s certificate. Additionally, to verify the signature, I must generate a hash from the certificate, which is done before the signature computation. OpenSSL provides a command called asn1parse, which is useful for extracting data from ASN.1 formatted data and can parse an X.509 certificate.





Now using the body of the certificate I calculated the hash of the body..



**Task 6.5: Verify the signature**

In this subtask, I will use the calculated values of the modulus (m), exponent (e), the CA signature (S), and the message (M) to verify the validity of the signature.

Below are the values.

n

e = 010001

s = 

m = aea48d0cd96a42ba442cc12608a40dc30a3867aae9c2778c7902e101b4d3cd6e

**Code:**

| /\* bn\_sample.c \*/ #include <string.h> #include <stdio.h> #include <openssl/bn.h> #define NBITS 256 #define SHA\_LENGTH 64  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 \*S = BN\_new();  BIGNUM \*res = BN\_new();    // Initialize the variables  BN\_hex2bn(&nhex2bn(&e, "010001");  BN\_hex2bn(&S, "");  BN\_hex2bn(&M, "aea48d0cd96a42ba442cc12608a40dc30a3867aae9c2778c7902e101b4d3cd6e");   // Print the original message and signature  printBN("Original Message, i.e. body of the certificate is = ", M);  printBN("Original Signature = ", S);   // Decrypt the signature  BN\_mod\_exp(res, S, e, n, ctx);   char \*res\_decrypted = BN\_bn2hex(res);  char substr[100];  strncpy(substr, res\_decrypted + strlen(res\_decrypted) - SHA\_LENGTH, SHA\_LENGTH);  BN\_hex2bn(&res, substr);  printBN("Decrypted signature", res);   if (BN\_cmp(M, res) == 0)   printf("The signature has been verified, it is the right one. \n");  else   printf("Wrong signature\n");  return 0; } |
| --- |

Now I ran the above code below is the result.

And I successfully verified the signature from the CA.

