

Applied Cryptography Lab-05 Manual

10 October 2022 21:43

Name : Vishwas M
SRN : PES2UG20CS390
SEC: F
DATE: 25/10/2022
Lab: 4

Note: The demo showed in the video for this lab has been performed on Ubuntu 22.04 running on wsl2. So far, there have been no changes noted between the execution of this lab on wsl and execution on the seedlabs Ubuntu 20.04 virtualbox vm. Therefore, despite the differences in environment, all tasks in the lab should run smoothly.

Task 1: A Complete Example of BIGNUM

The program below shows a complete example of BIGNUM. This program uses three BIGNUM variables, a, b, and n; and then compute $a * b$ and $(a * b \bmod n)$

Code

```
#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 *a = BN_new();
    BIGNUM *b = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *res = BN_new();
    // Initialize
    BN_generate_prime_ex(a,NBITS,1,NULL,NULL,NULL);
    BN_dec2bn(&b,"273489463796838501848592769467194369268");
    BN_rand(n,NBITS,0,0);
    // res = a*b
    BN_mul(res,a,b,ctx);
    printBN("a*b=",res);
    // res = a^b mod n
    BN_mod_exp(res,a,b,n,ctx);
    printBN("a^b mod n=",res);
    return 0;
}
```

Commands

```
$ gcc task1.c -o task1 -lcrypto  
$ ./task1
```

```
seed@VM: ~/.../AC_lab4
[10/25/22] seed@VM:~/.../AC_lab4$ gcc task1.c -o task1 -lcrypto
[10/25/22] seed@VM:~/.../AC_lab4$ ./task1
a*b= A89BDCBF0B68420FBD62B5F7954341565927D1A8F4D7B0473D5257EBBB6D841A94E928AA62
94A704117229735C758EEC
a^b mod n= 06258137B39DDF9D86D5A3095747FED01B04AAABB01803812FEB3D02FCBBBA70
[10/25/22] seed@VM:~/.../AC_lab4$
```

We can see the result of multiplication of 2 large numbers in the first row of the output and we can see the modulus of that result by a number 'n' in the second row of the output in the terminal.

Task 2: Deriving the Private Key

The objective of this task is to derive private key. Given are the hexadecimal values of p, q, e, and public key pair (e,n)

p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3

Code

```
#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 *d = BN_new();
    BIGNUM *res1 = BN_new();
```

```

BIGNUM *res2 = BN_new();
BIGNUM *res3 = BN_new();
BIGNUM *one = BN_new();
// Initialize
BN_hex2bn(&p,"F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q,"E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e,"0D88C3");
BN_hex2bn(&one,"1");
BN_sub(res1,p,one);
BN_sub(res2,q,one);
BN_mul(res3,res1,res2,ctx);
BN_mod_inverse(d,e,res3,ctx);
printBN("d=",d);
return 0;
}

```

Commands

```

$ gcc task2.c -o task2 -lcrypto
$ ./task2

```



```

seed@VM: ~/.../AC_lab4
[10/25/22] seed@VM:~/.../AC_lab4$ gcc task2.c -o task2 -lcrypto
[10/25/22] seed@VM:~/.../AC_lab4$ ./task2
d= 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[10/25/22] seed@VM:~/.../AC_lab4$ 

```

The decryption key found is mentioned in the above screenshot by Elgamal method.

