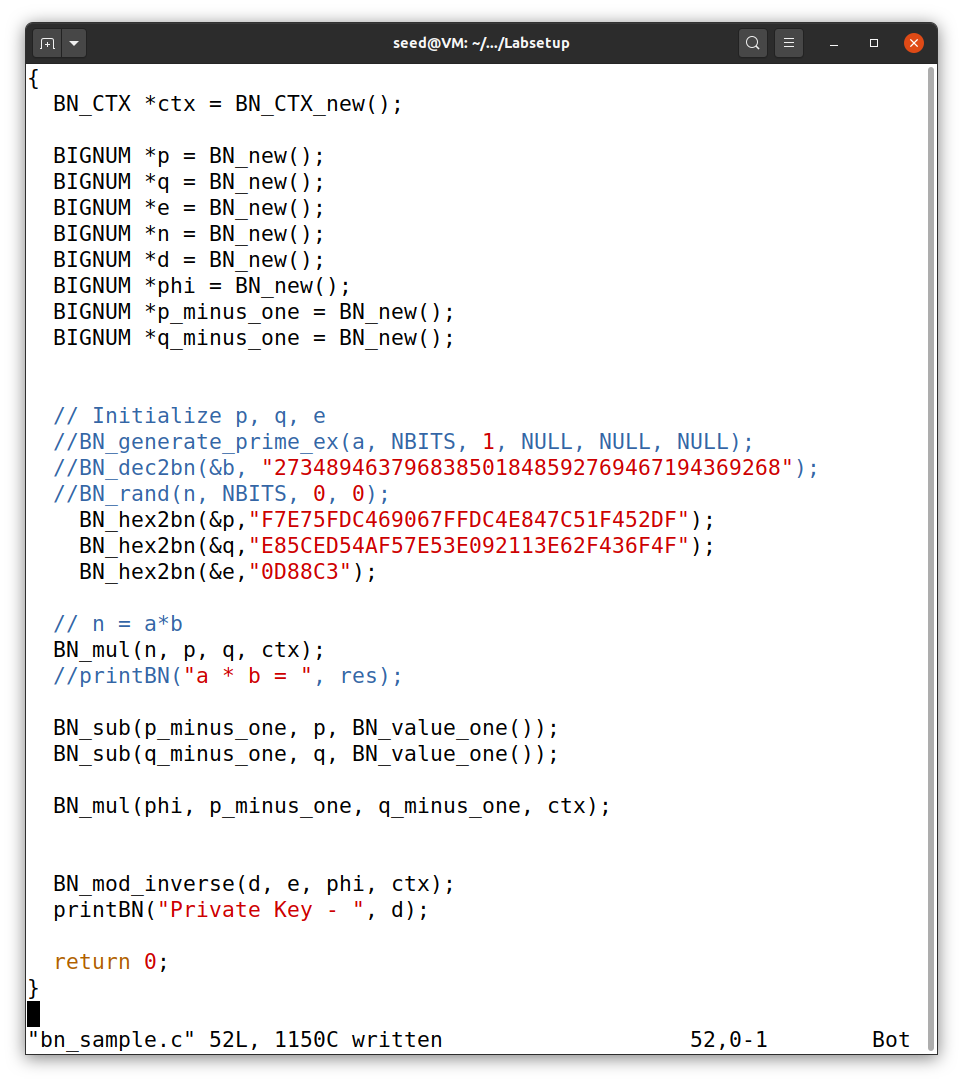
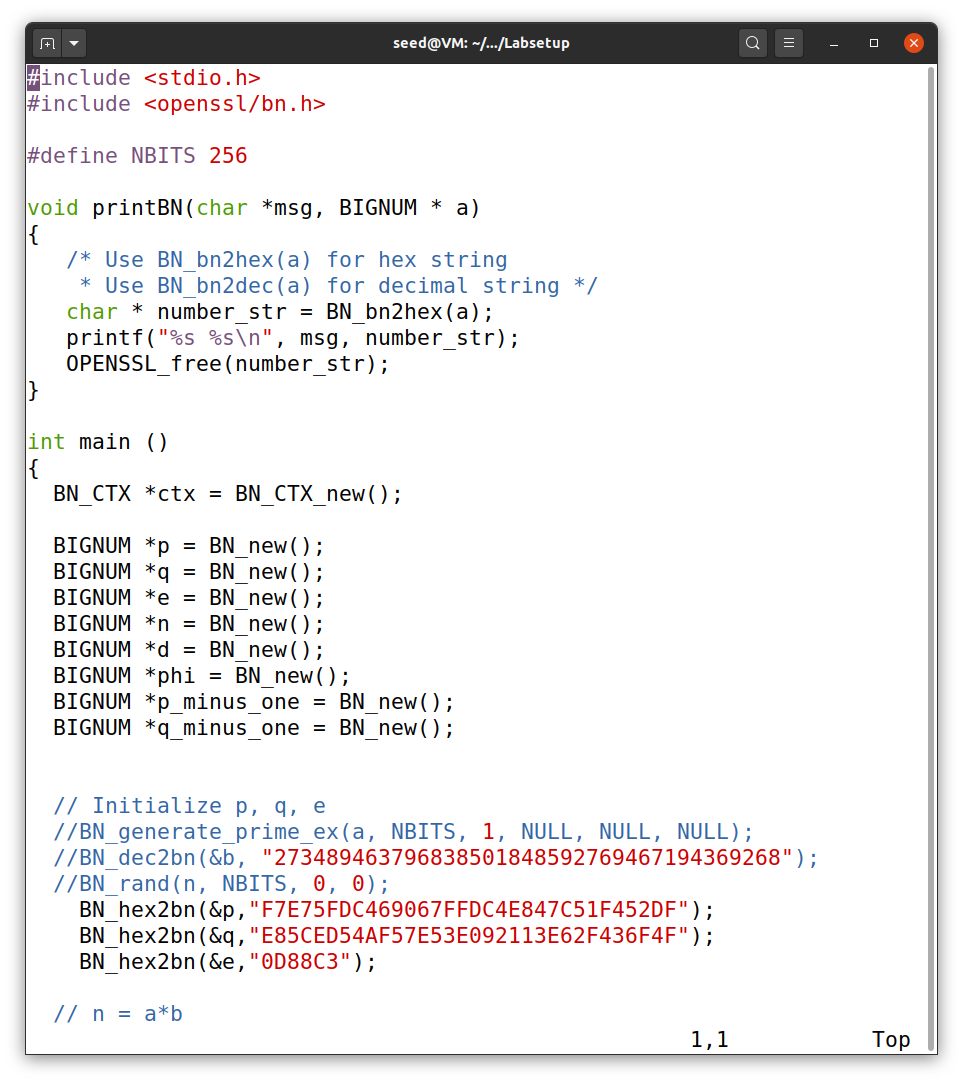
**Lab 10 RSA Public-Key Encryption and Signature Lab**

**Task 1: Deriving the Private Key**

****

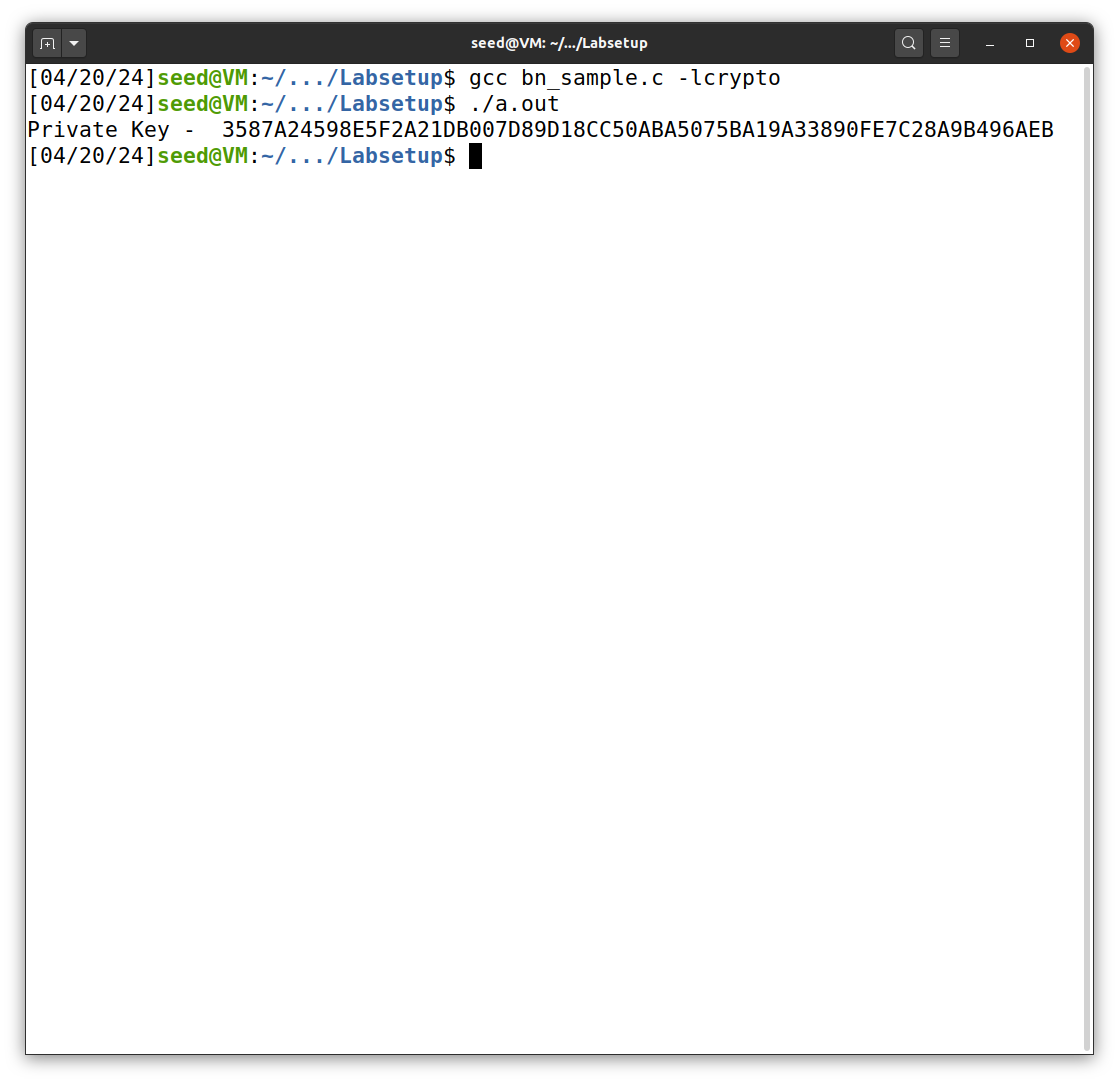
This is the program we will use for the BIGNUM APIs, which will calculate d, given values of p, e, and q. The value of modulus n, is calculated using the formula - n = p \*q

So in this following program the value of p, e, and q are -

p = F7E75FDC469067FFDC4E847C51F452DF

*e =* 0D88C3

q = E85CED54AF57E53E092113E62F436F4F



Then we first compile the program using gcc compiler and then run it, we get the value of private key.

- 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

**Task 2: Encrypting a Message**

In this task, using the given hexadecimal values of the modulus n, exponent e and private key d. we encrypt the given message by first converting it from ASCII to hex string, and then use those values to to encrypt the message.

