CPS 633 Section 09

RSA Public-Key Encryption and Signature Lab

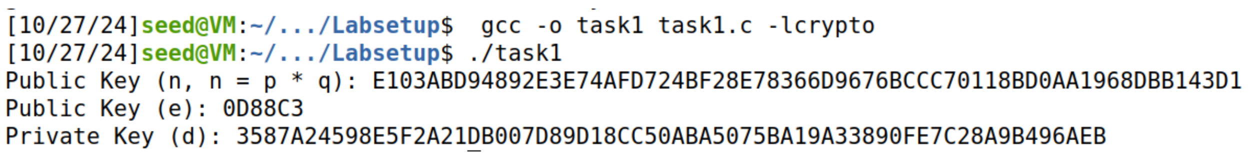
Group 18

Roxie Reginold (501087897)
Hetu Virajkumar Patel (501215707)
Sayyada Aisha Mehvish (501106795)

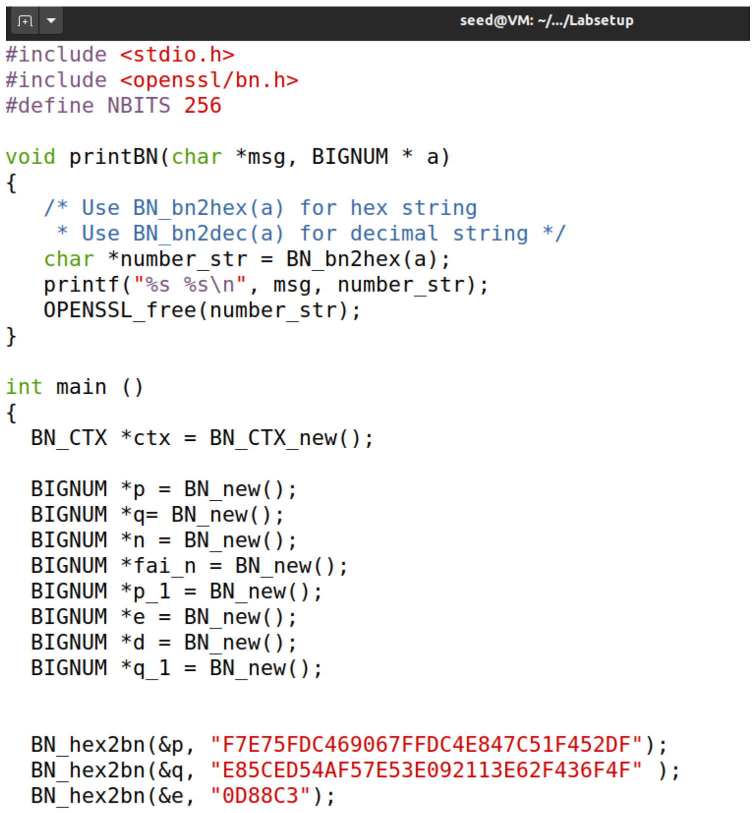
Task 1: Deriving the Private Key

In this task, we have the given information:
p and q are 512 bits long (the ones used here are only 128 bits).
p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3

To calculate the private key (d), we first need to calculate the value of the public key (p *q).
Then we calculate the value of phi, which is (p-1)(q-1).
Lastly, we use phi and e to calculate d (the modular inverse of e mod phi).



```
[10/27/24]seed@VM:~/.../Labsetup$  gcc -o task1 task1.c -lcrypto
[10/27/24]seed@VM:~/.../Labsetup$ ./task1
Public Key (n, n = p * q): E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
Public Key (e): 0D88C3
Private Key (d): 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```



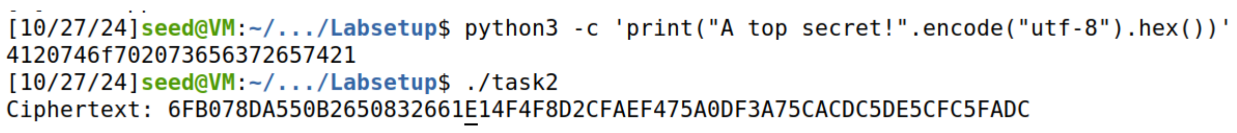
```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

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 *n = BN_new();
  BIGNUM *fai_n = BN_new();
  BIGNUM *p_1 = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *d = BN_new();
  BIGNUM *q_1 = BN_new();


  BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
  BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F" );
  BN_hex2bn(&e, "0D88C3");
```

```
    BN_mul(n, p, q, ctx);
    printBN("Public key n =", n);

    BN_sub(p_1, p, BN_value_one());
    BN_sub(q_1, q, BN_value_one());
    BN_mul(fai_n, p_1, q_1, ctx);
    // printBN("fai_n =", fai_n);

    BN_mod_inverse(d, e, fai_n, ctx);
    printBN("Private key d =", d);
    printBN("Public key e =", e);

    return 0;
}
```

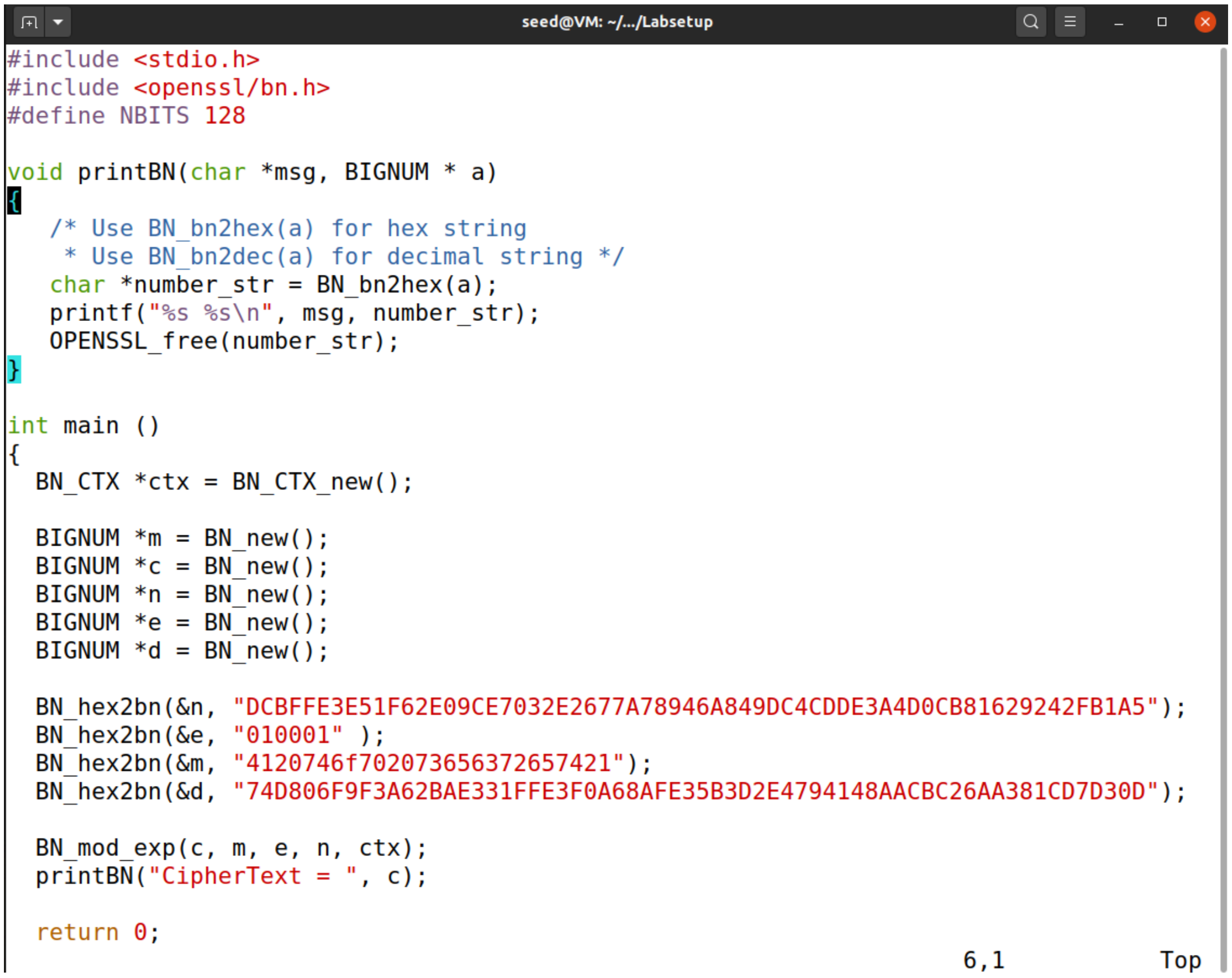
Task 2: Encrypting a Message

Let (e, n) be the public key. Please encrypt the message "A top secret!" (the quotations are not included). We need to convert this ASCII string to a hex string, and then convert the hex string to a BIGNUM using the hex-to-bn API BN hex2bn().

After finding the hexadecimal conversion of the message and using BN_hex2bn(), we encrypt the message using the formula c = m^e mod n, where n and e are given.
We get the resulting Ciphertext using the following C program below:



```
[10/27/24]seed@VM:~/.../Labsetup$ python3 -c 'print("A top secret!".encode("utf-8").hex())'
4120746f702073656372657421
[10/27/24]seed@VM:~/.../Labsetup$ ./task2
Ciphertext: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

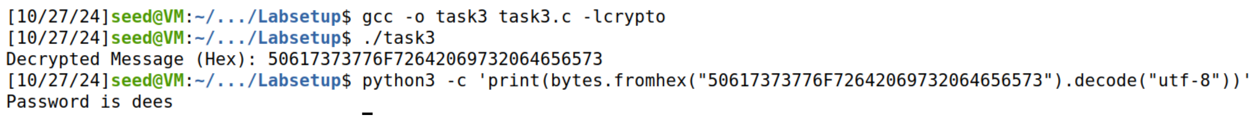


```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128

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 *m = BN_new();
  BIGNUM *c = BN_new();
  BIGNUM *n = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *d = BN_new();

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

  BN_mod_exp(c, m, e, n, ctx);
  printBN("CipherText = ", c);

  return 0;
                                                                6,1           Top
```

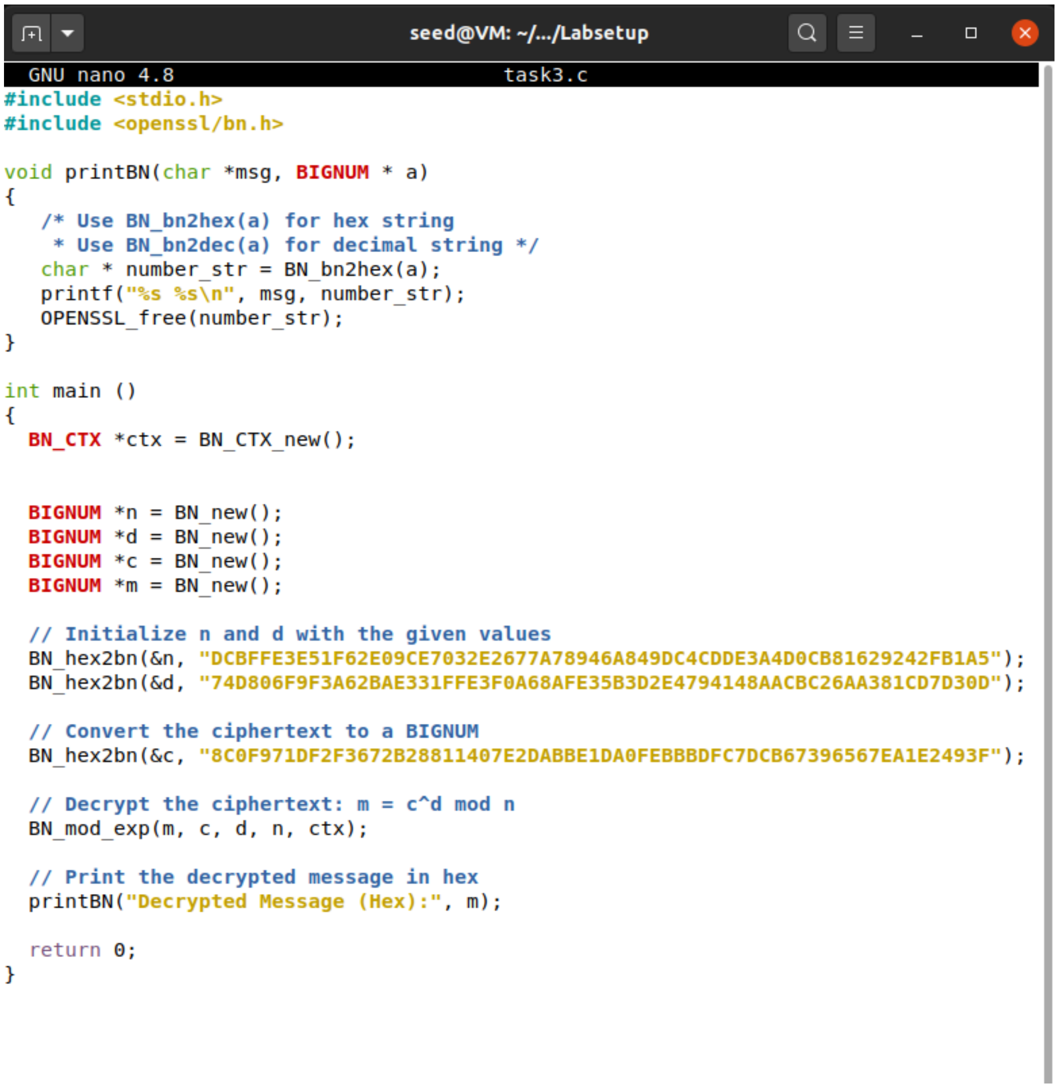
Task 3: Decrypting a Message

The public/private keys used in this task are the same as the ones used in Task 2. Please decrypt the following ciphertext C, and convert it back to a plain ASCII string.