First we calculate Euler's totient $(p-1)*(q-1)$

Then we find mod inverse to find the decryption key to find d

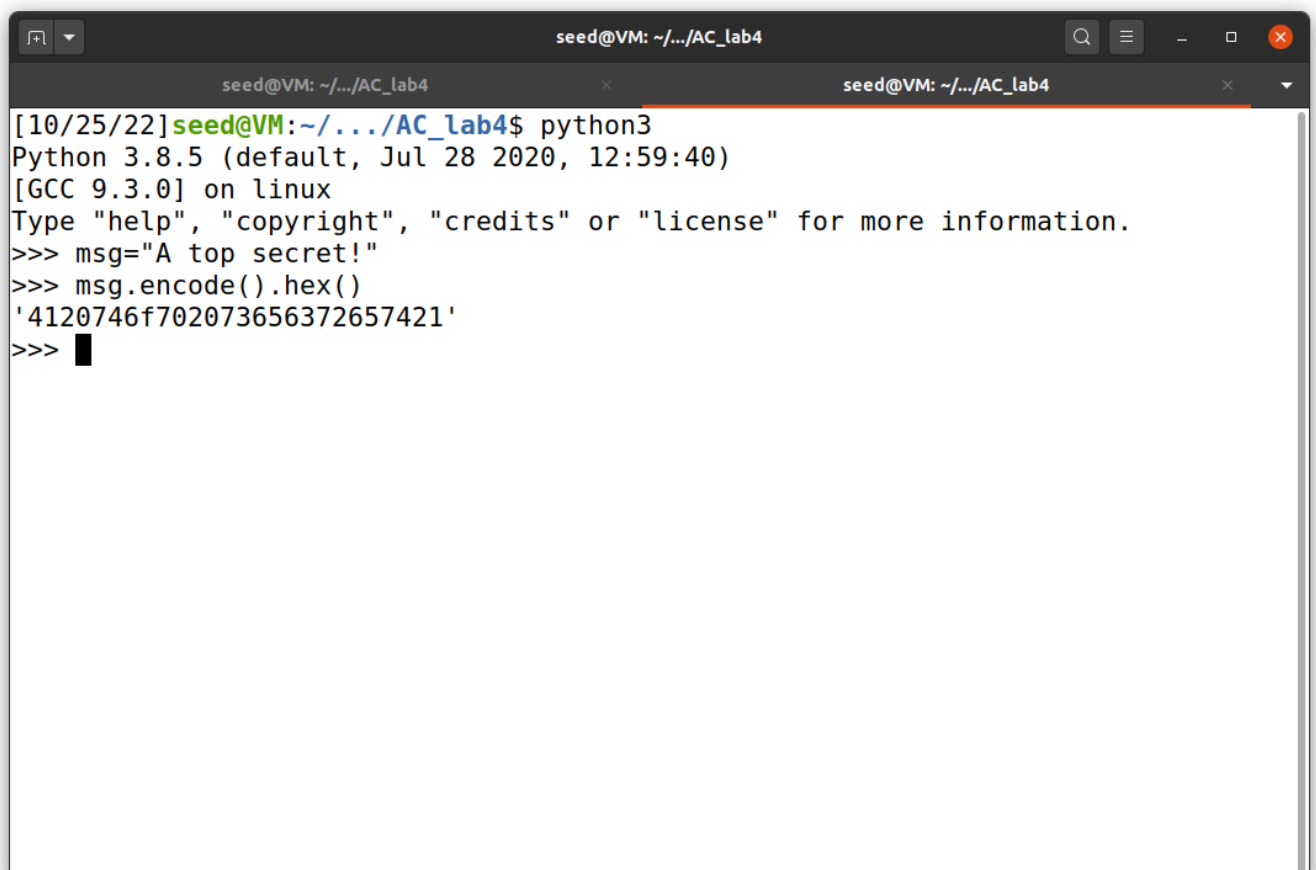
Task 3: Encrypting a Message

The objective of this task is to encrypt a given message. Given are the hexadecimal values of n , e , M (you can use whatever message you want). The value of “ d ” is also given to verify the result.

Step 1

Convert the ASCII String message to a hex string

```
python3 -c "print('A top secret!'.encode().hex())"
```



```
seed@VM: ~/.../AC_lab4
[10/25/22] seed@VM: ~/.../AC_lab4$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> msg="A top secret!"
>>> msg.encode().hex()
'41207466f702073656372657421'
>>>
```

We are finding the hexadecimal form of the text message and inserting it in the code as the text message.

Give your observation with a screenshot

Step 2

Execute the below program to encrypt the message M and verify it by decrypting it.

Code

```

#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 *m = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5;
    e = 010001; // This hex value equal to decimal 65537
    M = "A top secret!";
    d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D;

    // step 1 >>");

    BIGNUM *dec = BN_new();

    // Initialize
    BN_hex2bn(&m,"<< Enter the message in hex, obtained in
    BN_hex2bn(&e,"010001");
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    // Encryption
    BN_mod_exp(enc,m,e,n,ctx);
    printBN("Encrypted Message =",enc);
    // Decryption
    BN_mod_exp(dec,enc,d,n,ctx);
    printBN("Decrypted Message =",dec);
    return 0;