We have the following values -

n = *DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5*

e = 010001

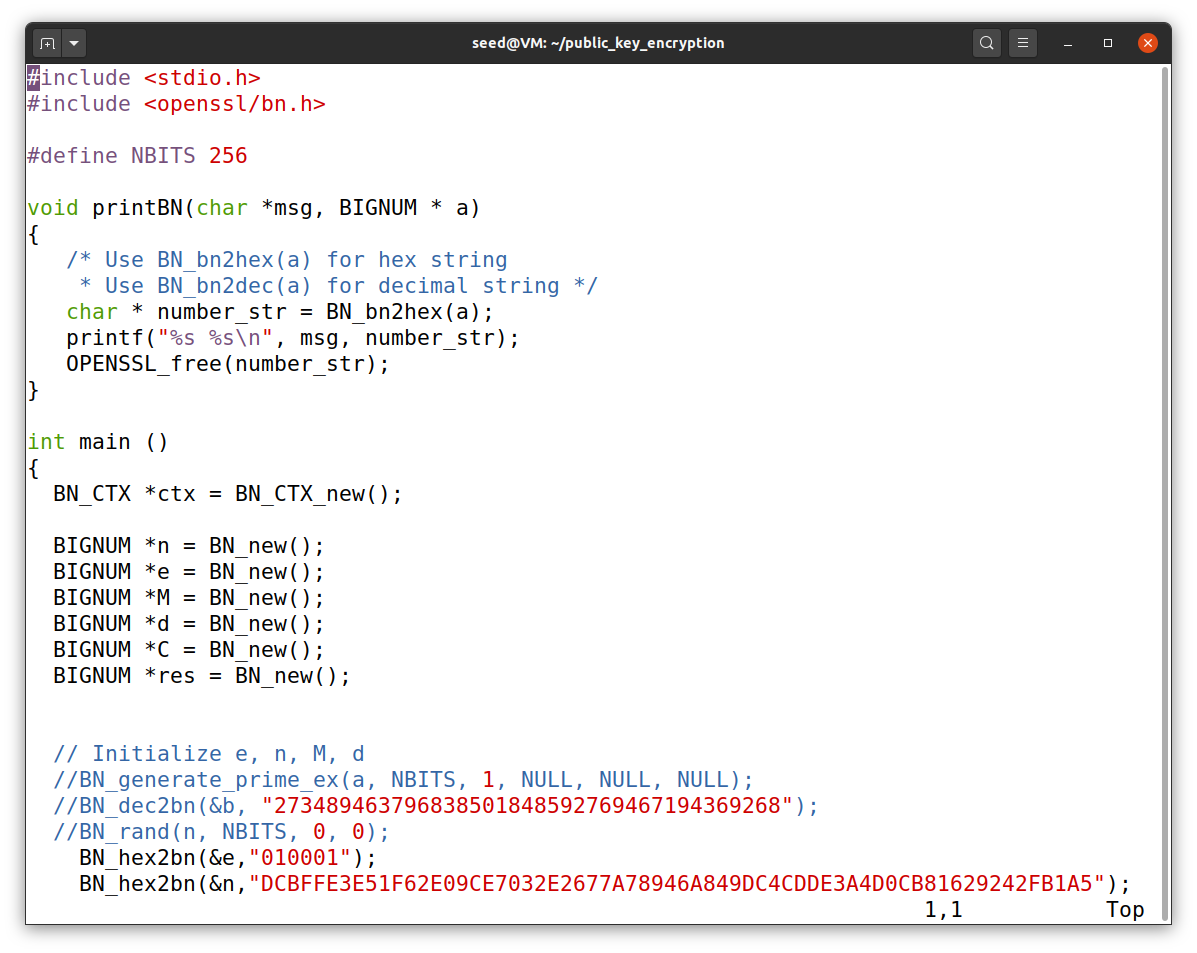
m = A top secret!

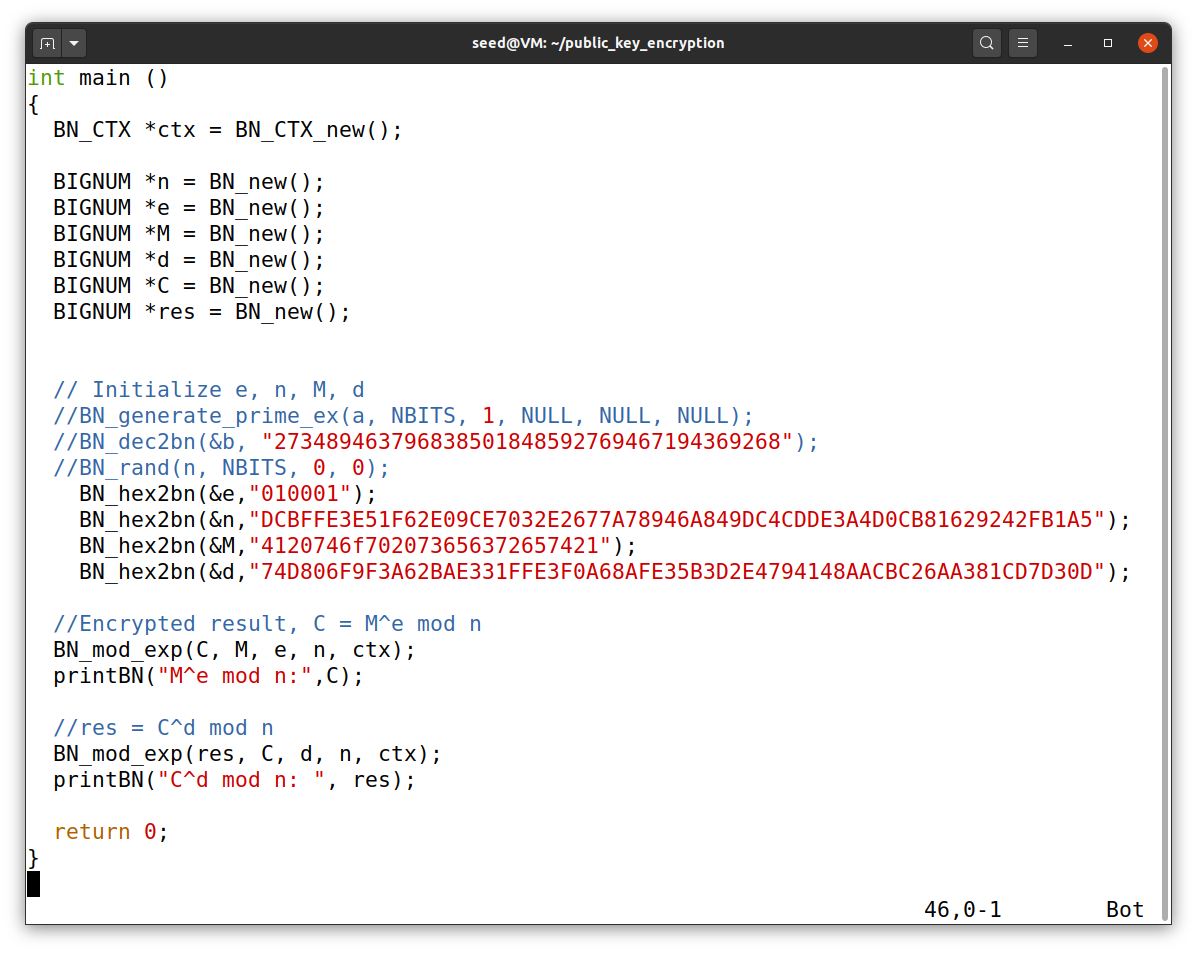
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D



Then we convert the ASCII string message to hex string using the above command.

We get the value of M - 4120746f702073656372657421





We use the following program to verify the private key value to the encrypted result by decrypting the encrypted result.

When we compile the program using the gcc compiler and run it, we get the the same value of n, thus encrypting was successful.

**Task 3 : Decrypting a Message**

This task is somewhat similar to previous task, we are given the modulus n, exponent e and ciphertext C. We have to decrypt the given cipher text using all the given values. And then we decrypt the hexadecimal string back to ASCII string to get the original message.