We will decrypted the given ciphertext using the equation m=c^d mod n, then we convert the resulting hexadecimal result to ASCII using python below, therefore the decrypted ciphertext is “Password is dess”:



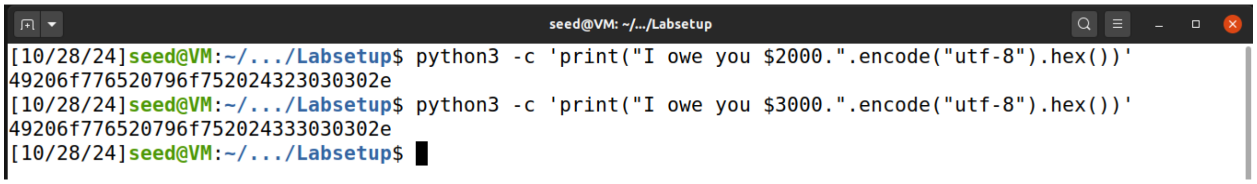
```
[10/27/24]seed@VM:~/.../Labsetup$ gcc -o task3 task3.c -lcrypto
[10/27/24]seed@VM:~/.../Labsetup$ ./task3
Decrypted Message (Hex): 50617373776F726420697320646565573
[10/27/24]seed@VM:~/.../Labsetup$ python3 -c 'print(bytes.fromhex("50617373776F726420697320646565573").decode("utf-8"))'
Password is dees
```



```
GNU nano 4.8                           task3.c
#include <stdio.h>
#include <openssl/bn.h>

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 *d = BN_new();
  BIGNUM *c = BN_new();
  BIGNUM *m = BN_new();

  // Initialize n and d with the given values
  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

  // Convert the ciphertext to a BIGNUM
  BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

  // Decrypt the ciphertext: m = c^d mod n
  BN_mod_exp(m, c, d, n, ctx);

  // Print the decrypted message in hex
  printBN("Decrypted Message (Hex):", m);

  return 0;
}
```

Task 4: Signing a Message

In this task, we first convert the two messages into hex codes by running the following:

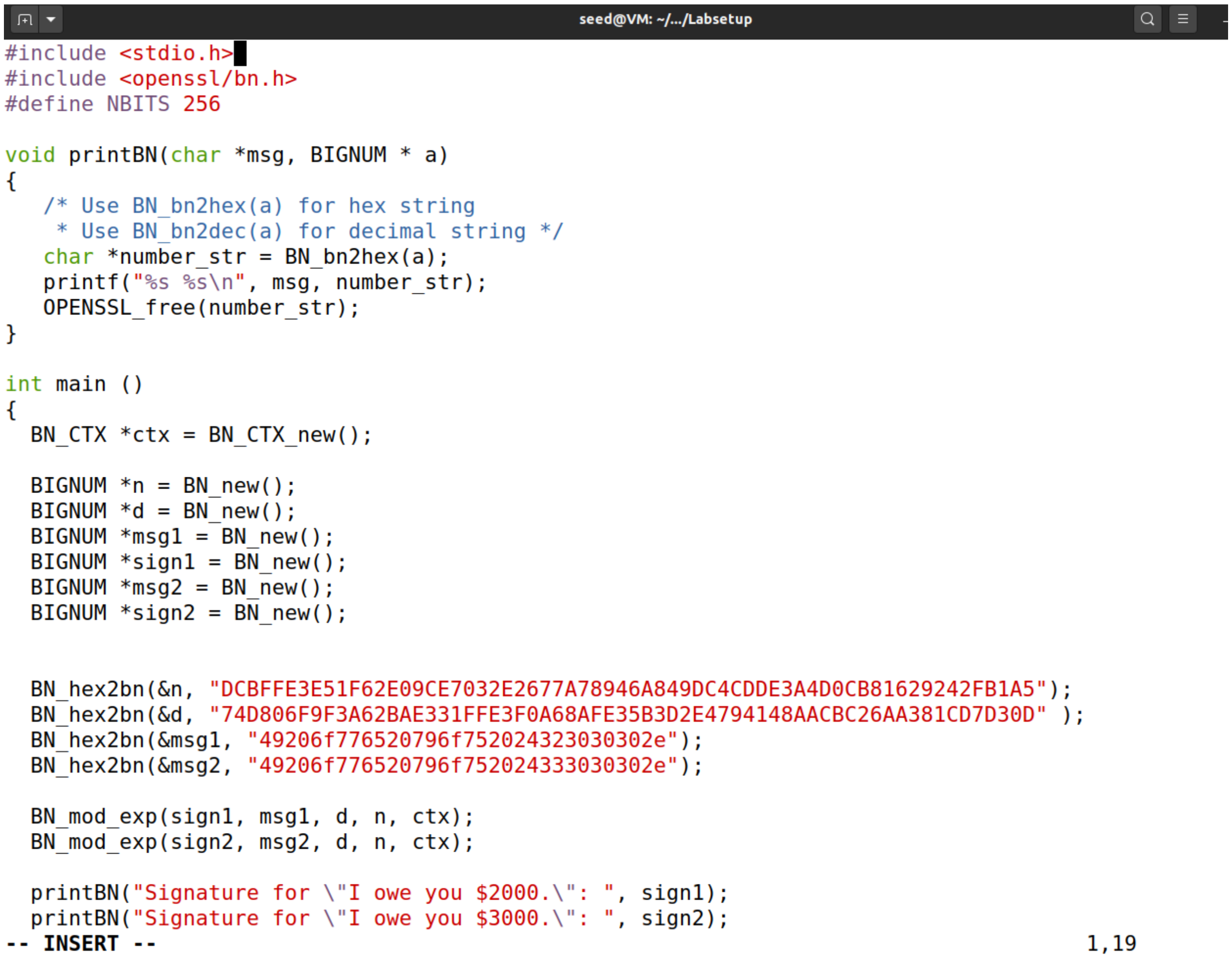


```
[10/28/24]seed@VM:~/.../Labsetup$ python3 -c 'print("I owe you $2000.".encode("utf-8").hex())'
49206f776520796f752024323030302e
[10/28/24]seed@VM:~/.../Labsetup$ python3 -c 'print("I owe you $3000.".encode("utf-8").hex())'
49206f776520796f752024333030302e
[10/28/24]seed@VM:~/.../Labsetup$ 
```

The difference in the hex codes is underlined:
49206f776520796f75202432<u>3</u>030302e — underline on the "2"
49206f776520796f75202433<u>3</u>030302e

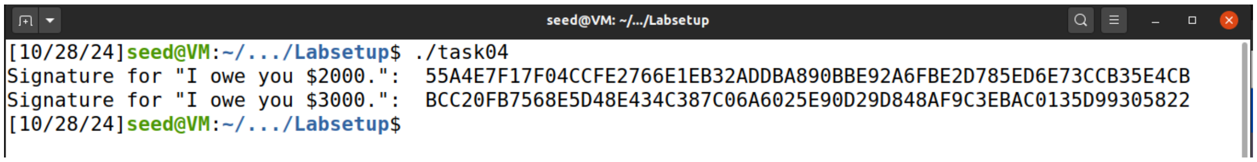
We then created the signatures for both messages using private exponent d and the modulus n and the formula Signature = Message^d mod n.

Code for Task4



```
#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 *d = BN_new();
  BIGNUM *msg1 = BN_new();
  BIGNUM *sign1 = BN_new();
  BIGNUM *msg2 = BN_new();
  BIGNUM *sign2 = BN_new();


  BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
  BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D" );
  BN_hex2bn(&msg1, "49206f776520796f752024323030302e");
  BN_hex2bn(&msg2, "49206f776520796f752024333030302e");

  BN_mod_exp(sign1, msg1, d, n, ctx);
  BN_mod_exp(sign2, msg2, d, n, ctx);

  printBN("Signature for \"I owe you $2000.\": ", sign1);
  printBN("Signature for \"I owe you $3000.\": ", sign2);
-- INSERT --                                                                                 1,19
```

Observations

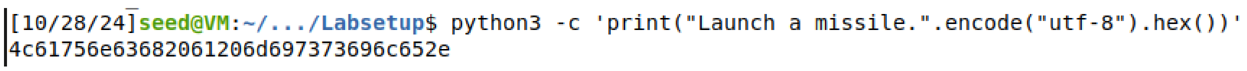


```
[10/28/24]seed@VM:~/.../Labsetup$ ./task04
Signature for "I owe you $2000.":  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
Signature for "I owe you $3000.":  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
[10/28/24]seed@VM:~/.../Labsetup$
```

After running the code we compare the signatures of the two messages. We notice that the signatures are completely different even though there was a small change in the original message This illustrates the sensitivity of cryptographic signatures to any difference in the message, which is a critical property for message integrity and authenticity.

Task 5: Verifying a Signature

In this task we first ran to get the hexadecimal representation of the message "Launch a missile.".

