**ACS 54500 Cryptography and Network Security – Lab 10**

**Task – 1:**

In this task, we are given the hexadecimal values of p, q and exponent e, and are asked to calculate the private key d. This can be done using the sample program that uses the BIGNUM APIs, modifying it to calculate d, given the values of p, e and q. The value of the modulus, n can be found using the formula, *n = p \* q*.

Given values of p, q and e:

*p = F7E75FDC469067FFDC4E847C51F452DF*

*q = E85CED54AF57E53E092113E62F436F4F*

*e = 0D88C3*

The modified program code is given as follows -

/\* 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 \*p = BN\_new();

    BIGNUM \*q = BN\_new();

    BIGNUM \*e = BN\_new();

    BIGNUM \*n = BN\_new();

    BIGNUM \*d = BN\_new();

    BIGNUM \*phi = BN\_new();

    BIGNUM \*p\_minus\_one = BN\_new();

    BIGNUM \*q\_minus\_one = BN\_new();

    // Initialize p, q, e

    BN\_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");

    BN\_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

    BN\_hex2bn(&e, "0D88C3");

    // n = a\*b

    BN\_mul(n, p, q, ctx);

    BN\_sub(p\_minus\_one, p, BN\_value\_one());     // Compute p-1

    BN\_sub(q\_minus\_one, q, BN\_value\_one());     // Compute q-1

    // find phi(n)

    BN\_mul(phi, p\_minus\_one, q\_minus\_one, ctx); // Compute phi(n)

    // Compute the private key exponent d, s.t. ed mod phi(n) = 1

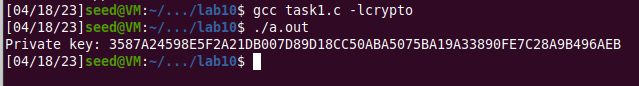
    BN\_mod\_inverse(d, e, phi, ctx);

    printBN("Private key:", d);

    return 0;

}

Now that we have the modified program, we compile it using the GCC compiler and then run the program as follows:



Therefore, the value of the private key, d is: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

**Task – 2:**

In this task, we are given the hexadecimal values of the modulus n, exponent e and private key d. We are asked to encrypt a given message by converting it from an ASCII string to a hex string, and then using the given values and sample program to encrypt.

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

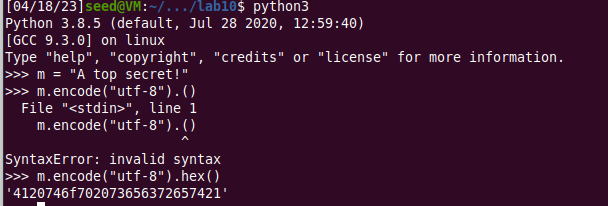
*n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5*

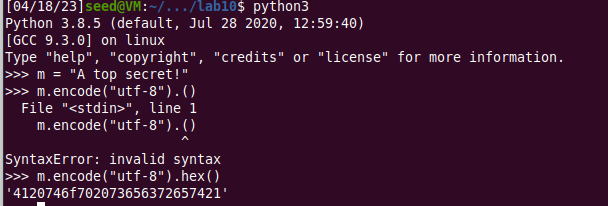
*e = 010001 (this hex value equals to decimal 65537)*

*M = A top secret!*

*d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D*

Now, we have to convert the given ASCII message string to a hex string. The lab description specified a python3 command, but that didn’t work, so I used the following command to convert the message to a hex string:





So, the hexadecimal value of M is: 4120746f702073656372657421.

The modified program code is given as follows -

/\* 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 \*d = BN\_new();

    BIGNUM \*C = BN\_new();

    BIGNUM \*res = BN\_new();

    // Initialize e, n, M, d

    BN\_hex2bn(&e, "010001");

    BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

    BN\_hex2bn(&M, "4120746f702073656372657421");

    BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // Encrypted result, C = Mˆe mod n

    BN\_mod\_exp(C, M, e, n, ctx);

    printBN("Mˆe mod n = ", C);

    // For verification, the decrypted result, res = Cˆd mod n

    BN\_mod\_exp(res, C, d, n, ctx);

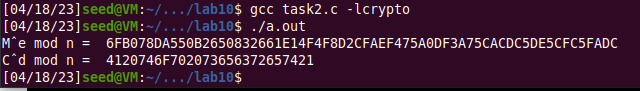
    printBN("Cˆd mod n = ", res);

    return 0;

}

We use the given value of the private key to verify if our encrypted result is right by decrypting the encrypted result.