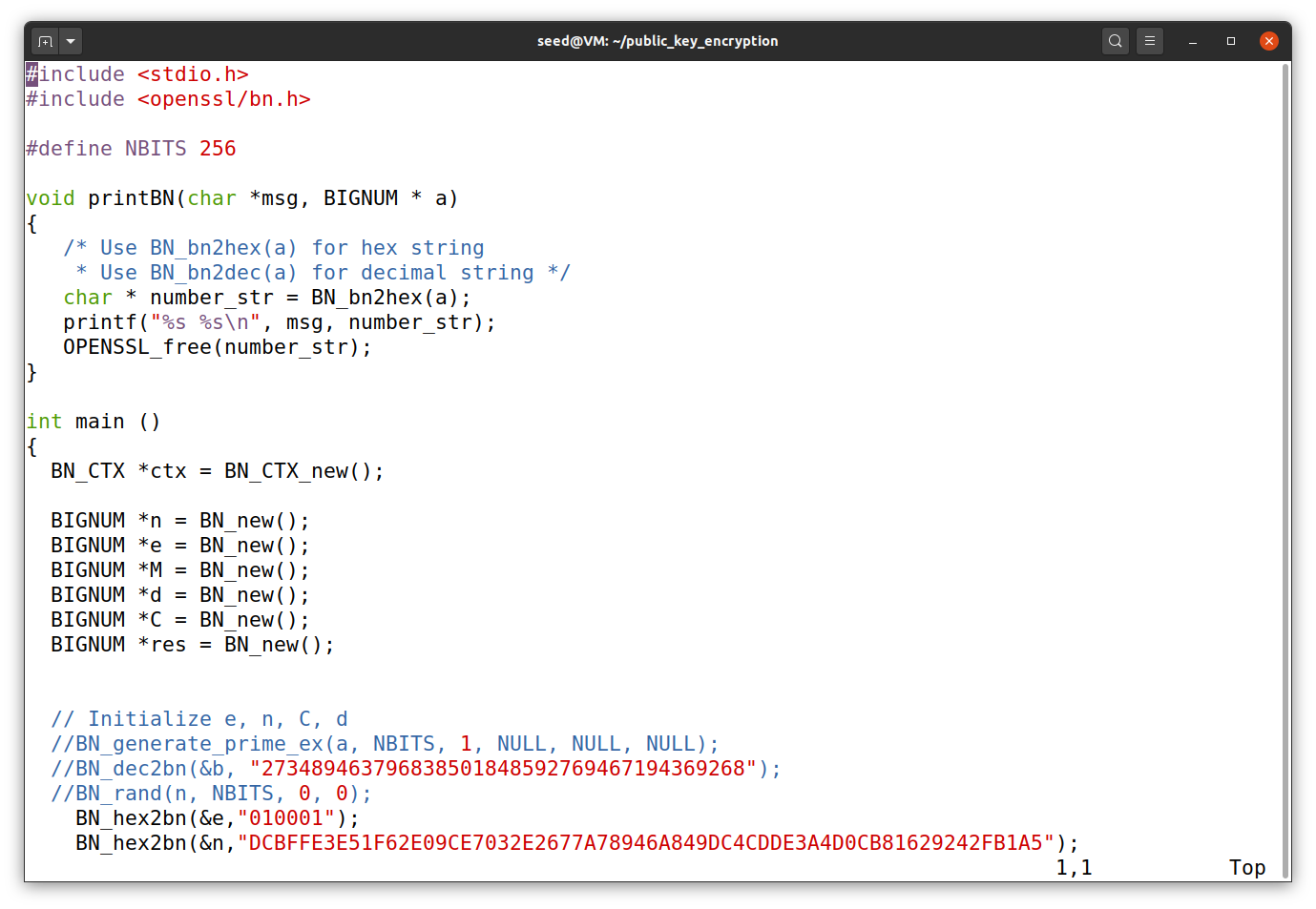
We have the following values -

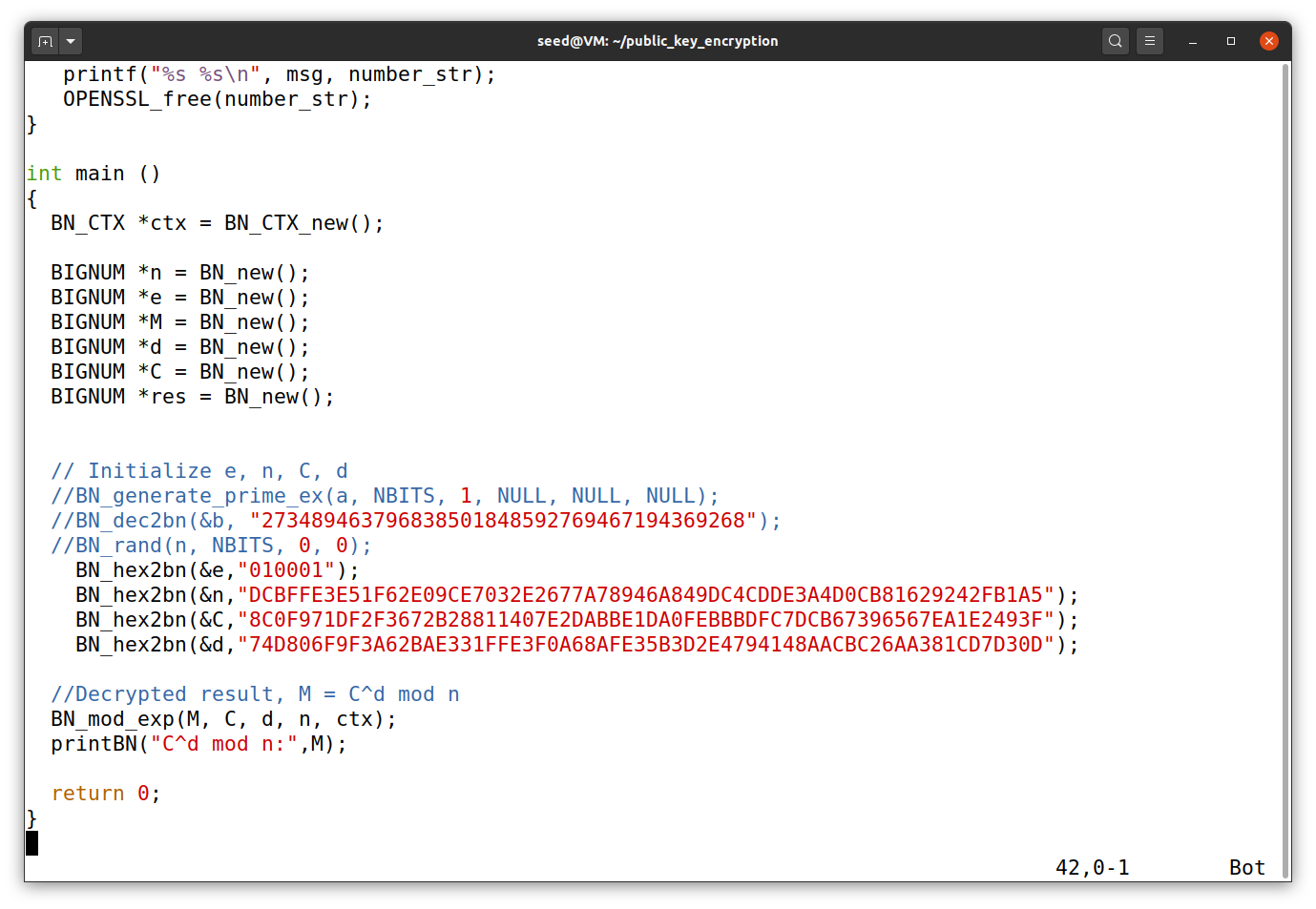
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

e = 010001

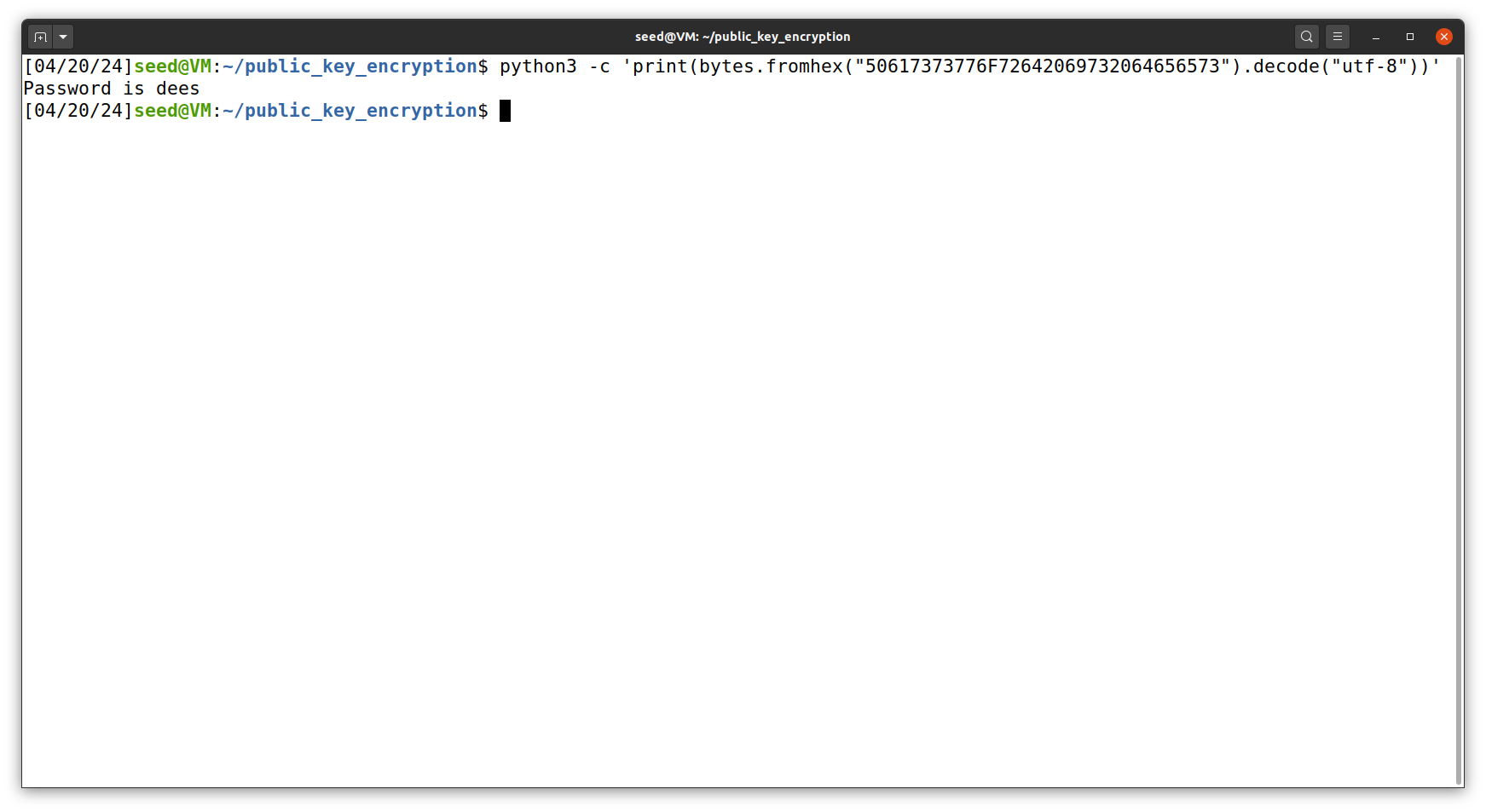
C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D



This is the program we will use to get the decrypted hexadecimal string.

So then we compile the code using the gcc compiler and then run it to get the hex string.

So when convert the hex string back to ASCII string using the following command we get the value of the cipher text as - “Password is dees”

**Task 4: Signing a Message**

In this task, we have the following values that is the modulus m, exponent e, private key d and message M1. We have to generate a signature for the given message by signing it directly instead of its hash value.

The given values of n, e, d original message M1, along with the modified message M2 are -