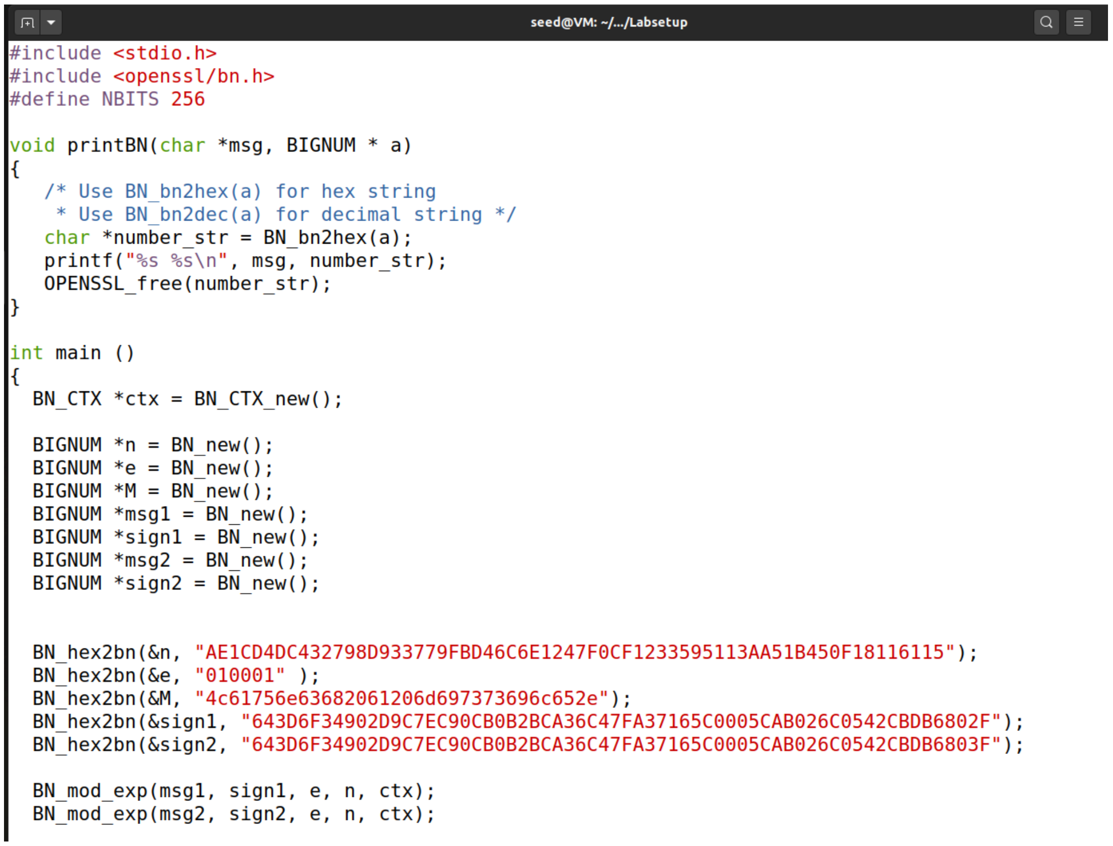


```
[10/28/24]seed@VM:~/.../Labsetup$ python3 -c 'print("Launch a missile.".encode("utf-8").hex())'
4c61756e63682061206d697373696c652e
```

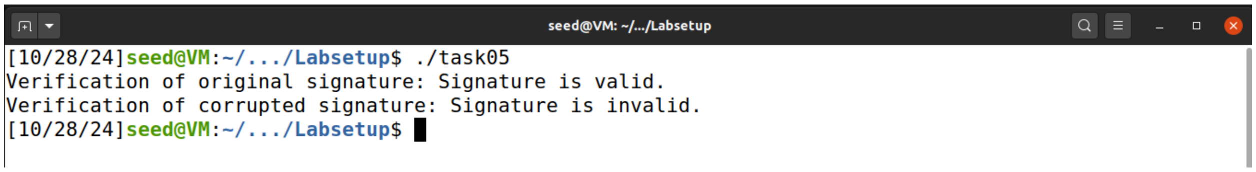
The process involves comparing the hexadecimal representation of the message "Launch a missile." with msg1. A match confirms that Alice's private key generated the signature from the original message, thus validating its authenticity. This comparison is critical in establishing the message's integrity and sender verification.

This task involves the following steps:

- Transform the signature from its hexadecimal format into its corresponding numerical value.
- Apply Alice's public key components (e, n) to decrypt S. This is achieved by calculating S^e mod n.
- Generate a hash of the message (M) using the identical hash function employed in the signature's creation. Then, compare this hash with the decrypted signature.



```
#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 *msg1 = BN_new();
  BIGNUM *sign1 = BN_new();
  BIGNUM *msg2 = BN_new();
  BIGNUM *sign2 = BN_new();


  BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
  BN_hex2bn(&e, "010001" );
  BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
  BN_hex2bn(&sign1, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
  BN_hex2bn(&sign2, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

  BN_mod_exp(msg1, sign1, e, n, ctx);
  BN_mod_exp(msg2, sign2, e, n, ctx);
```

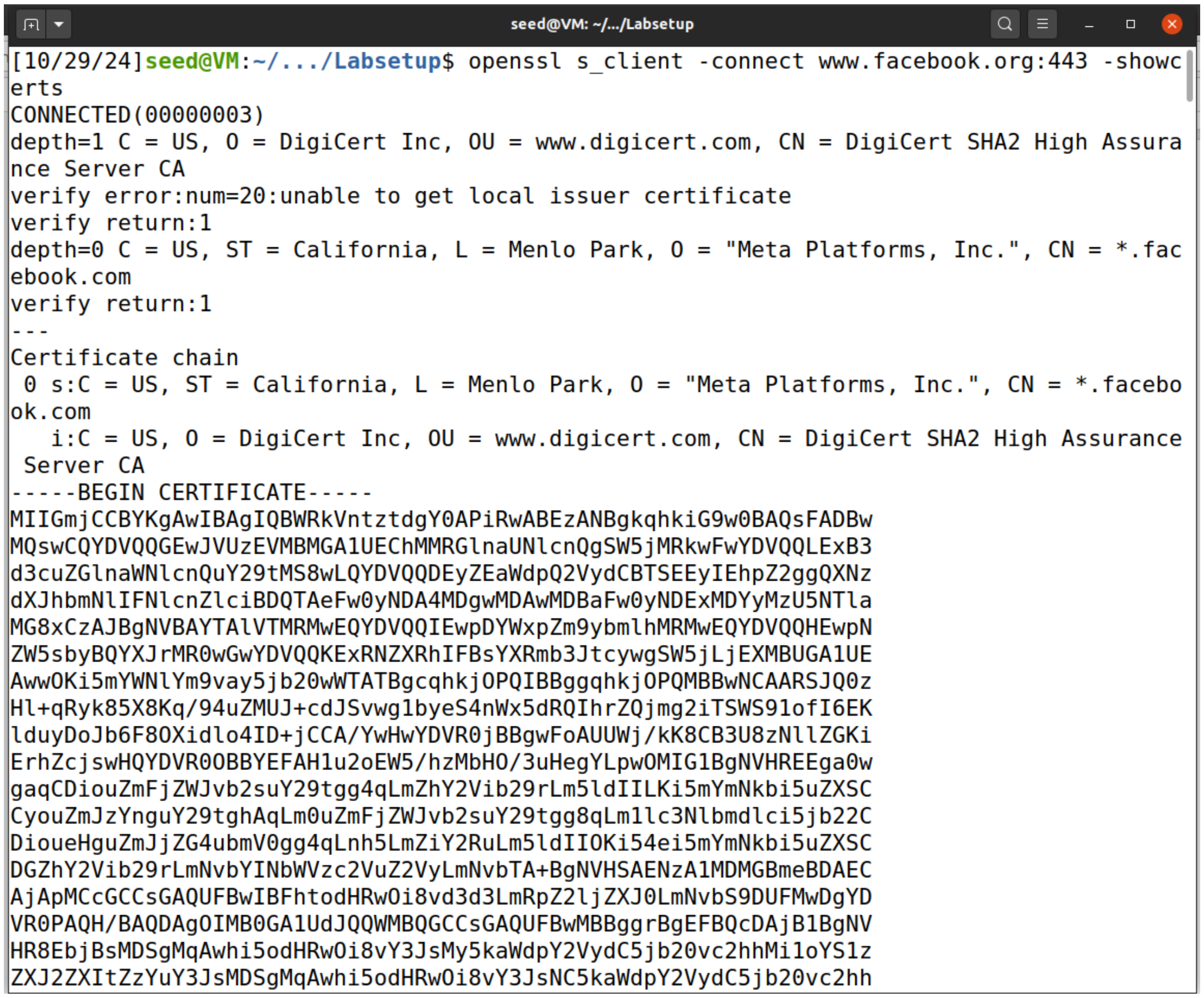


```
[10/28/24]seed@VM:~/.../Labsetup$ ./task05
Verification of original signature: Signature is valid.
Verification of corrupted signature: Signature is invalid.
[10/28/24]seed@VM:~/.../Labsetup$ 
```

The signature is no longer valid after changing
S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB680<u>2F</u>
to
S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB680<u>3F</u>
And therefore the signature has become corrupted. This demonstrates the integrity feature of RSA signatures: any interference with the signature is readily detectable, ensuring the message remains unaltered.

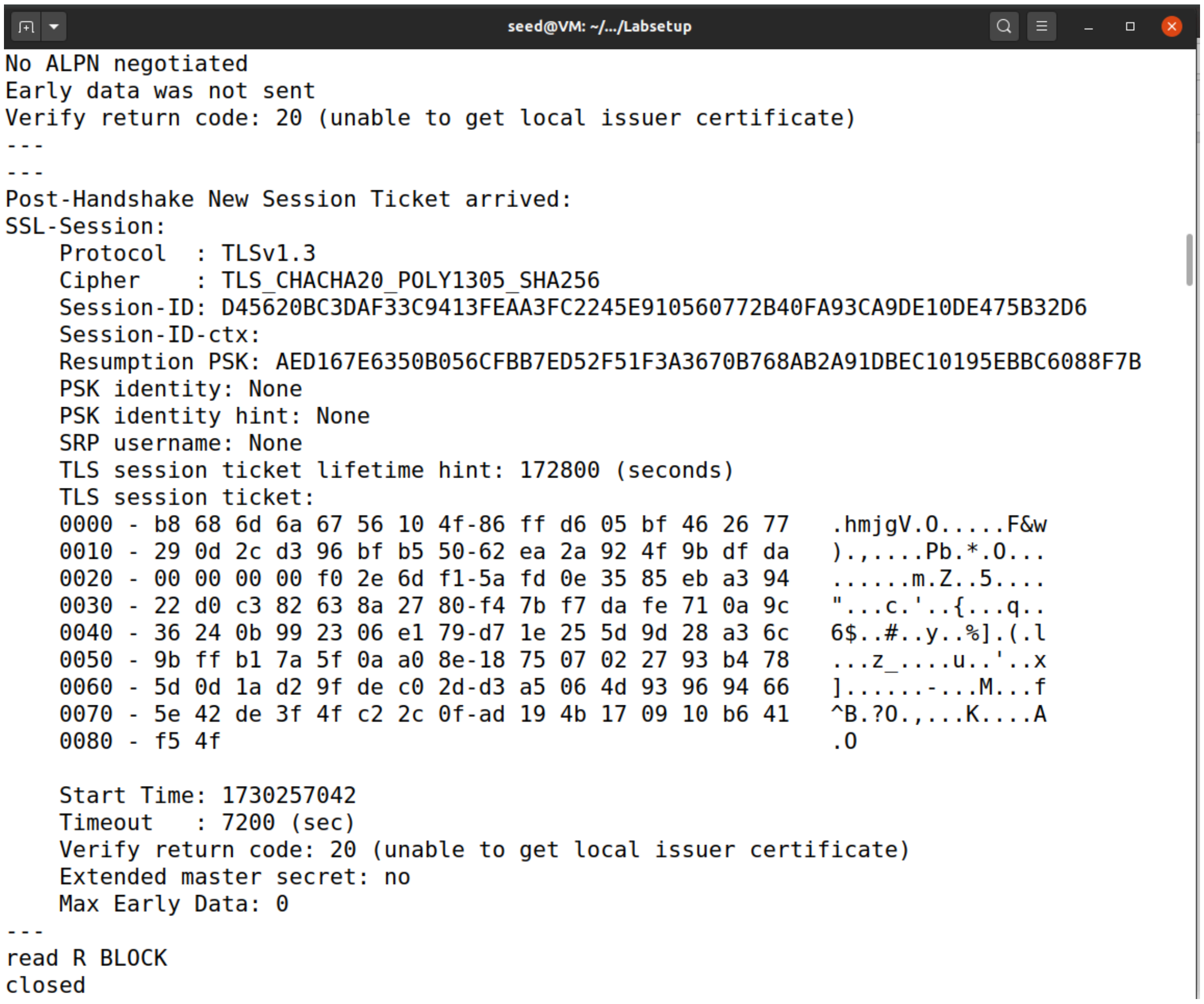
Task 6: Manually Verifying an X.509 Certificate