Now that we have the modified program, we compile it using the GCC compiler and then run the program as follows:



Thus, the encrypted result is correct, because on decryption, we get the same value of M.

**Task – 3:**

In this task, we are given the hexadecimal values of the modulus n, exponent e and ciphertext C. We are asked to decrypt the given ciphertext by using the given values and sample program. We then convert the decrypted hexadecimal string back to an ASCII string to find out what the original message was.

The given values of n, e, C and d are:

*n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5*

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

*C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F*

*d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D*

The modified program code is given as follows -

/\* 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 \*d = BN\_new();

    BIGNUM \*C = BN\_new();

    BIGNUM \*res = BN\_new();

    // Initialize e, n, C, d

    BN\_hex2bn(&e, "010001");

    BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

    BN\_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

    BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // The decrypted result, M = Cˆd mod n

    BN\_mod\_exp(M, C, d, n, ctx);

    printBN("Cˆd mod n = ", M);

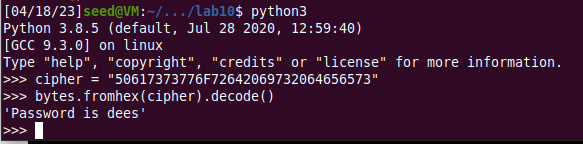
    return 0;

}

Now that we have the modified program, we compile it using the GCC compiler and then run the program as follows:



Now, we have to convert the given hex string back into an ASCII message string. The lab description specified a python3 command, but that didn’t work, so I used the following command to convert the hex string back into an ASCII message:



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

**Task – 4:**

In this task, we are given the hexadecimal values of the modulus n, exponent e, private key d and a message M1. We are asked to generate a signature for the given message by signing it directly instead of its hash value. We are also asked to modify the message by one or two characters, sign it and compare both signatures.

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

*n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5*

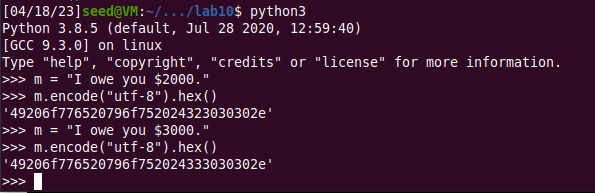
*e = 010001 (this hex value equals to decimal 65537)*

*M1 = I owe you $2000.*

*M2 = I owe you $3000.*

*d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D*

Now, we convert both messages to hexadecimal strings as shown below -



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

*M1 = 49206f776520796f752024323030302e*

*M2 = 49206f776520796f752024333030302e*

We can see that they differ by a couple of digits.

The modified program code is given as follows -

/\* 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 \*M1 = BN\_new();

    BIGNUM \*d = BN\_new();

    BIGNUM \*M2 = BN\_new();

    BIGNUM \*res1 = BN\_new();

    BIGNUM \*res2 = BN\_new();

    // Initialize e, n, M1, M2, d

    BN\_hex2bn(&e, "010001");

    BN\_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

    BN\_hex2bn(&M1, "49206f776520796f752024323030302e");

    BN\_hex2bn(&M2, "49206f776520796f752024333030302e");

    BN\_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // Signed results, res1 = M1ˆd mod n and M2^d mod n

    BN\_mod\_exp(res1, M1, e, n, ctx);

    printBN("M1ˆd mod n = ", res1);

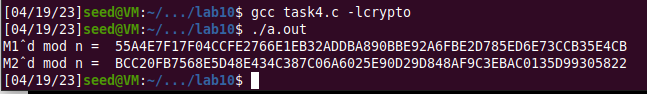
    BN\_mod\_exp(res2, M2, d, n, ctx);

    printBN("M2ˆd mod n = ", res2);

    return 0;

}

Now that we have the modified program, we compile it using the GCC compiler and then run the program as follows:



We can see that even though the messages differed by a single digit, the generated signatures vary and are so different. This shows that even one difference in a digit or character when generating signatures can cause so much of a difference.