n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

*e =* 010001

M1 = I owe you $2000

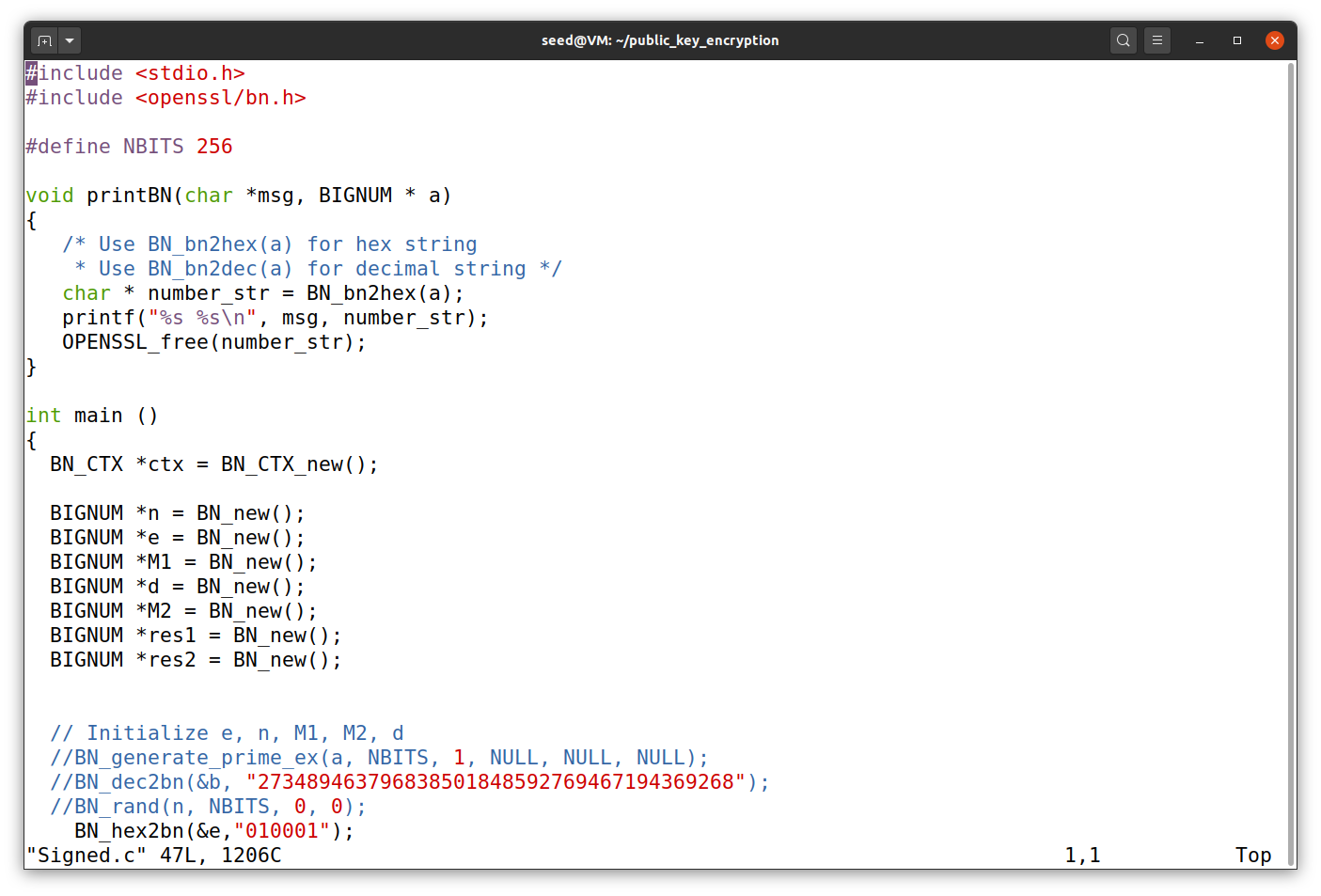
M2 = I owe you $3000

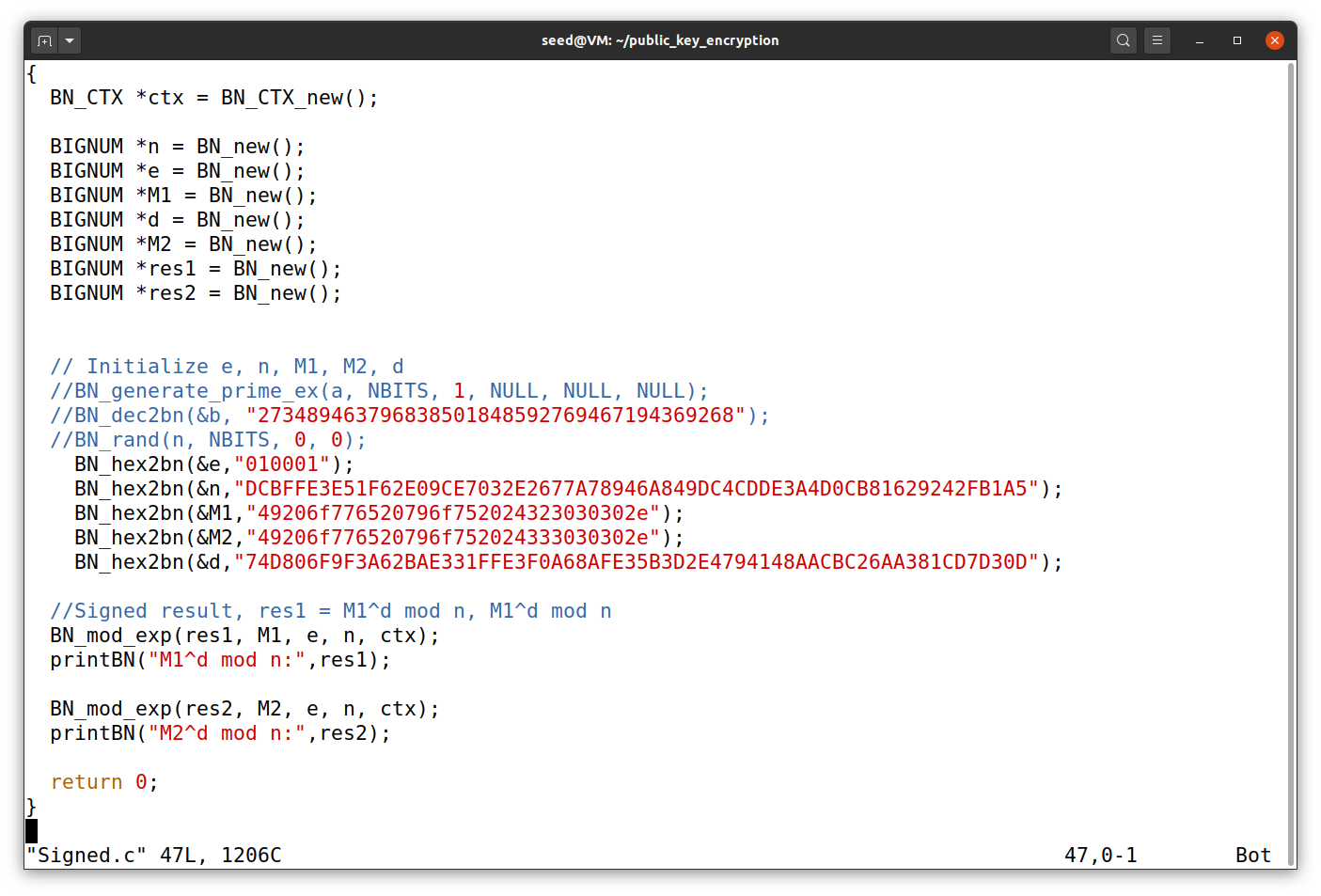
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

So from this hex encoded values of M1 and M2 are -

M1 = 49206f776520796f752024323030302e

M2 = 49206f776520796f752024323030302e





Now we compile the program using gcc compiler and then run the program

From the output we can see that by changing a single digit in message, it’s generated signature are very different.

**Task 5: Verifying a Signature**

In this task, we are given the following values modulus n, exponent e, signature S1, and message M1. We have to verify the generated signature is right one for the given message. We have to modify one bit and see what results we get

We have the following values of M, n, e, Signature S1, and modified Signature S2 -

S1 = *643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F*

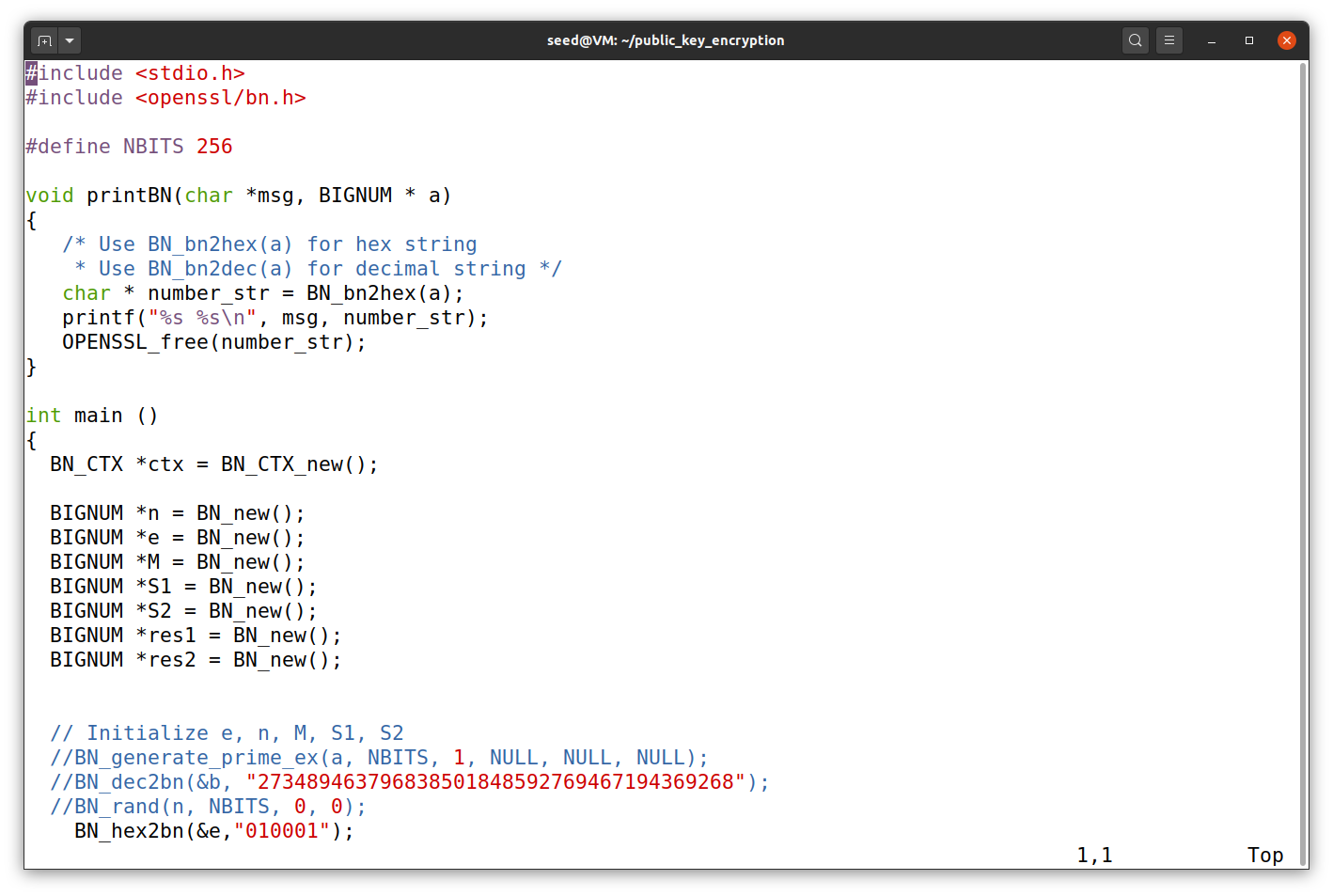
S2 = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F

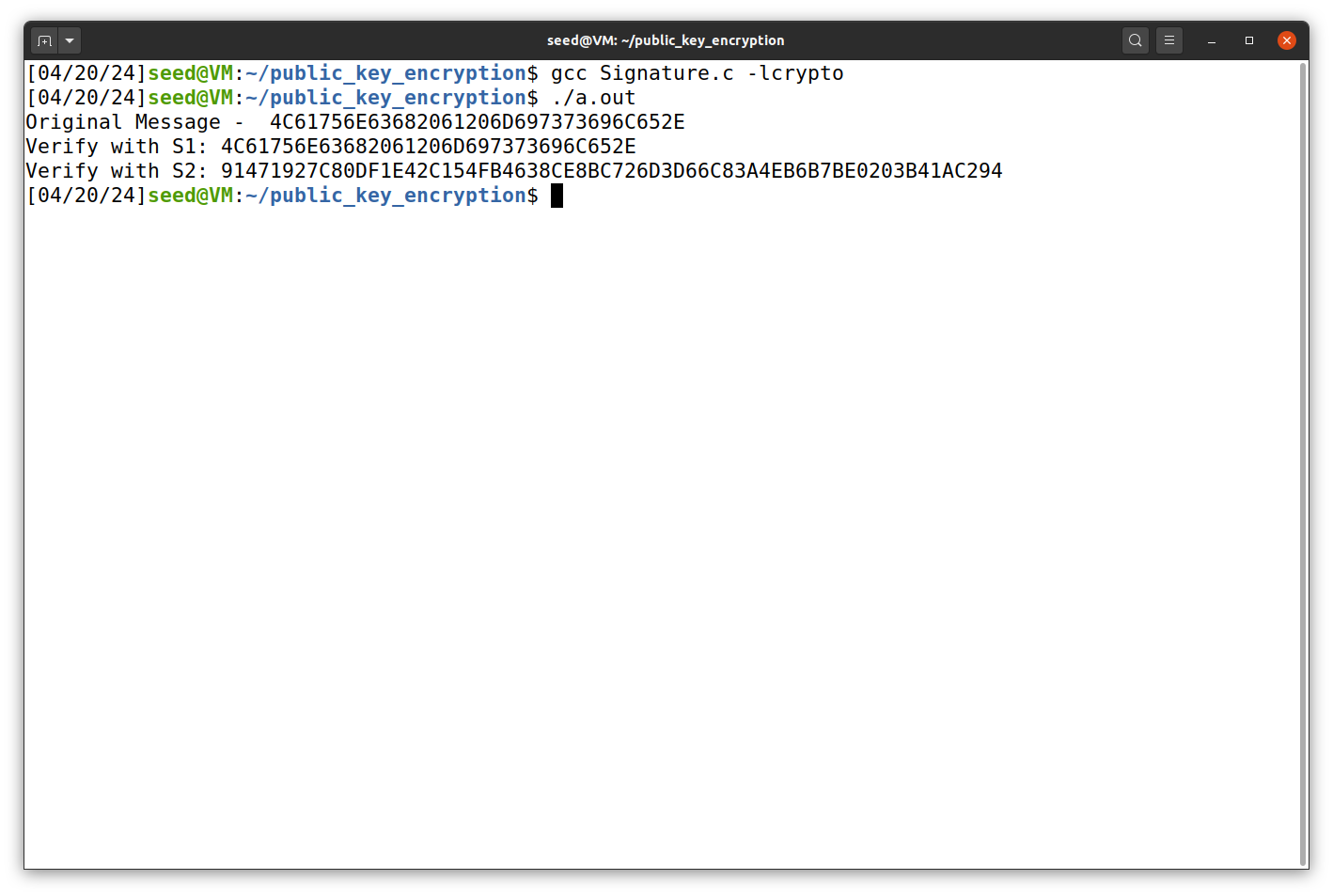
*e = 010001*

*n =* AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115



We convert the ASCII string to hex decimal string using the following command.