Step 1: Download a certificate from a real web server



seed@VM: ~/.../Labsetup

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl s_client -connect www.facebook.org:443 -showc
erts
CONNECTED(00000003)
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assura
nce Server CA
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Menlo Park, O = "Meta Platforms, Inc.", CN = *.fac
ebook.com
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Menlo Park, O = "Meta Platforms, Inc.", CN = *.facebo
ok.com
   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance
 Server CA
-----BEGIN CERTIFICATE-----
MIIGmjCCBYKgAwIBAgIQBWRkVntztdgY0APiRwABEzANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMS8wLQYDVQQDEyZEaWdpQ2VydCBTSEEyIEhpZ2ggQXNz
dXJhbmNlIFNlcnZlciBDQTAeFw0yNDA4MDgwMDAwMDBaFw0yNDExMDYyMzU5NTla
MG8xCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRMwEQYDVQQHEwpN
ZW5sbyBQYXJrMR0wGwYDVQQKExRNZXRhIFBsYXRmb3JtcywgSW5jLjEXMBUGA1UE
AwwOKi5mYWNlYm9vay5jb20wWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAARSJQ0z
Hl+qRyk85X8Kq/94uZMUJ+cdJSvwg1byeS4nWx5dRQIhrZQjmg2iTSWS91ofI6EK
lduyDoJb6F8OXidlo4ID+jCCA/YwHwYDVR0jBBgwFoAUUWj/kK8CB3U8zNllZGKi
ErhZcjswHQYDVR0OBBYEFAH1u2oEW5/hzMbHO/3uHegYLpwOMIG1BgNVHREEga0w
gaqCDiouZmFjZWJvb2suY29tgg4qLmZhY2Vib29rLm5ldIILKi5mYmNkbi5uZXSC
CyouZmJzYnguY29tghAqLm0uZmFjZWJvb2suY29tgg8qLm1lc3Nlbmdlci5jb22C
DioueHguZmJjZG4ubmV0gg4qLnh5LmZiY2RuLm5ldIIOKi54ei5mYmNkbi5uZXSC
DGZhY2Vib29rLmNvbYINbWVzc2VuZ2VyLmNvbTA+BgNVHSAENzA1MDMGBmeBDAEC
AjApMCcGCCsGAQUFBwIBFhtodHRwOi8vd3d3LmRpZ2ljZXJ0LmNvbS9DUFMwDgYD
VR0PAQH/BAQDAgOIMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjB1BgNV
HR8EbjBsMDSgMqAwhi5odHRwOi8vY3JsMy5kaWdpY2VydC5jb20vc2hhMi1oYS1z
ZXJ2ZXItZzYuY3JsMDSgMqAwhi5odHRwOi8vY3JsNC5kaWdpY2VydC5jb20vc2hh
```



```
No ALPN negotiated
Early data was not sent
Verify return code: 20 (unable to get local issuer certificate)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_CHACHA20_POLY1305_SHA256
    Session-ID: D45620BC3DAF33C9413FEAA3FC2245E910560772B40FA93CA9DE10DE475B32D6
    Session-ID-ctx:
    Resumption PSK: AED167E6350B056CFBB7ED52F51F3A3670B768AB2A91DBEC10195EBBC6088F7B
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 172800 (seconds)
    TLS session ticket:
    0000 - b8 68 6d 6a 67 56 10 4f-86 ff d6 05 bf 46 26 77   .hmjgV.O.....F&w
    0010 - 29 0d 2c d3 96 bf b5 50-62 ea 2a 92 4f 9b df da   ).,....Pb.*.O...
    0020 - 00 00 00 00 f0 2e 6d f1-5a fd 0e 35 85 eb a3 94   ......m.Z..5....
    0030 - 22 d0 c3 82 63 8a 27 80-f4 7b f7 da fe 71 0a 9c   "...c.'..{...q..
    0040 - 36 24 0b 99 23 06 e1 79-d7 1e 25 5d 9d 28 a3 6c   6$..#..y..%].(.l
    0050 - 9b ff b1 7a 5f 0a a0 8e-18 75 07 02 27 93 b4 78   ...z_....u..'..x
    0060 - 5d 0d 1a d2 9f de c0 2d-d3 a5 06 4d 93 96 94 66   ]......-...M...f
    0070 - 5e 42 de 3f 4f c2 2c 0f-ad 19 4b 17 09 10 b6 41   ^B.?O.,...K....A
    0080 - f5 4f                                             .O