**Task – 5:**

In this task, we are given the hexadecimal values of the modulus n, exponent e, a signature S1, and a message M1. We are asked to verify if the generated signature is the right one for the given message. We are also asked to modify it by one bit and describe what happens.

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

*M = Launch a missile.*

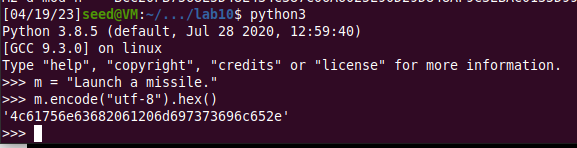
*S1 = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F*

*S2 = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F*

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

*n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115*

Now, we convert the message into a hexadecimal string as shown below -



The modified program code is given as follows -

/\* 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);

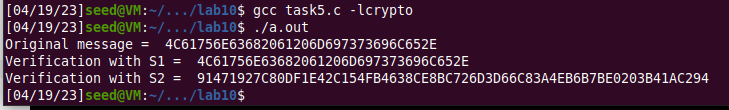
    BN\_mod\_exp(res2, S2, e, n, ctx);

    printBN("Verification with S2 = ", res2);

    return 0;

}

Now that we have the modified program, we compile it using the GCC compiler and then run the program as follows:



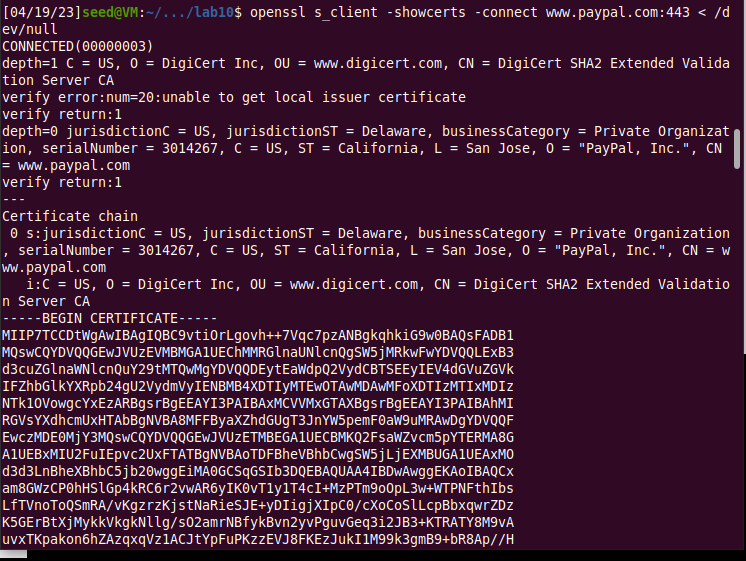
Therefore, S1 is the correct signature because it gives back the original message value. We can also see that if we modify the signature by even one bit, it completely changes the generated message.

**Task – 6:**

In this task, we are asked to download a real X.509 certificate from the paypal.com web server, get its issuer’s public key, and then use this public key to manually verify the signature on the certificate on the certificate using a program. An X.509 certificate contains data about a public key and an issuer’s signature on the data.

**Task – 6.1:**

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



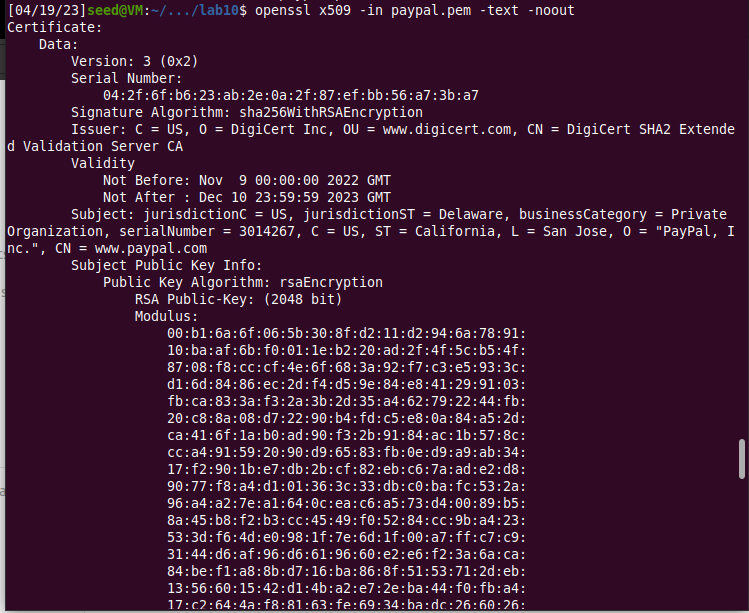
Now, we copy and paste each of the certificates (the text between the line containing "Begin CERTIFICATE" and the line containing "END CERTIFICATE", including these two lines) to two files. We call the first one paypal.pem and the second one paypal1.pem.