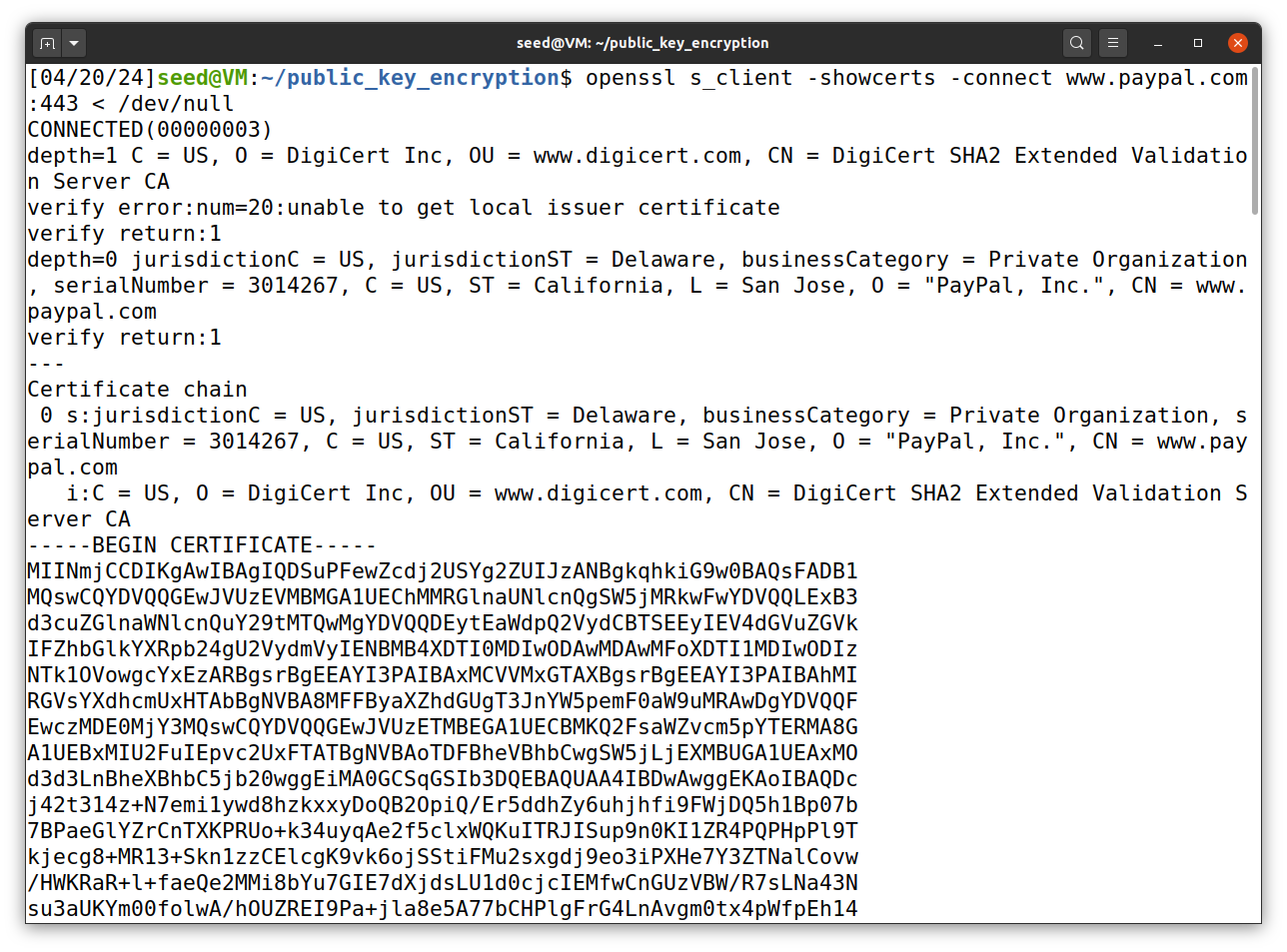
So when we compile the program using the gcc compiler and then run the code, we get the following output. As we can see from the output S1, is the correct signature because it gives the original message whereas S2 gives wrong output as we changed 1 last bit of it, so by changing one bit also can change complete message.

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

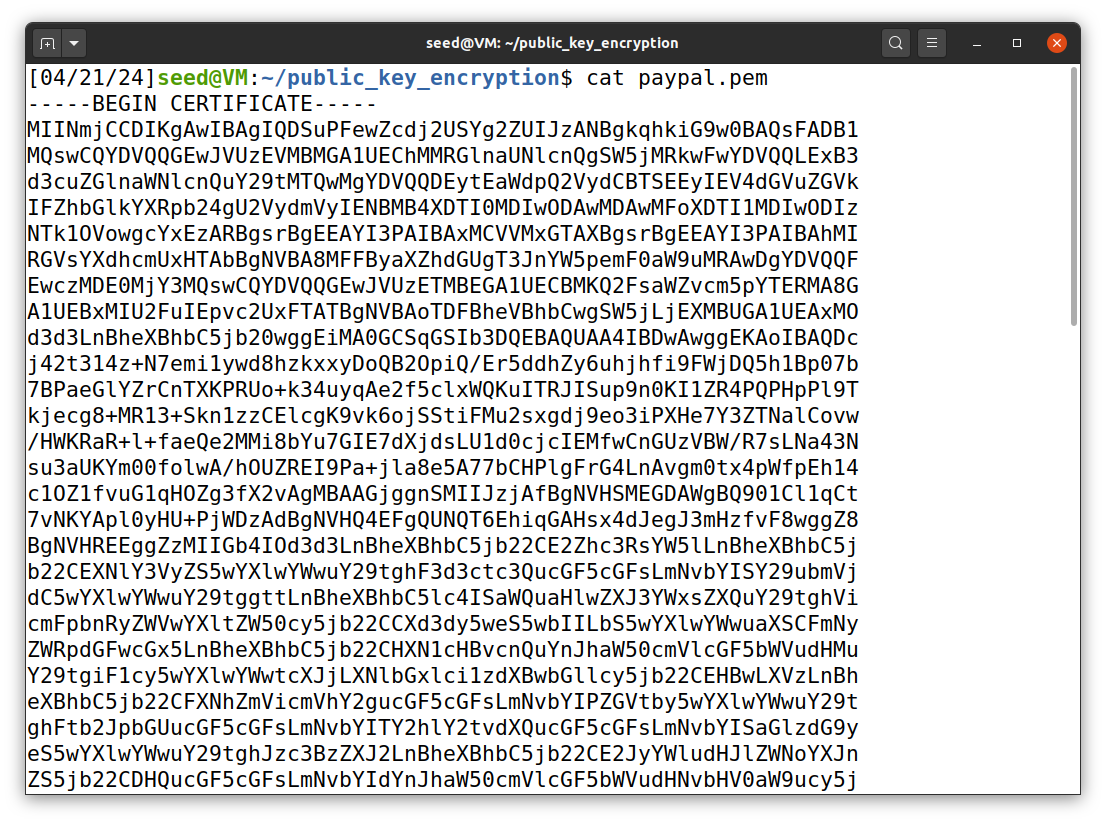
In this task, we have to download a real X.509 certificate from paypal.com and get its issuer’s public key and then use the public key to manually verify the signature on the certificate using the program.

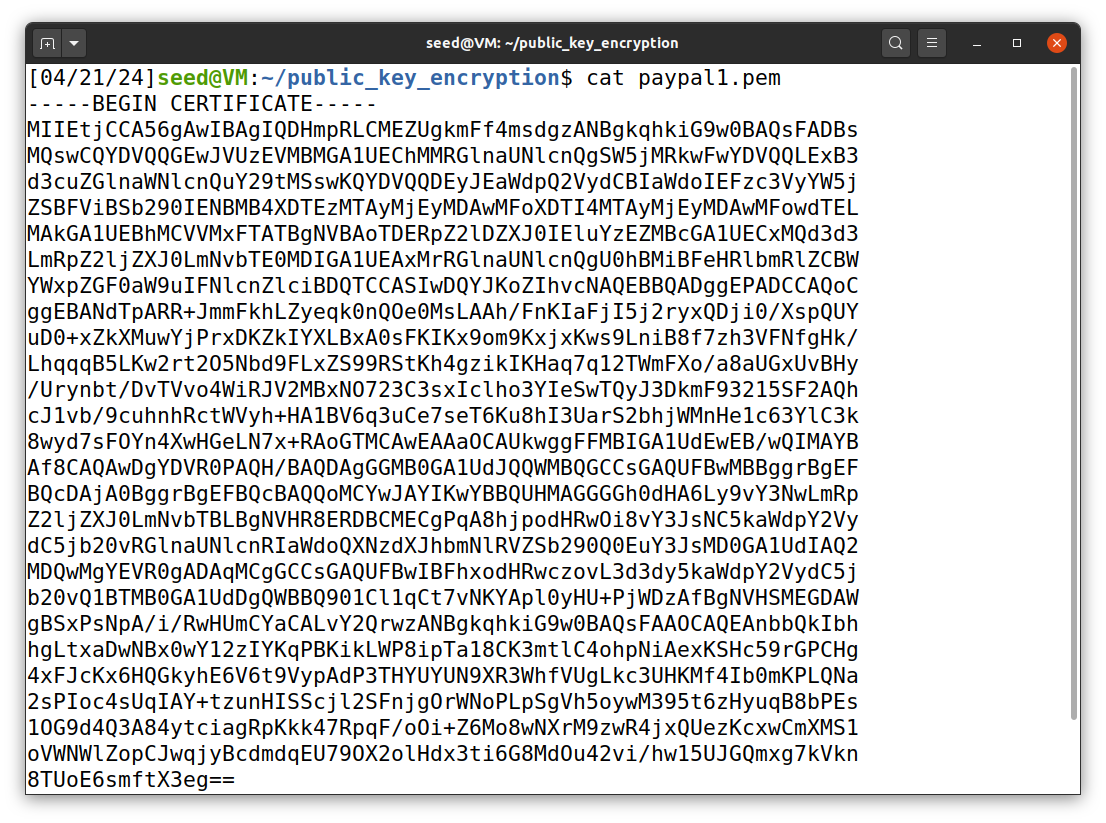
**Task 6.1: Download a certificate from a real web server’s**

In this sub- task we have the command to download the X.509 certificate from the paypal web server.