    Start Time: 1730257042
    Timeout   : 7200 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
closed
```

From the results of the command above, we get the following certificates:

C0.perm

-----BEGIN CERTIFICATE-----
MIIGmjCCBYKgAwIBAgIQBWRkVntztdgY0APiRwABEzANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMS8wLQYDVQQDEyZEaWdpQ2VydCBTSEEyIEhpZ2ggQXNz
dXJhbmNlIFNlcnZlciBDQTAeFw0yNDA4MDgwMDAwMDBaFw0yNDExMDYyMzU5NTla
MG8xCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRMwEQYDVQQHEwpN
ZW5sbyBQYXJrMR0wGwYDVQQKExRNZXRhIFBsYXRmb3JtcywgSW5jLjEXMBUGA1UE
AwwOKi5mYWNlYm9vay5jb20wWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAARSJQ0z
HI+qRyk85X8Kq/94uZMUJ+cdJSvwg1byeS4nWx5dRQIhrZQjmg2iTSWS91ofI6EK
IduyDoJb6F8OXidIo4ID+jCCA/YwHwYDVR0jBBgwFoAUUWj/kK8CB3U8zNllZGKi
ErhZcjswHQYDVR0OBBYEFAH1u2oEW5/hzMbHO/3uHegYLpwOMIG1BgNVHREEga0w
gaqCDiouZmFjZWJvb2suY29tgg4qLmZhY2Vib29rLm5ldIILKi5mYmNkbi5uZXSC
CyouZmJzYnguY29tghAqLm0uZmFjZWJvb2suY29tgg8qLm1lc3Nlbmdlci5jb22C

DioueHguZmJjZG4ubmV0gg4qLnh5LmZiY2RuLm5ldIIOKi54ei5mYmNkbi5uZXSC
DGZhY2Vib29rLmNvbYINbWVzc2VuZ2VyLmNvbTA+BgNVHSAENzA1MDMGBmeBDAEC
AjApMCcGCCsGAQUFBwIBFhtodHRwOi8vd3d3LmRpZ2ljZXJ0LmNvbS9DUFMwDgYD
VR0PAQH/BAQDAgOIMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjB1BgNV
HR8EbjBsMDSgMqAwhi5odHRwOi8vY3JsMy5kaWdpY2VydC5jb20vc2hhMi1oYS1z
ZXJ2ZXItZzYuY3JsMDSgMqAwhi5odHRwOi8vY3JsNC5kaWdpY2VydC5jb20vc2hh
Mi1oYS1zZXJ2ZXItZzYuY3JsMIGDBggrBgEFBQcBAQR3MHUwJAYIKwYBBQUHMAGG
GGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTBNBggrBgEFBQcwAoZBaHR0cDovL2Nh
Y2VydHMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0U0hBMkhpZ2hBc3N1cmFuY2VTZXJ2
ZXJDQS5jcnQwDAYDVR0TAQH/BAIwADCCAYAGCisGAQQB1nkCBAIEggFwBIIBbAFq
AHcA7s3QZNXbGs7FXLedtM0TojKHRny87N7DUUhZRnEftZsAAAGRL11PxgAABAMA
SDBGAiEAlOAB9893qqDZbU1Jzu29ZOajU4oXtnpiXIhj9Ks7524CIQD0mbFhUhaM
7rEJa3QdfBStGd4ihYsqKmIKrIBpHUaSnQB3ANq2v2s/tbYin5vCu1xr6HCRcWy7
UYSFNL2kPTBI1/urAAABkS9dTyQAAAQDAEgwRgIhAMhbcmHxCfTbKUB7n2AwUejJ
P1Iwe2Iw7XDVTXEyc/cwAiEAmYY+eF8a55nZ8kkTWpoWMh9JuI+G/bA53yCJzHiE
McIAdgA/F0tP1yJHWJQdZRyEvg0S7ZA3fx+FauvBvyiF7PhkbgAAAZEvXU+7AAAE
AwBHMEUCIFNb+BQzZkjpR9dwc8ZpvJhinq5XRrOLMDVi2b04Joh8AiEAtFZvyEyZ
e9/t0nNdd22QPDL0ClBqRY66iQlP+HMNhhIwDQYJKoZIhvcNAQELBQADggEBAFIC
/R/rKbZOvBDdbvsUvvPx/8pKVGuKxAGikgUqYZySh4iyudh582mvP0lIUNgBmbYX
A4iJz7dk0PLShWWjCDw5xkhGZ5bXpXqJzrKMFV2r44ZjzyXnHbUOu7uniUr7cmNf
gB8sQIIWjYBOOSGmCXNi4DGbGB1zMR69c3UsjqHnSbL3jWg2cyyLgHHwIcCKfBPP
PVrh27ABFzNH+7mIVVDhiv/wD6jXrNe7noQc1tDVaHb6YRRJInA2veVmEoCmTAcb
b2Kao2DC8YccoPgJmYTjIORY1C5G3Va1IG6YYQC9zF0DlAGF3I+21IsIxSqJcl9m
Di5yq7gXjhXCrqZdka4=
-----END CERTIFICATE-----

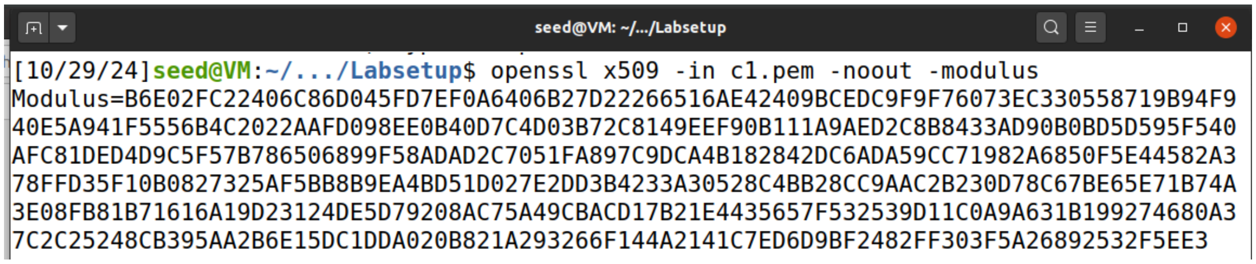
C1.perm

-----BEGIN CERTIFICATE-----
MIIEsTCCA5mgAwIBAgIQBOHnpNxc8vNtwCtCuF0VnzANBgkqhkiG9w0BAQsFADBs
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSswKQYDVQQDEyJEaWdpQ2VydCBIaWdoIEFzc3VyYW5j
ZSBFViBSb290IENBMB4XDTEzMTAyMjEyMDAwMFoXDTI4MTAyMjEyMDAwMFowcDEL
MAkGA1UEBhMCVVMxFTATBgNVBAoTDERpZ2lDZXJ0IEluYzEZMBcGA1UECxMQd3d3
LmRpZ2ljZXJ0LmNvbTEvMC0GA1UEAxMmRGlnaUNlcnQgU0hBMiBIaWdoIEFzc3Vy
YW5jZSBTZXJ2ZXIgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC2
4C/CJAbIbQRf1+8KZAayfSImZRauQkCbztyfn3YHPsMwVYcZuU+UDlqUH1VWtMIC
Kq/QmO4LQNfE0DtyyBSe75CxEamu0si4QzrZCwvV1ZX1QK/IHe1NnF9Xt4ZQaJn1
itrSxwUfqJfJ3KSxgoQtxq2lnMcZgqaFD15EWCo3j/018QsIJzJa9buLnqS9UdAn
4t07QjOjBSjEuyjMmqwrIw14xnvmXnG3Sj4I+4G3FhahnSMSTeXXkgisdaScus0X
sh5ENWV/UyU50RwKmmMbGZJ0aAo3wsJSSMs5WqK24V3B3aAguCGikyZvFEohQcft
bZvySC/zA/WiaJJTL17jAgMBAAGjggFJMIIBRTASBgNVHRMBAf8ECDAGAQH/AgEA
MA4GA1UdDwEB/wQEAwIBhjAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIw
NAYIKwYBBQUHAQEEKDAmMCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5kaWdpY2Vy

dC5jb20wSwYDVR0fBEQwQjBAoD6gPIY6aHR0cDovL2NybDQuZGlnaWNlcnQuY29t
L0RpZ2lDZXJ0SGlnaEFzc3VyYW5jZUVWUm9vdENBLmNybDA9BgNVHSAENjA0MDIG
BFUdIAAwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cuZGlnaWNlcnQuY29tL0NQ
UzAdBgNVHQ4EFgQUUWj/kK8CB3U8zNllZGKiErhZcjswHwYDVR0jBBgwFoAUsT7D
aQP4v0cB1JgmGggC72NkK8MwDQYJKoZIhvcNAQELBQADggEBABiKlYkD5m3fXPwd
aOpKj4PWUS+Na0QWnqxj9dJubISZi6qBcYRb7TROsLd5kinMLYBq8I4g4Xmk/gNH
E+r1hspZcX30BJZr01lYPf7TMSVcGDiEo+afgv2MW5gxTs14nhr9hctJqvIni5ly
/D6q1UEL2tU2ob8cbkdJf17ZSHwD2f2LSaCYJkJA69aSEaRkCldUxPUd1gJea6zu
xICaEnL6VpPX/78whQYwvwt/Tv9XBZ0k7YXDK/umdaisLRbvfXknsuvCnQsH6qqF
0wGjIChBWUMo0oHjqvbsezt3tkBigAVBRQHvFwY+3sAzm2fTYS5yh+Rp/BIAV0Ae
cPUeybQ=
-----END CERTIFICATE-----

Step 2: Extract the public key (e, n) from the issuer’s certificate.

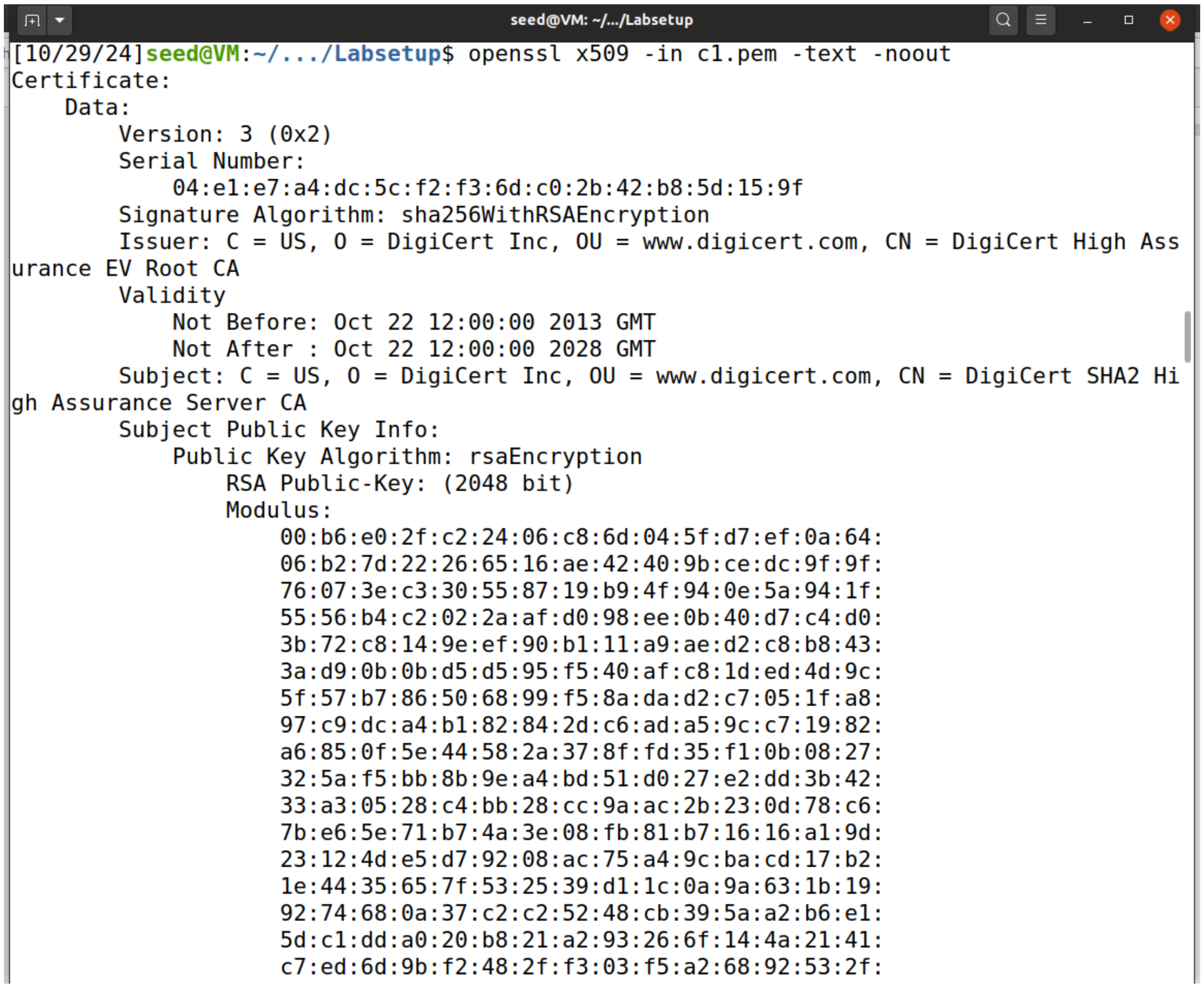
For modulus (n): $ openssl x509 -in c1.pem -noout -modulus

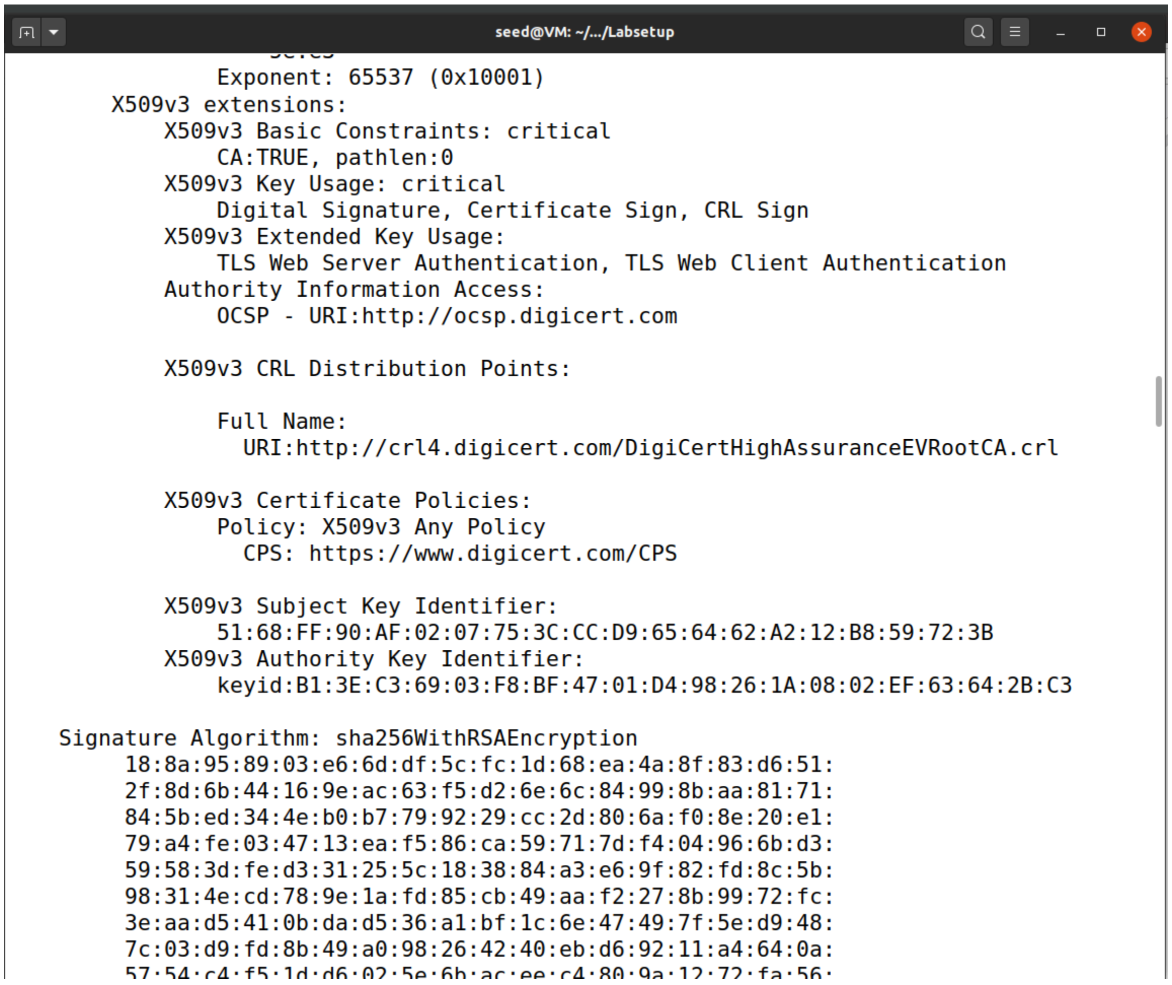


```
[10/29/24]seed@VM:~/.../Labsetup$ openssl x509 -in c1.pem -noout -modulus
Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330558719B94F9
40E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90B0BD5D595F540
AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC71982A6850F5E44582A3
78FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528C4BB28CC9AAC2B230D78C67BE65E71B74A
3E08FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E4435657F532539D11C0A9A631B199274680A3
7C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF2482FF303F5A26892532F5EE3
```

Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073
EC330558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B1
11A9AED2C8B8433AD90B0BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C705
1FA897C9DCA4B182842DC6ADA59CC71982A6850F5E44582A378FFD35F10B0827325AF5B
B8B9EA4BD51D027E2DD3B4233A30528C4BB28CC9AAC2B230D78C67BE65E71B74A3E08
FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E4435657F532539D11C0A9A631
B199274680A37C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2141C7ED6D9
BF2482FF303F5A26892532F5EE3

For exponent (e): $ openssl x509 -in c1.pem -text-noout

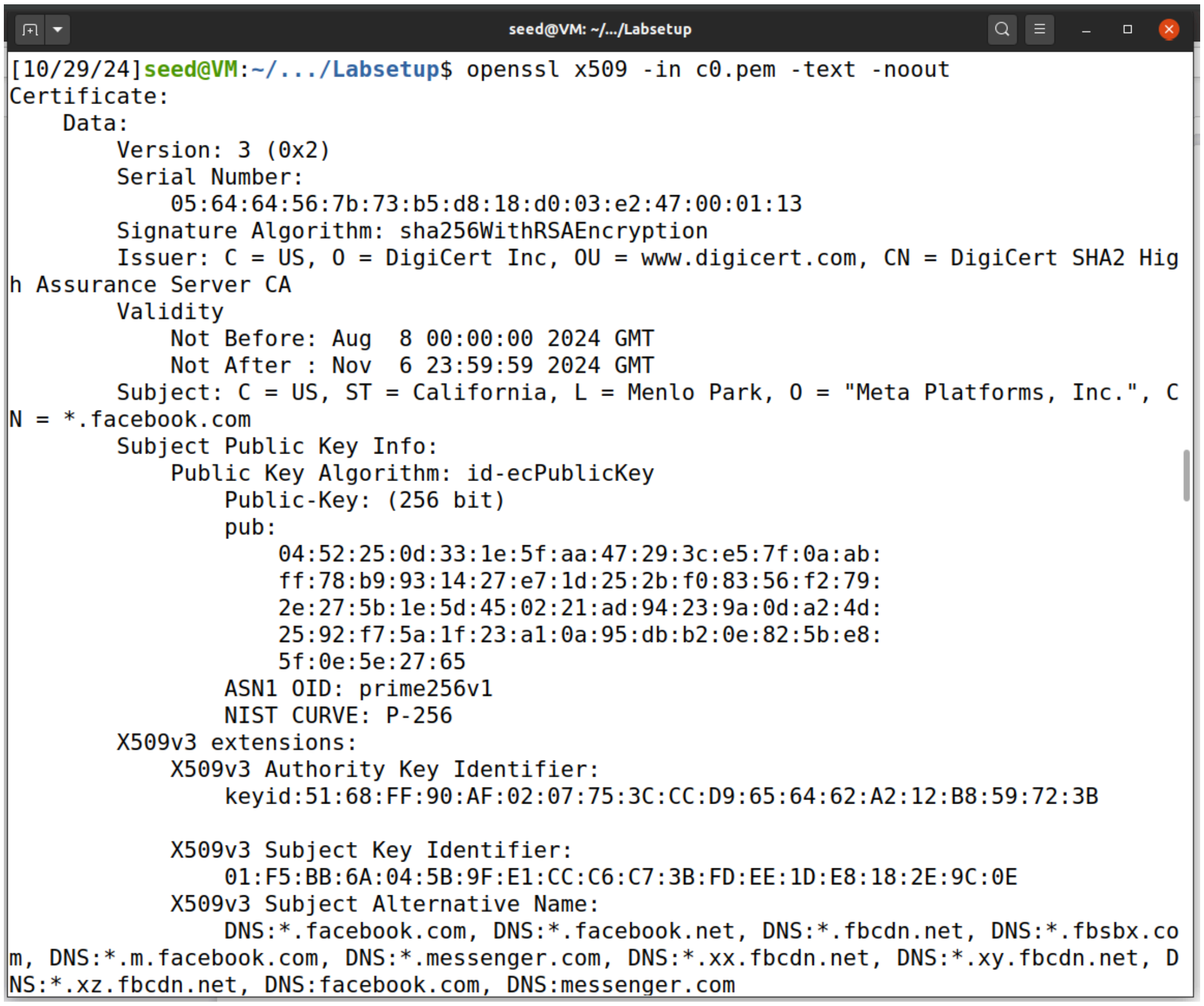

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl x509 -in c1.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:e1:e7:a4:dc:5c:f2:f3:6d:c0:2b:42:b8:5d:15:9f
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Ass
urance EV Root CA
        Validity
            Not Before: Oct 22 12:00:00 2013 GMT
            Not After : Oct 22 12:00:00 2028 GMT
        Subject: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 Hi
gh Assurance Server CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:b6:e0:2f:c2:24:06:c8:6d:04:5f:d7:ef:0a:64:
                    06:b2:7d:22:26:65:16:ae:42:40:9b:ce:dc:9f:9f:
                    76:07:3e:c3:30:55:87:19:b9:4f:94:0e:5a:94:1f:
                    55:56:b4:c2:02:2a:af:d0:98:ee:0b:40:d7:c4:d0:
                    3b:72:c8:14:9e:ef:90:b1:11:a9:ae:d2:c8:b8:43:
                    3a:d9:0b:0b:d5:d5:95:f5:40:af:c8:1d:ed:4d:9c:
                    5f:57:b7:86:50:68:99:f5:8a:da:d2:c7:05:1f:a8:
                    97:c9:dc:a4:b1:82:84:2d:c6:ad:a5:9c:c7:19:82:
                    a6:85:0f:5e:44:58:2a:37:8f:fd:35:f1:0b:08:27:
                    32:5a:f5:bb:8b:9e:a4:bd:51:d0:27:e2:dd:3b:42:
                    33:a3:05:28:c4:bb:28:cc:9a:ac:2b:23:0d:78:c6:
                    7b:e6:5e:71:b7:4a:3e:08:fb:81:b7:16:16:a1:9d:
                    23:12:4d:e5:d7:92:08:ac:75:a4:9c:ba:cd:17:b2:
                    1e:44:35:65:7f:53:25:39:d1:1c:0a:9a:63:1b:19:
                    92:74:68:0a:37:c2:c2:52:48:cb:39:5a:a2:b6:e1:
                    5d:c1:dd:a0:20:b8:21:a2:93:26:6f:14:4a:21:41:
                    c7:ed:6d:9b:f2:48:2f:f3:03:f5:a2:68:92:53:2f:
```