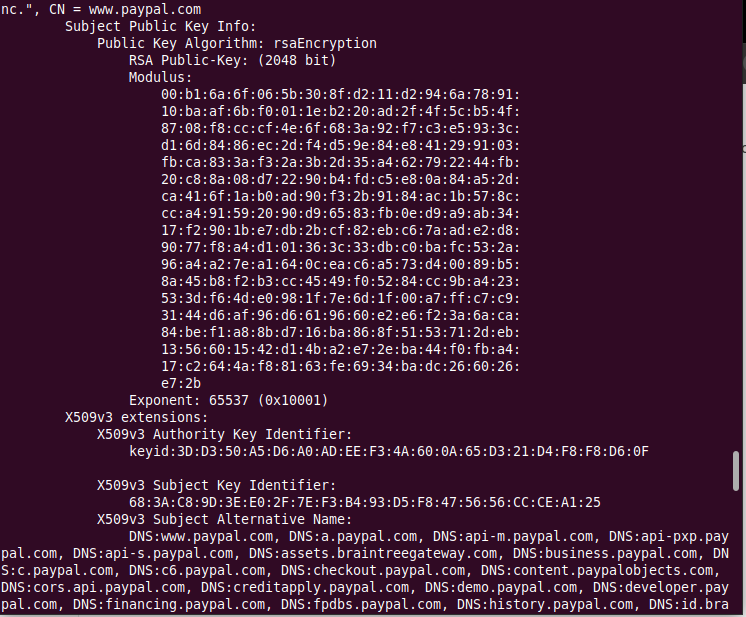
**Task – 6.2:**

In this sub-task, we need to extract the public key (e, n) from the issuer’s certificate. Openssl provides commands to extract certain attributes from the x509 certificates. We can extract the value of n using the flag -modulus. There is no specific command to extract e, but we can print out all the fields and can easily find the value of e.

The below screenshots show these commands and the outputs we get with them -



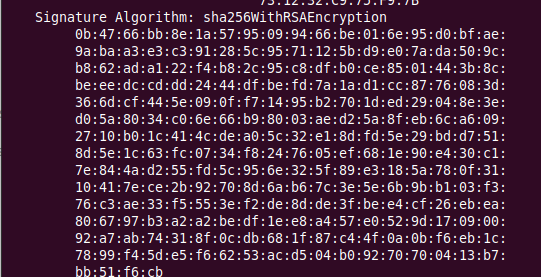
The values of the modulus, n and the exponent, e are:



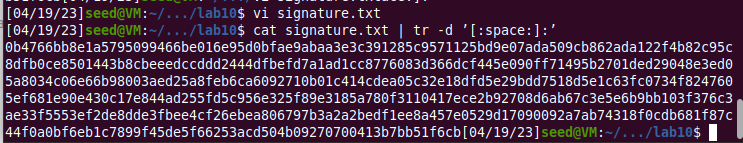
The hexadecimal value of the exponent, e is: *65537 (0x10001).*

**Task – 6.3:**

In this sub-task, we extract the signature from the server’s certicate.