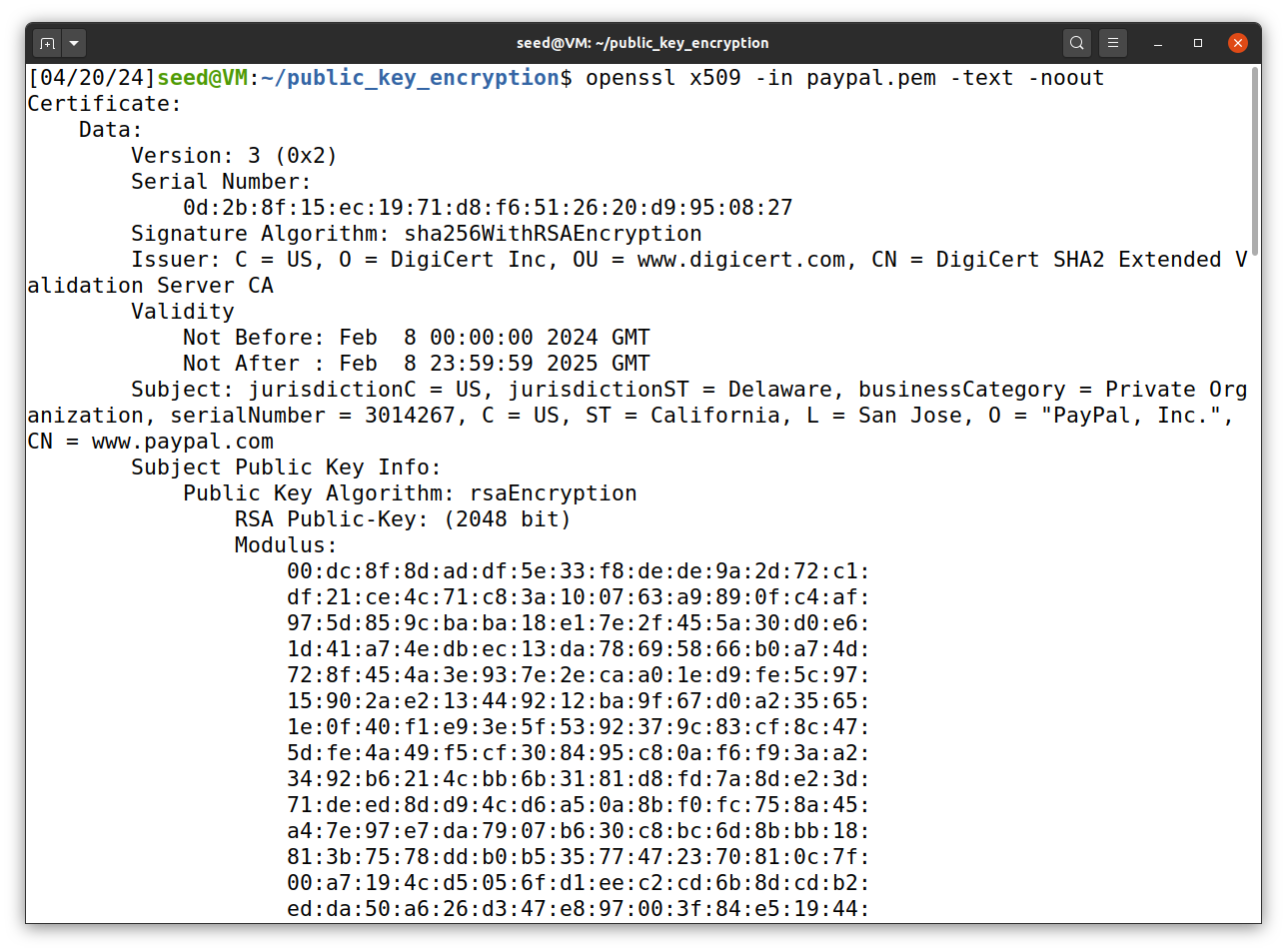
So as you can see from the above we download the certificate from the paypal and then download we copy the both certificate one is of the org paypal and the other is CA signed certificate and store them in two files paypal.pem and paypal1.pem

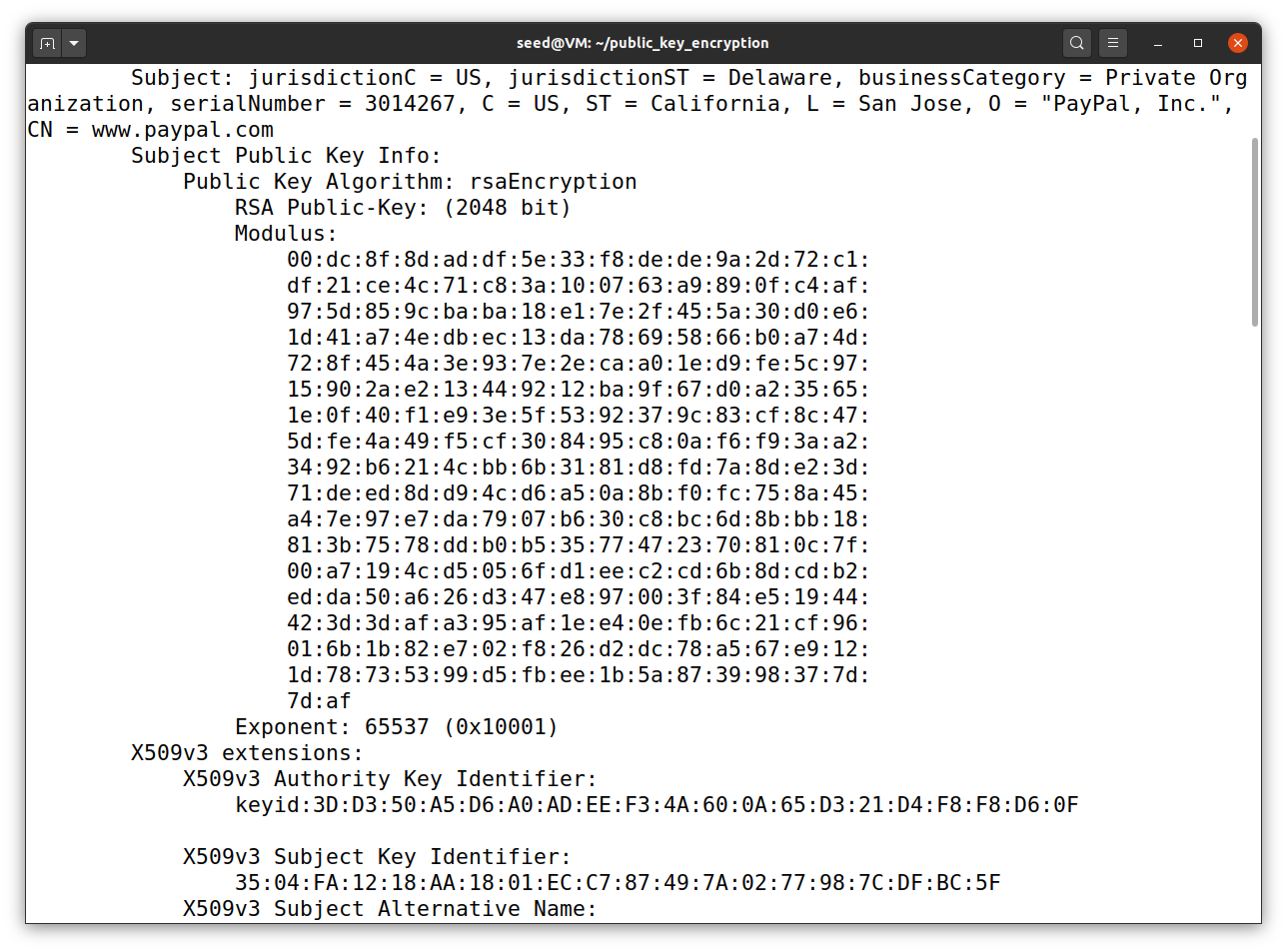




**Task 6.2: Extract the public key (e, n) from the issuer’s certificate -**

In this sub task we need to extract the public key (e, n) from issuer’s certificate. Openssl provides command to extract certain attributes from the x509 certificates. In this we extract the value of n using the modulus

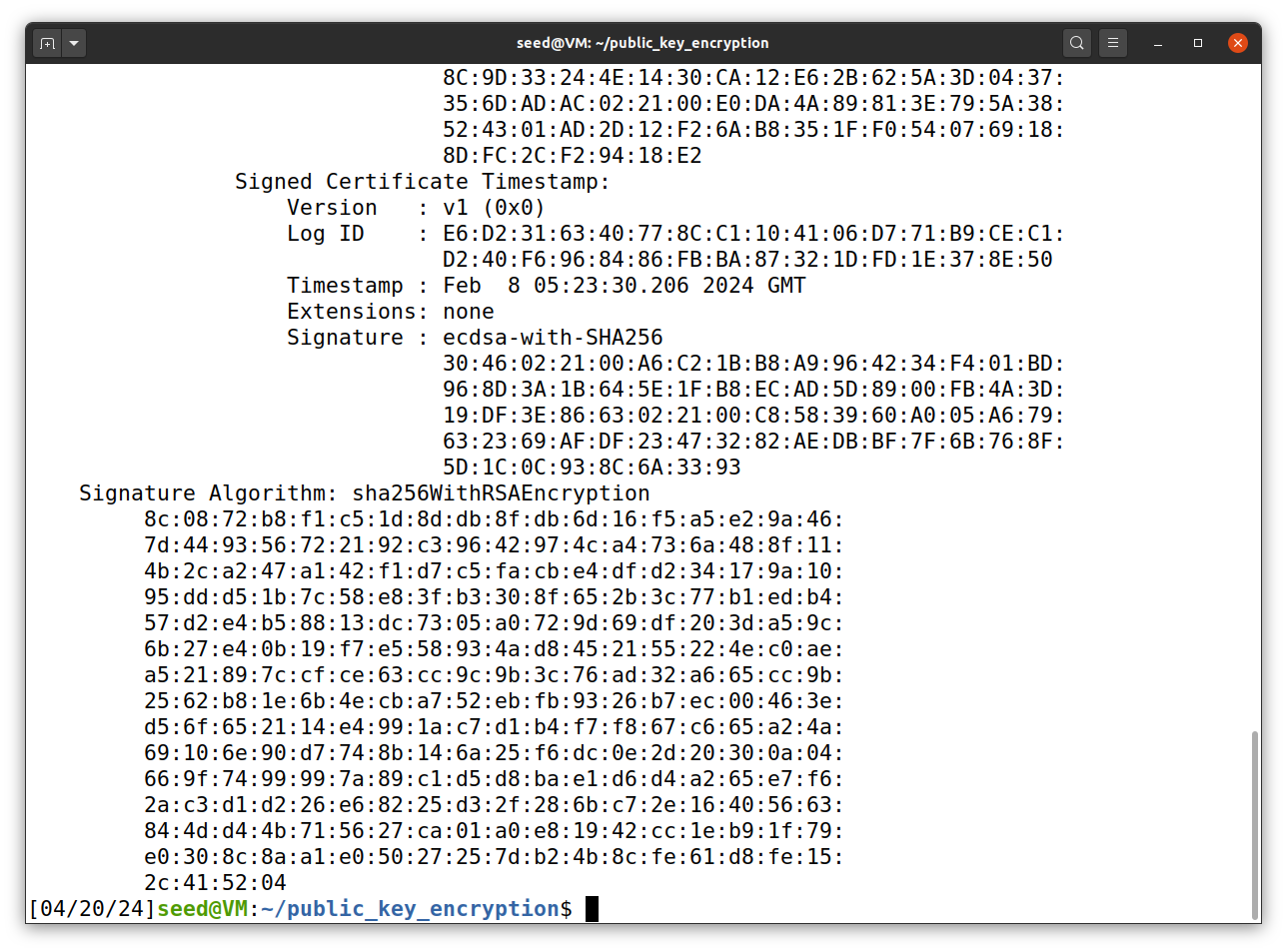


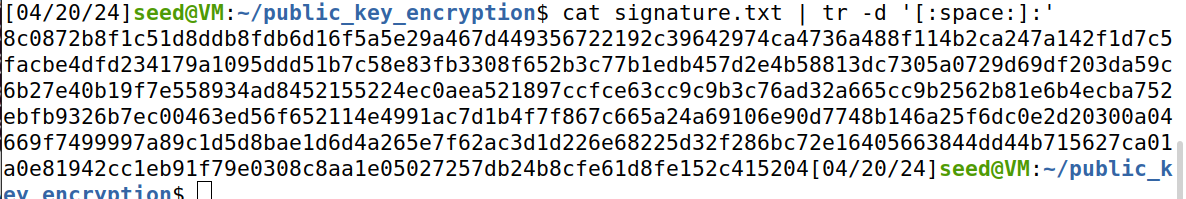


The hexadecimal value of the exponent , e is 65537 as you can see from the above screenshots