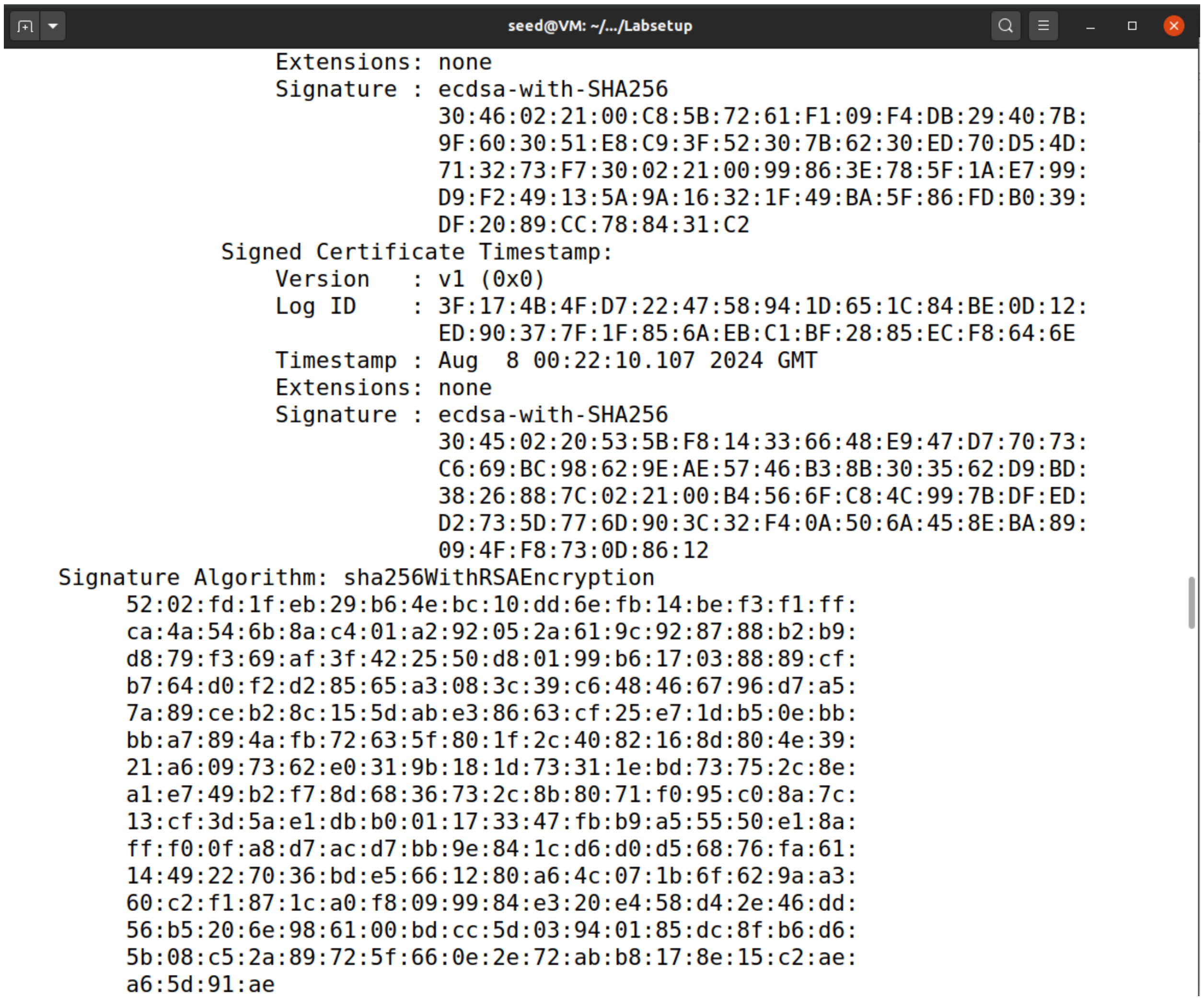
```
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            Authority Information Access:
                OCSP - URI:http://ocsp.digicert.com

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl4.digicert.com/DigiCertHighAssuranceEVRootCA.crl

            X509v3 Certificate Policies:
                Policy: X509v3 Any Policy
                  CPS: https://www.digicert.com/CPS

            X509v3 Subject Key Identifier:
                51:68:FF:90:AF:02:07:75:3C:CC:D9:65:64:62:A2:12:B8:59:72:3B
            X509v3 Authority Key Identifier:
                keyid:B1:3E:C3:69:03:F8:BF:47:01:D4:98:26:1A:08:02:EF:63:64:2B:C3

    Signature Algorithm: sha256WithRSAEncryption
         18:8a:95:89:03:e6:6d:df:5c:fc:1d:68:ea:4a:8f:83:d6:51:
         2f:8d:6b:44:16:9e:ac:63:f5:d2:6e:6c:84:99:8b:aa:81:71:
         84:5b:ed:34:4e:b0:b7:79:92:29:cc:2d:80:6a:f0:8e:20:e1:
         79:a4:fe:03:47:13:ea:f5:86:ca:59:71:7d:f4:04:96:6b:d3:
         59:58:3d:fe:d3:31:25:5c:18:38:84:a3:e6:9f:82:fd:8c:5b:
         98:31:4e:cd:78:9e:1a:fd:85:cb:49:aa:f2:27:8b:99:72:fc:
         3e:aa:d5:41:0b:da:d5:36:a1:bf:1c:6e:47:49:7f:5e:d9:48:
         7c:03:d9:fd:8b:49:a0:98:26:42:40:eb:d6:92:11:a4:64:0a:
         57:54:c4:f5:1d:d6:02:5e:6b:ac:ee:c4:80:9a:12:72:fa:56:
```

For finding the exponent e, we print out all of the fields, in which we found:
Exponent: 65537 (0x10001)

Step 3: Extract the signature from the server's certificate.