```

```
}
```

Commands

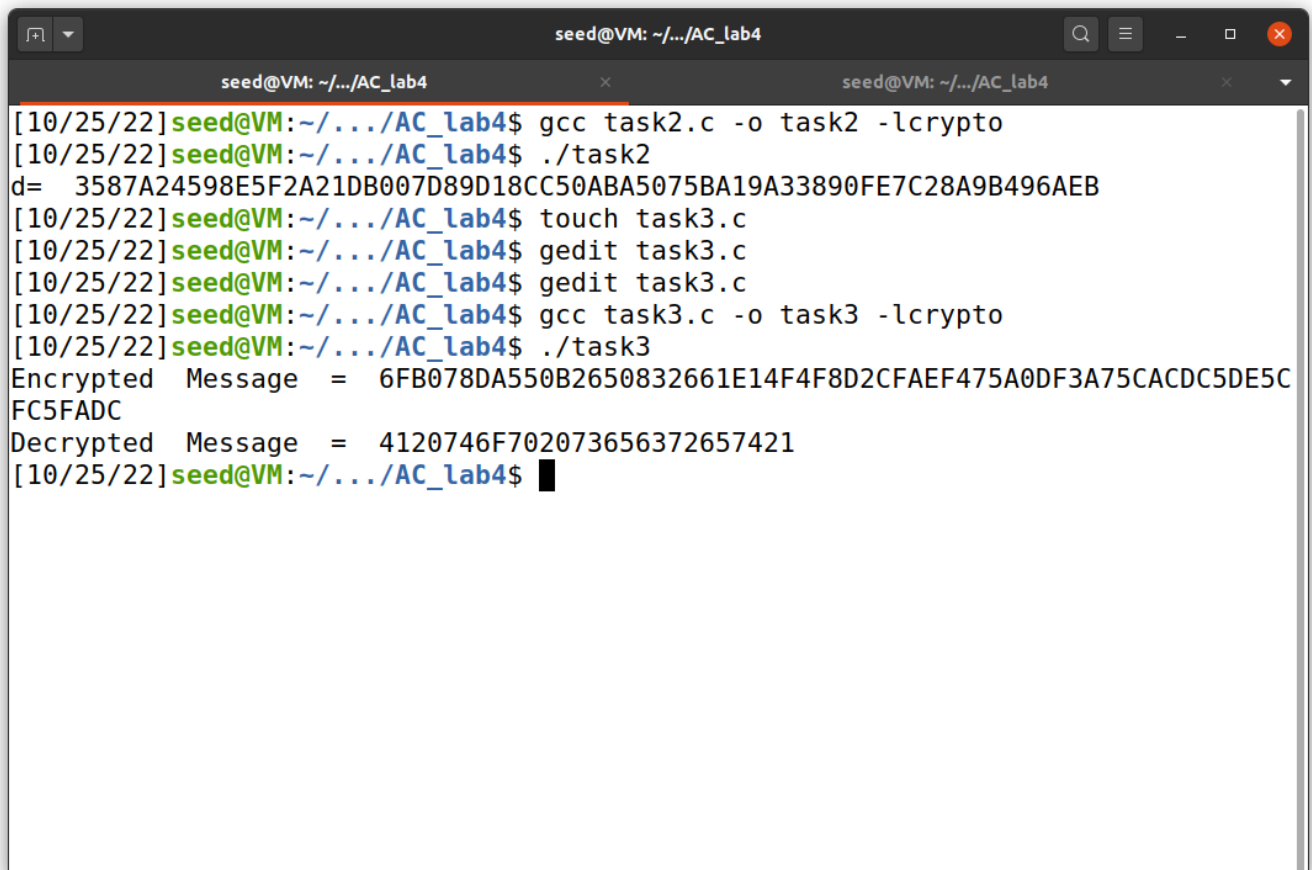
```
$ gcc task3.c -o task3 -lcrypto  
$ ./task3
```

First we are doing

encryption: $enc = m^e \bmod n$

Then we are doing

decryption: $dec = enc^d \bmod n$



```
seed@VM: ~/.../AC_lab4  
[10/25/22] seed@VM: ~/.../AC_lab4$ gcc task2.c -o task2 -lcrypto  
[10/25/22] seed@VM: ~/.../AC_lab4$ ./task2  
d= 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB  
[10/25/22] seed@VM: ~/.../AC_lab4$ touch task3.c  
[10/25/22] seed@VM: ~/.../AC_lab4$ gedit task3.c  
[10/25/22] seed@VM: ~/.../AC_lab4$ gedit task3.c  
[10/25/22] seed@VM: ~/.../AC_lab4$ gcc task3.c -o task3 -lcrypto  
[10/25/22] seed@VM: ~/.../AC_lab4$ ./task3  
Encrypted Message = 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5C  
FC5FADC  
Decrypted Message = 4120746F702073656372657421  
[10/25/22] seed@VM: ~/.../AC_lab4$
```

We are getting the same text message when we decrypt the ciphertext.