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

In sub task, we extract the signature from the server’s certificate



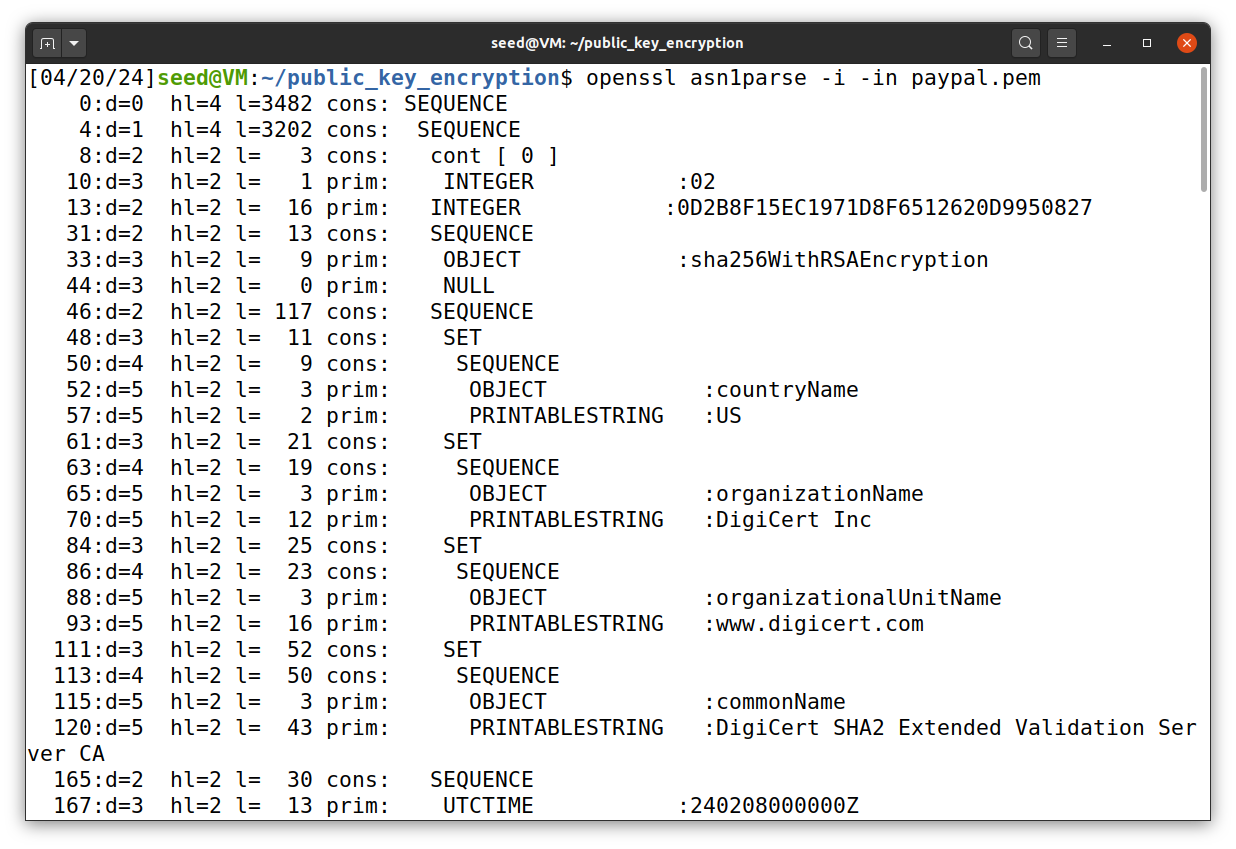
So as you can see from the above screenshot we removed the extra space and colons from the file.

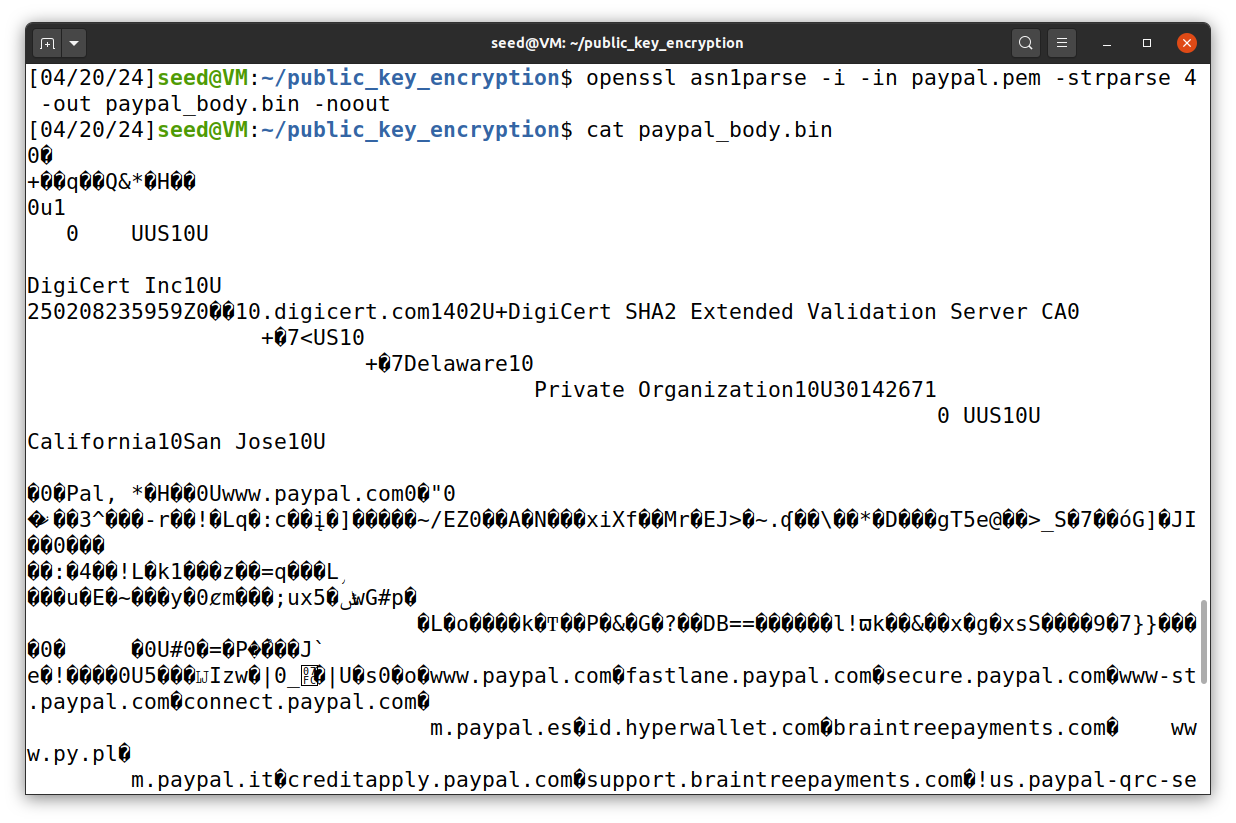
Thus the hexadecimal value of Signature S, is -   
  


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

In this sub task, we extract the body of the server’s certificate. To verify the signature, we also need to generate the hash from a certificate. Since the hash is generated before the signature is computed.

Openssl has a command called asn1parse used to extract data from ASN.1 formatted data, and is able to parse an X.509 certificate.

Now using the -strparse option to get field from offset 4, which gives the body of the certificate, excluding the signature block.



Now using the body of the certificate we can calculate the hash of the body as shown above.   
  
So from this we get the hash of the certificate as -

aea48d0cd96a42ba442cc12608a40dc30a3867aae9c2778c7902e101b4d3cd6e

**Task 6.5 : Verify the signature**

In this sub task using the calculated values of modulus m, exponent e, and CA signature S and message M. We have to verify whether the signature is valid or not.