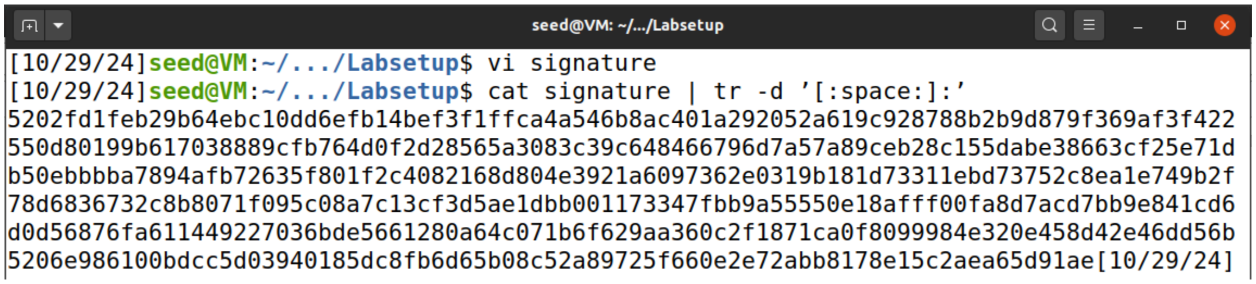

seed@VM: ~/.../Labsetup

```
[10/29/24]seed@VM:~/.../Labsetup$ openssl x509 -in c0.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            05:64:64:56:7b:73:b5:d8:18:d0:03:e2:47:00:01:13
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 Hig
h Assurance Server CA
        Validity
            Not Before: Aug  8 00:00:00 2024 GMT
            Not After : Nov  6 23:59:59 2024 GMT
        Subject: C = US, ST = California, L = Menlo Park, O = "Meta Platforms, Inc.", C
N = *.facebook.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:52:25:0d:33:1e:5f:aa:47:29:3c:e5:7f:0a:ab:
                    ff:78:b9:93:14:27:e7:1d:25:2b:f0:83:56:f2:79:
                    2e:27:5b:1e:5d:45:02:21:ad:94:23:9a:0d:a2:4d:
                    25:92:f7:5a:1f:23:a1:0a:95:db:b2:0e:82:5b:e8:
                    5f:0e:5e:27:65
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:51:68:FF:90:AF:02:07:75:3C:CC:D9:65:64:62:A2:12:B8:59:72:3B

            X509v3 Subject Key Identifier:
                01:F5:BB:6A:04:5B:9F:E1:CC:C6:C7:3B:FD:EE:1D:E8:18:2E:9C:0E
            X509v3 Subject Alternative Name:
                DNS:*.facebook.com, DNS:*.facebook.net, DNS:*.fbcdn.net, DNS:*.fbsbx.co
m, DNS:*.m.facebook.com, DNS:*.messenger.com, DNS:*.xx.fbcdn.net, DNS:*.xy.fbcdn.net, D
NS:*.xz.fbcdn.net, DNS:facebook.com, DNS:messenger.com
```



```
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:46:02:21:00:C8:5B:72:61:F1:09:F4:DB:29:40:7B:
                                9F:60:30:51:E8:C9:3F:52:30:7B:62:30:ED:70:D5:4D:
                                71:32:73:F7:30:02:21:00:99:86:3E:78:5F:1A:E7:99:
                                D9:F2:49:13:5A:9A:16:32:1F:49:BA:5F:86:FD:B0:39:
                                DF:20:89:CC:78:84:31:C2
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : 3F:17:4B:4F:D7:22:47:58:94:1D:65:1C:84:BE:0D:12:
                                ED:90:37:7F:1F:85:6A:EB:C1:BF:28:85:EC:F8:64:6E
                    Timestamp : Aug  8 00:22:10.107 2024 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:45:02:20:53:5B:F8:14:33:66:48:E9:47:D7:70:73:
                                C6:69:BC:98:62:9E:AE:57:46:B3:8B:30:35:62:D9:BD:
                                38:26:88:7C:02:21:00:B4:56:6F:C8:4C:99:7B:DF:ED:
                                D2:73:5D:77:6D:90:3C:32:F4:0A:50:6A:45:8E:BA:89:
                                09:4F:F8:73:0D:86:12
    Signature Algorithm: sha256WithRSAEncryption
         52:02:fd:1f:eb:29:b6:4e:bc:10:dd:6e:fb:14:be:f3:f1:ff:
         ca:4a:54:6b:8a:c4:01:a2:92:05:2a:61:9c:92:87:88:b2:b9:
         d8:79:f3:69:af:3f:42:25:50:d8:01:99:b6:17:03:88:89:cf:
         b7:64:d0:f2:d2:85:65:a3:08:3c:39:c6:48:46:67:96:d7:a5:
         7a:89:ce:b2:8c:15:5d:ab:e3:86:63:cf:25:e7:1d:b5:0e:bb:
         bb:a7:89:4a:fb:72:63:5f:80:1f:2c:40:82:16:8d:80:4e:39:
         21:a6:09:73:62:e0:31:9b:18:1d:73:31:1e:bd:73:75:2c:8e:
         a1:e7:49:b2:f7:8d:68:36:73:2c:8b:80:71:f0:95:c0:8a:7c:
         13:cf:3d:5a:e1:db:b0:01:17:33:47:fb:b9:a5:55:50:e1:8a:
         ff:f0:0f:a8:d7:ac:d7:bb:9e:84:1c:d6:d0:d5:68:76:fa:61:
         14:49:22:70:36:bd:e5:66:12:80:a6:4c:07:1b:6f:62:9a:a3:
         60:c2:f1:87:1c:a0:f8:09:99:84:e3:20:e4:58:d4:2e:46:dd:
         56:b5:20:6e:98:61:00:bd:cc:5d:03:94:01:85:dc:8f:b6:d6:
         5b:08:c5:2a:89:72:5f:66:0e:2e:72:ab:b8:17:8e:15:c2:ae:
         a6:5d:91:ae
```

We then saved this signature in a file called signature and ran the following command to remove necessary spaces and keywords.

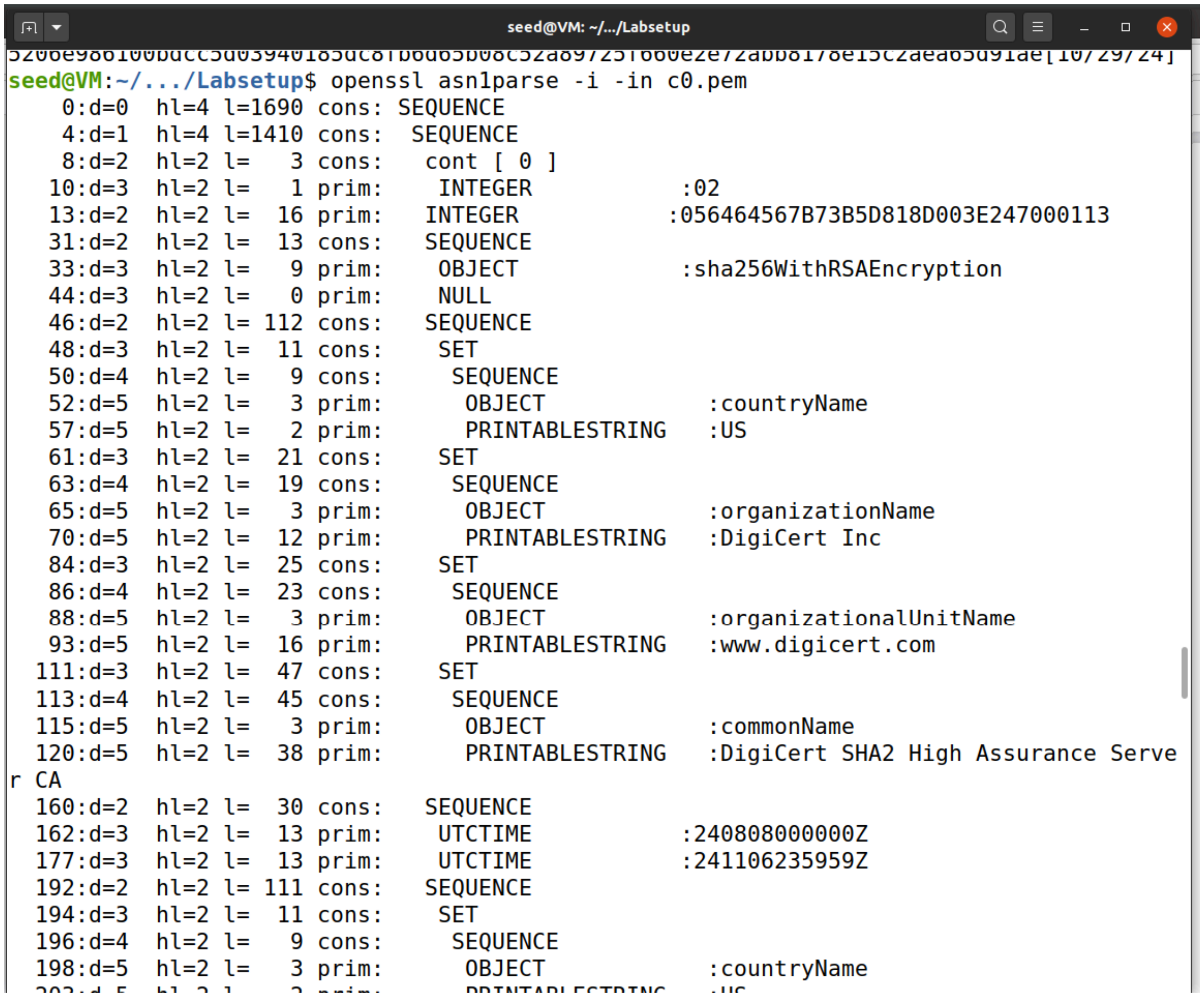


```
[10/29/24]seed@VM:~/.../Labsetup$ vi signature
[10/29/24]seed@VM:~/.../Labsetup$ cat signature | tr -d '[:space:]:'
5202fd1feb29b64ebc10dd6efb14bef3f1ffca4a546b8ac401a292052a619c928788b2b9d879f369af3f422
550d80199b617038889cfb764d0f2d28565a3083c39c648466796d7a57a89ceb28c155dabe38663cf25e71d
b50ebbbba7894afb72635f801f2c4082168d804e3921a6097362e0319b181d73311ebd73752c8ea1e749b2f
78d6836732c8b8071f095c08a7c13cf3d5ae1dbb001173347fbb9a55550e18afff00fa8d7acd7bb9e841cd6
d0d56876fa611449227036bde5661280a64c071b6f629aa360c2f1871ca0f8099984e320e458d42e46dd56b
5206e986100bdcc5d03940185dc8fb6d65b08c52a89725f660e2e72abb8178e15c2aea65d91ae[10/29/24]
```

5202fd1feb29b64ebc10dd6efb14bef3f1ffca4a546b8ac401a292052a619c928788b2b9d879f3 69af3f422550d80199b617038889cfb764d0f2d28565a3083c39c648466796d7a57a89ceb28c1 55dabe38663cf25e71db50ebbbba7894afb72635f801f2c4082168d804e3921a6097362e0319b 181d73311ebd73752c8ea1e749b2f78d6836732c8b8071f095c08a7c13cf3d5ae1dbb0011733

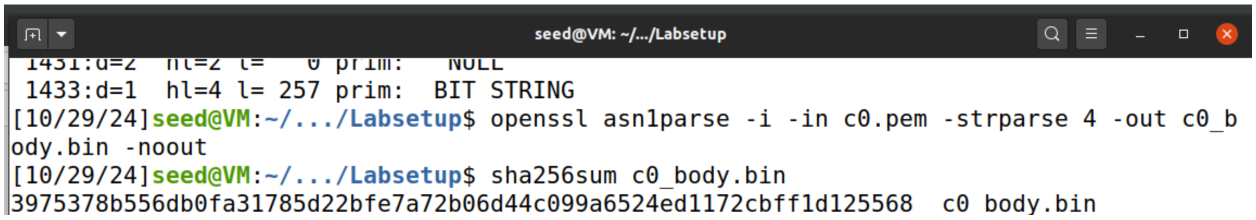
**47fbb9a55550e18afff00fa8d7acd7bb9e841cd6d0d56876fa611449227036bde5661280a64c0
71b6f629aa360c2f1871ca0f8099984e320e458d42e46dd56b5206e986100bdcc5d03940185d
c8fb6d65b08c52a89725f660e2e72abb8178e15c2aea65d91ae**

Step 4: Extract the body of the server's certificate



```
5206e986100bdcc5d03940185dc8fb6d65b08c52a89725f660e2e72abb8178e15c2aea65d91ae[10/29/24]
seed@VM:~/.../Labsetup$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1690 cons: SEQUENCE
    4:d=1  hl=4 l=1410 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           :056464567B73B5D818D003E247000113
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l= 112 cons:   SEQUENCE
   48:d=3  hl=2 l=  11 cons:    SET
   50:d=4  hl=2 l=   9 cons:     SEQUENCE
   52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   61:d=3  hl=2 l=  21 cons:    SET
   63:d=4  hl=2 l=  19 cons:     SEQUENCE
   65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   70:d=5  hl=2 l=  12 prim:      PRINTABLESTRING   :DigiCert Inc
   84:d=3  hl=2 l=  25 cons:    SET
   86:d=4  hl=2 l=  23 cons:     SEQUENCE
   88:d=5  hl=2 l=   3 prim:      OBJECT            :organizationalUnitName
   93:d=5  hl=2 l=  16 prim:      PRINTABLESTRING   :www.digicert.com
  111:d=3  hl=2 l=  47 cons:    SET
  113:d=4  hl=2 l=  45 cons:     SEQUENCE
  115:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  120:d=5  hl=2 l=  38 prim:      PRINTABLESTRING   :DigiCert SHA2 High Assurance Serve
r CA
  160:d=2  hl=2 l=  30 cons:   SEQUENCE
  162:d=3  hl=2 l=  13 prim:    UTCTIME           :240808000000Z
  177:d=3  hl=2 l=  13 prim:    UTCTIME           :241106235959Z
  192:d=2  hl=2 l= 111 cons:   SEQUENCE
  194:d=3  hl=2 l=  11 cons:    SET
  196:d=4  hl=2 l=   9 cons:     SEQUENCE
  198:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
  203:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
```

Just like the example provided in the lab, the certificate body is at offset 4.
We then ran the following command.