There is no specific openssl command to extract the signature field. However, we can print out all the fields and then copy and paste the signature block into a file, as shown below. We also use the tr command to remove spaces and colons from the file -

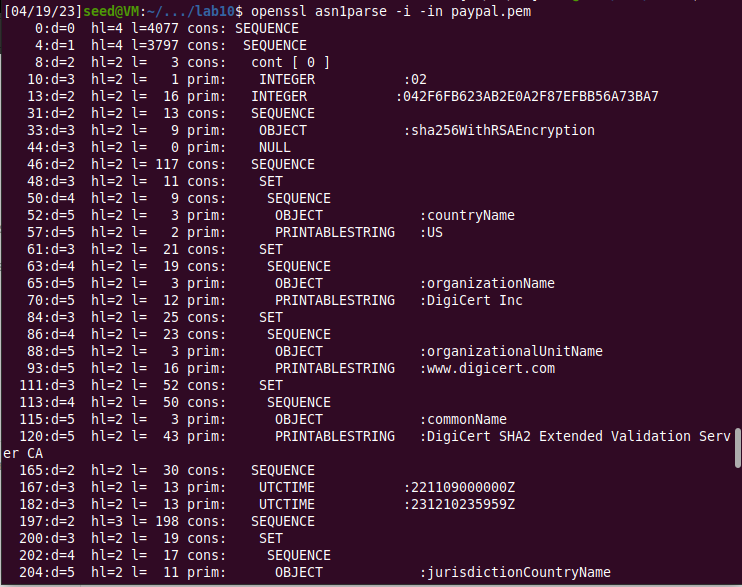


Thus, the hexadecimal value of the signature, S is: **

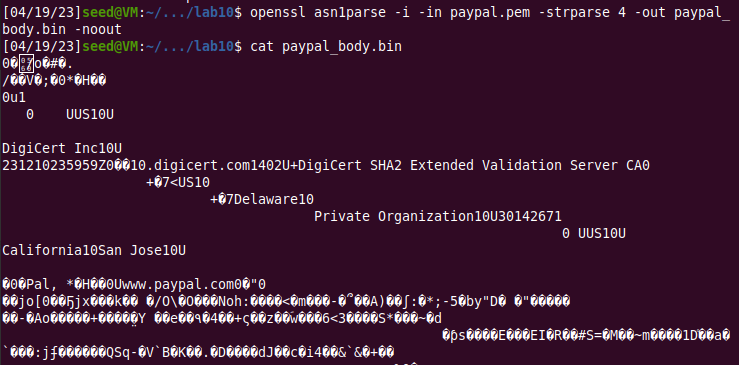
**Task – 6.4:**

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, we need to exclude the signature block of a certificate when computing the hash.

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



Once we have the body of the certificate, we calculate its hash using the following command:



So, the hash of the body of the certificate is = *710e017b0e642997a3115fb9c4f0a1f8e5378a8578e6e05a06b43b4d21b7fc09*

**Task – 6.5:**

In this sub-task, we are given the hexadecimal values of the modulus n, exponent e, CA’s signature S and a message M. We are asked to verify whether this signature is valid or not by writing a program.

The given values of n, e, S and M are:

*n

*e = 10001*

*S = *

*M = 710e017b0e642997a3115fb9c4f0a1f8e5378a8578e6e05a06b43b4d21b7fc09*

The modified program code is given as follows -

/\* 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(&n

    BN\_hex2bn(&e, "65537");

    BN\_hex2bn(&S, "");

    BN\_hex2bn(&M, "710e017b0e642997a3115fb9c4f0a1f8e5378a8578e6e05a06b43b4d21b7fc09");

    // 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");

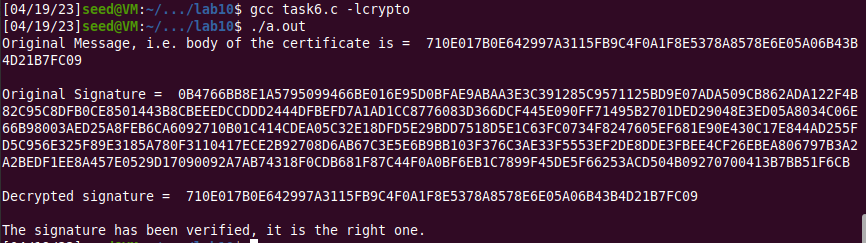
    else

        printf("Wrong signature\n");

return 0;

}

Now that we have the modified program, we compile it using the GCC compiler and then run the program as follows:



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