So we have the following values -



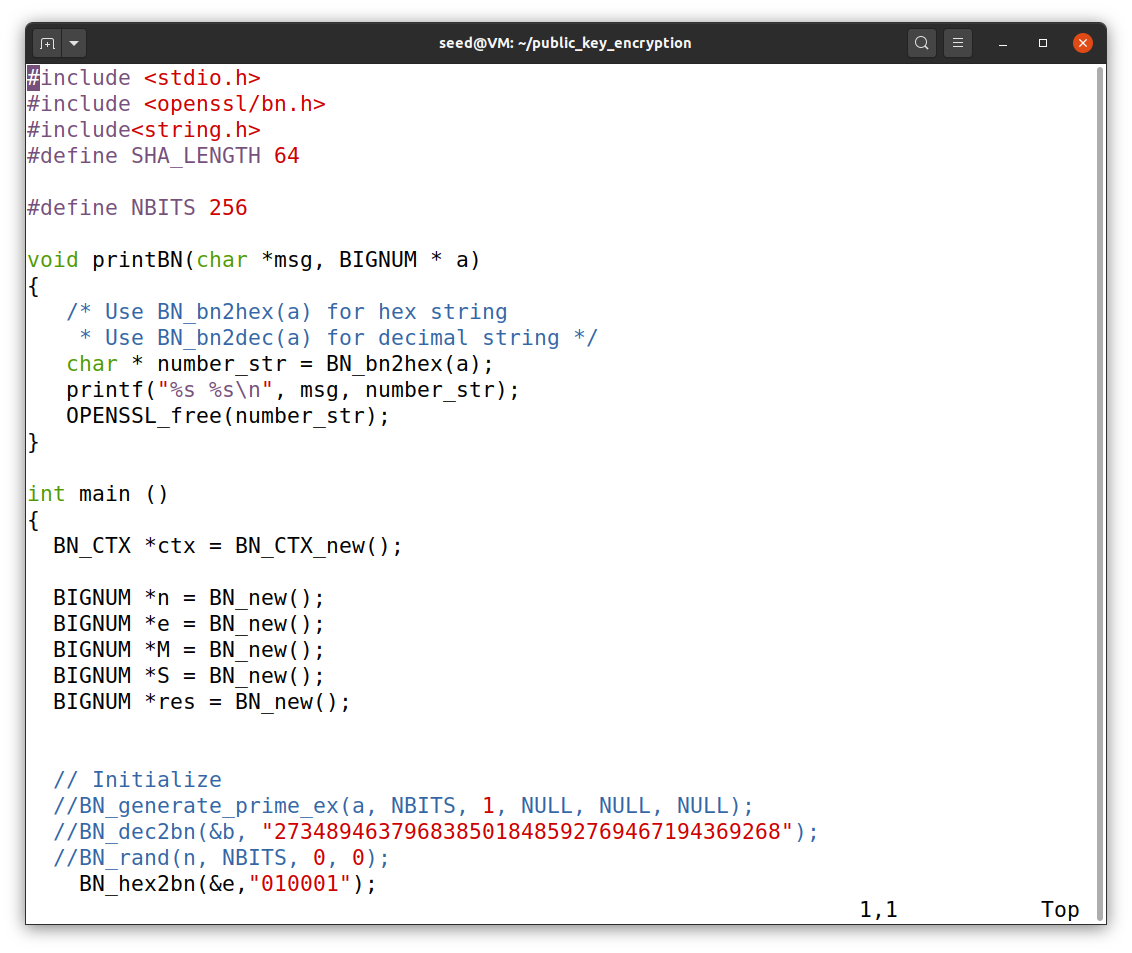
n =

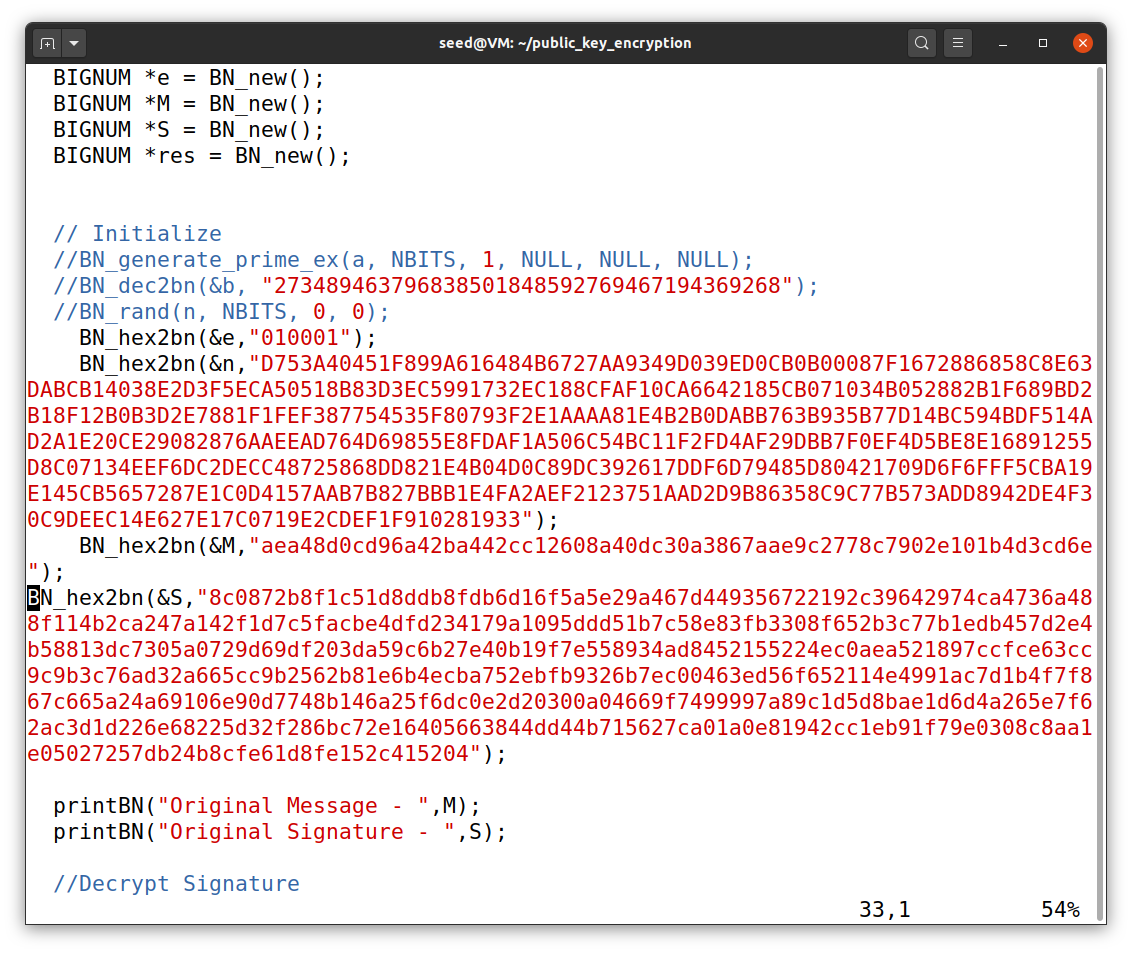


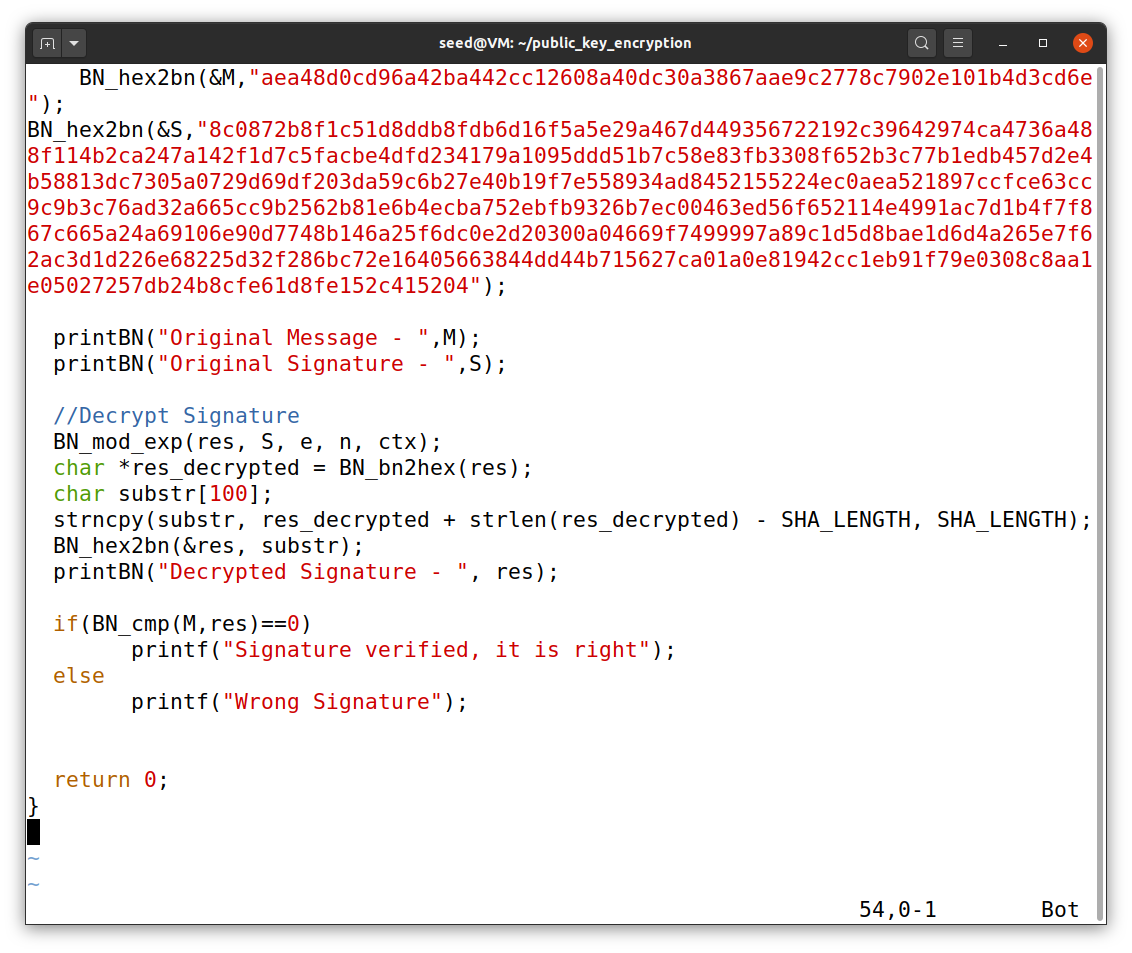
e = 010001

S =

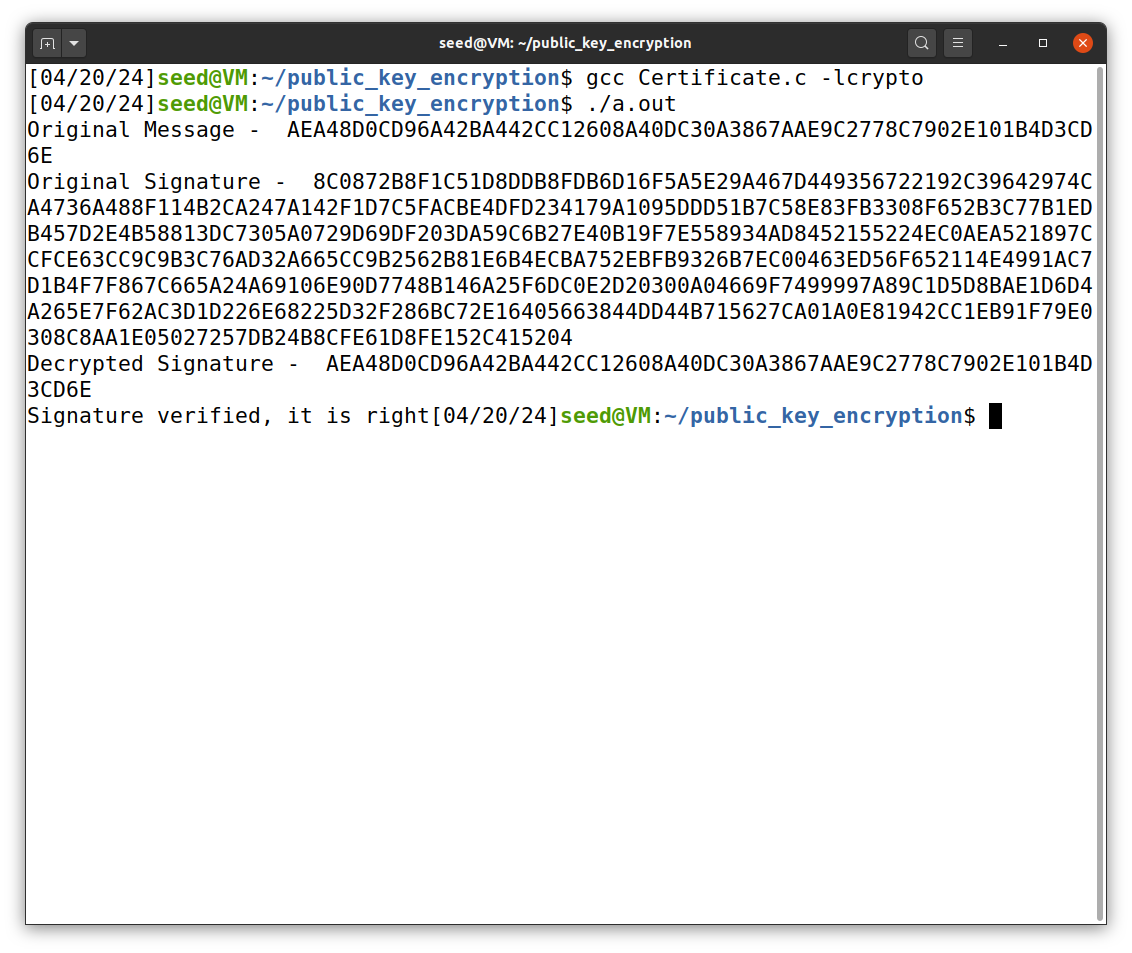
  
  
M = aea48d0cd96a42ba442cc12608a40dc30a3867aae9c2778c7902e101b4d3cd6e







Now we compile the above program using gcc compiler and run the program.



Thus we have successfully verified the signature to be valid signature from the CA.