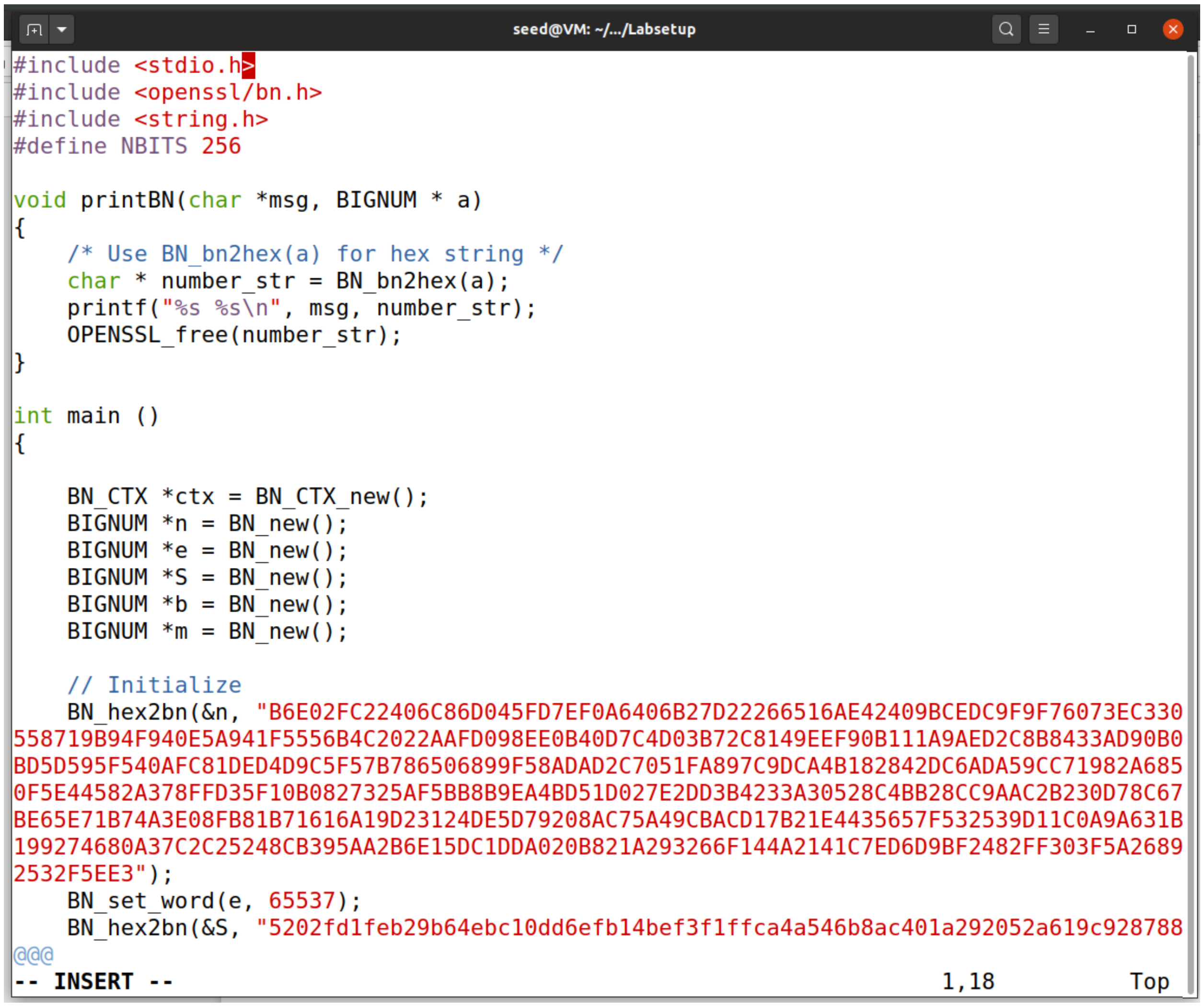


```
 1431:d=2  hl=2 l=   0 prim:   NULL
 1433:d=1  hl=4 l= 257 prim:  BIT STRING
[10/29/24]seed@VM:~/.../Labsetup$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_b
ody.bin -noout
[10/29/24]seed@VM:~/.../Labsetup$ sha256sum c0_body.bin
3975378b556db0fa31785d22bfe7a72b06d44c099a6524ed1172cbff1d125568  c0_body.bin
```

Step 5: Verify the signature

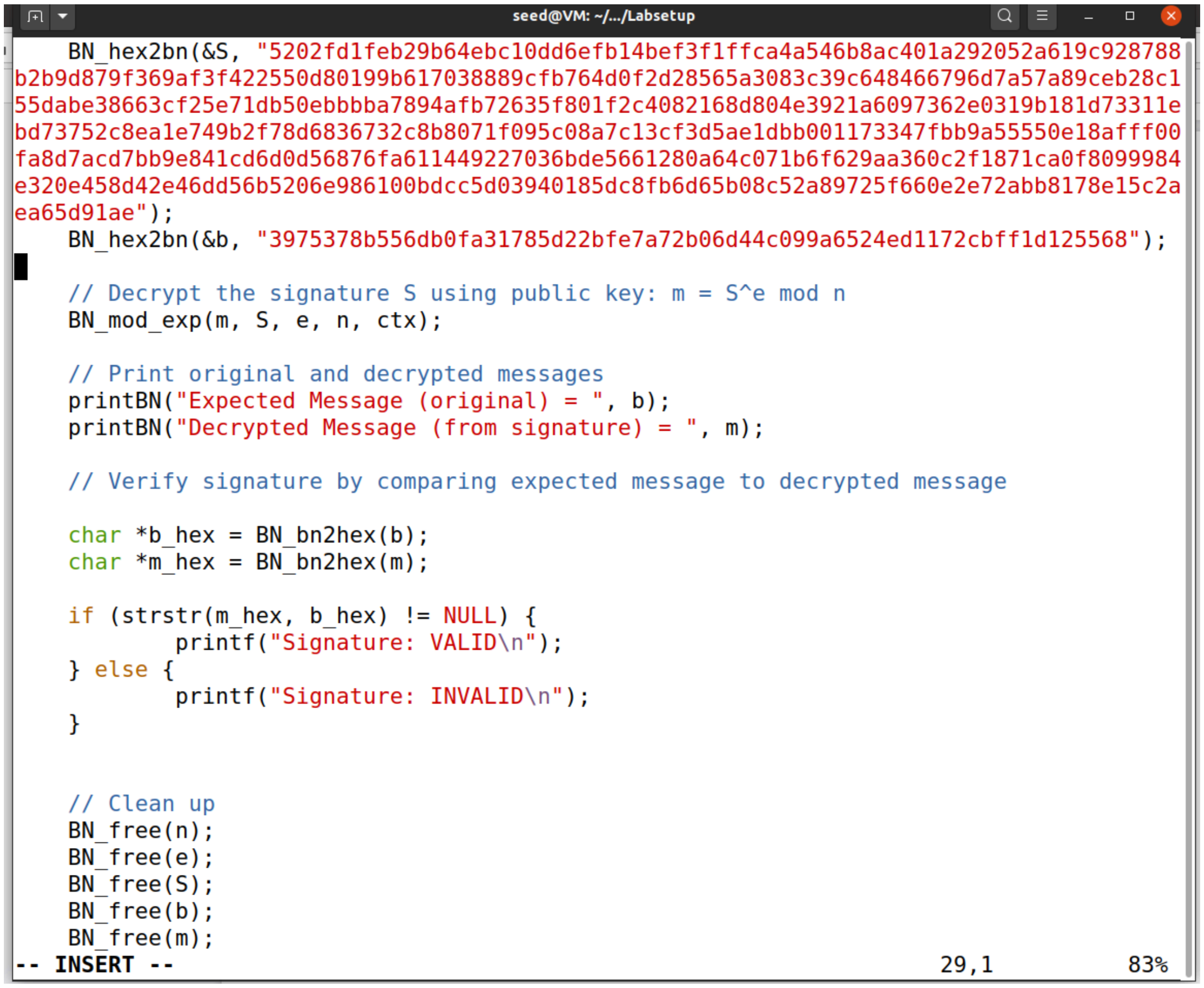
In this step, we modified our task 5 code and changed the values for the public key and signature that we got from the previous steps.

The signature is decrypted using RSA (m = S^e mod n) to obtain the original message. The code then prints both b and m in hexadecimal format and verifies if m matches b using substring matching. This check is done in hexadecimal format due to padding that may appear in the decrypted result, ensuring an accurate comparison.



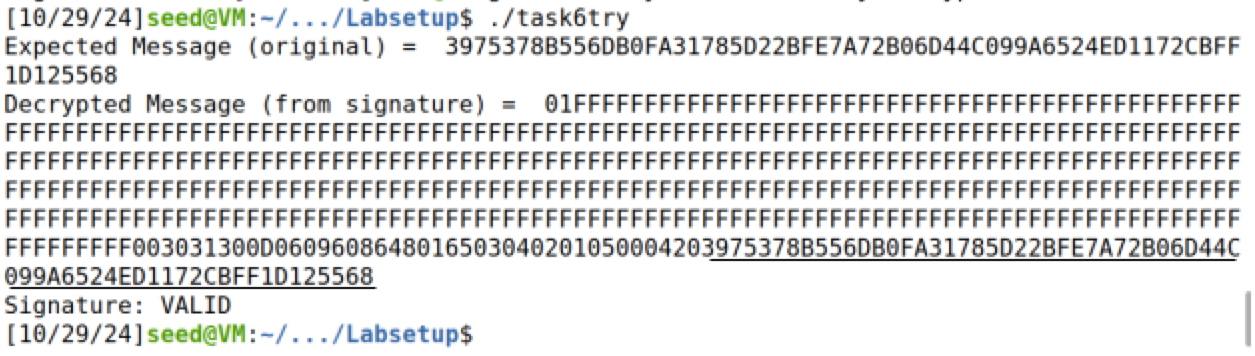
seed@VM: ~/.../Labsetup

```
#include <stdio.h>
#include <openssl/bn.h>
#include <string.h>
#define NBITS 256

void printBN(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 ()
{

    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *S = BN_new();
    BIGNUM *b = BN_new();
    BIGNUM *m = BN_new();

    // Initialize
    BN_hex2bn(&n, "B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330
558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90B0
BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC71982A685
0F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528C4BB28CC9AAC2B230D78C67
BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E4435657F532539D11C0A9A631B
199274680A37C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF2482FF303F5A2689
2532F5EE3");
    BN_set_word(e, 65537);
    BN_hex2bn(&S, "5202fd1feb29b64ebc10dd6efb14bef3f1ffca4a546b8ac401a292052a619c928788
@@@
-- INSERT --                                                    1,18          Top
```



```
    BN_hex2bn(&S, "5202fd1feb29b64ebc10dd6efb14bef3f1ffca4a546b8ac401a292052a619c928788
b2b9d879f369af3f422550d80199b617038889cfb764d0f2d28565a3083c39c648466796d7a57a89ceb28c1
55dabe38663cf25e71db50ebbbba7894afb72635f801f2c4082168d804e3921a6097362e0319b181d73311e
bd73752c8ea1e749b2f78d6836732c8b8071f095c08a7c13cf3d5ae1dbb001173347fbb9a55550e18afff00
fa8d7acd7bb9e841cd6d0d56876fa611449227036bde5661280a64c071b6f629aa360c2f1871ca0f8099984
e320e458d42e46dd56b5206e986100bdcc5d03940185dc8fb6d65b08c52a89725f660e2e72abb8178e15c2a
ea65d91ae");
    BN_hex2bn(&b, "3975378b556db0fa31785d22bfe7a72b06d44c099a6524ed1172cbff1d125568");

    // Decrypt the signature S using public key: m = S^e mod n
    BN_mod_exp(m, S, e, n, ctx);

    // Print original and decrypted messages
    printBN("Expected Message (original) = ", b);
    printBN("Decrypted Message (from signature) = ", m);

    // Verify signature by comparing expected message to decrypted message

    char *b_hex = BN_bn2hex(b);
    char *m_hex = BN_bn2hex(m);

    if (strstr(m_hex, b_hex) != NULL) {
            printf("Signature: VALID\n");
    } else {
            printf("Signature: INVALID\n");
    }


    // Clean up
    BN_free(n);
    BN_free(e);
    BN_free(S);
    BN_free(b);
    BN_free(m);
-- INSERT --                                                          29,1          83%
```

After we run the task we get that it is valid. This means that *www.facebook.org*'s certificate is verified to be valid.



```
[10/29/24]seed@VM:~/.../Labsetup$ ./task6try
Expected Message (original) =  3975378B556DB0FA31785D22BFE7A72B06D44C099A6524ED1172CBFF
1D125568
Decrypted Message (from signature) =  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF003031300D0609608648016503040201050004203975378B556DB0FA31785D22BFE7A72B06D44C
099A6524ED1172CBFF1D125568
Signature: VALID
[10/29/24]seed@VM:~/.../Labsetup$
```