Task 4: Decrypting a Message

The objective of this task is to decrypt a given ciphertext. given are the hexadecimal values of n , e , d from the above task

$C =$

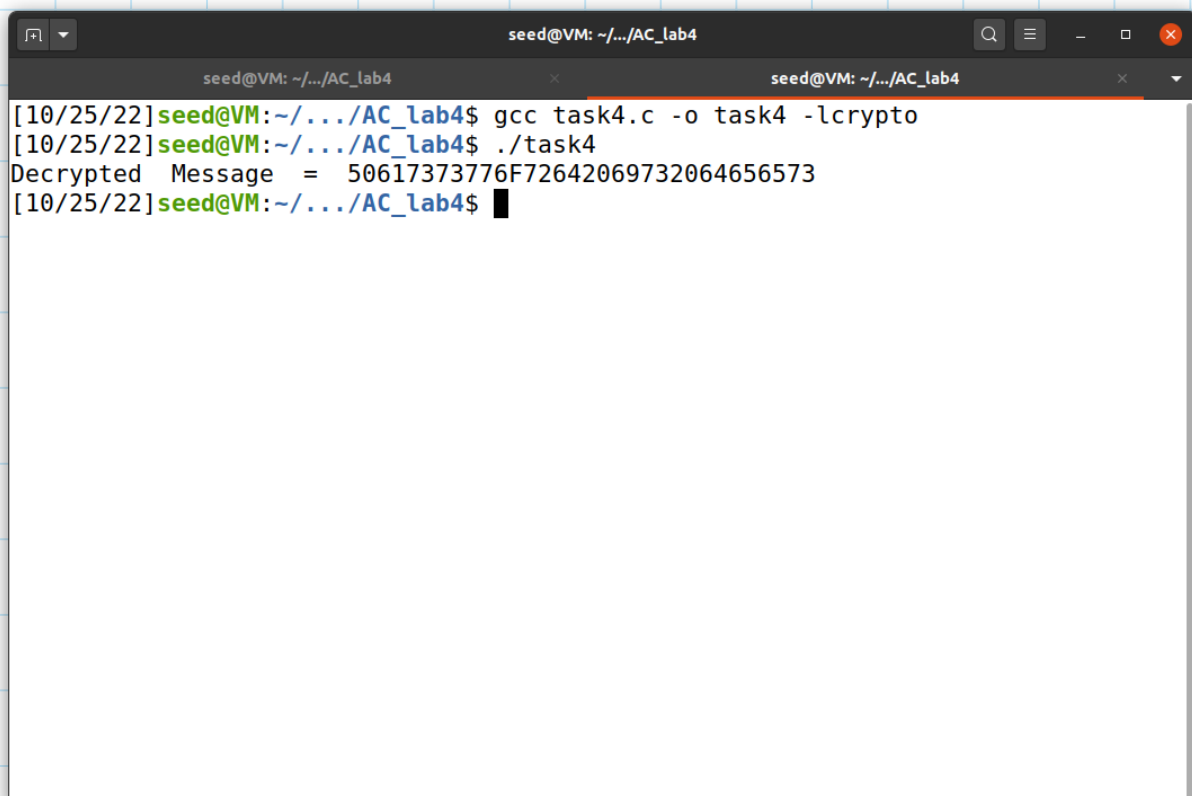
8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBD7C7DCB67396567E
A1E2493F

Code

```
#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 *m = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();
    // Initialize
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&enc,"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F");
    // Decryption
    BN_mod_exp(dec,enc,d,n,ctx);
    printBN("Decrypted Message =",dec);
    return 0;
}
```

Commands

```
$ gcc task4.c -o task4 -lcrypto
$ ./task4
```



The screenshot shows a terminal window with the following commands and output:

```
seed@VM: ~/.../AC_lab4
[10/25/22] seed@VM:~/.../AC_lab4$ gcc task4.c -o task4 -lcrypto
[10/25/22] seed@VM:~/.../AC_lab4$ ./task4
Decrypted Message = 50617373776F72642069732064656573
[10/25/22] seed@VM:~/.../AC_lab4$
```


We are finding the message given the ciphertext and decryption key

```
>>> bytes.fromhex('50617373776F72642069732064656573')
b'Password is dees'
>>> █
```

When we convert the plaintext to bytes we get the above text as the message text.

Task 5: Signing a Message

The objective of this task is to generate a signature for the following message. Use the public/private key set from task3

M= I owe you \$2000

Step 1

Generate hex for M

```
python3 -c "print('I owe you $2000'.encode().hex())"
```

Step 2

Execute the following program to generate signature of the given message. Using the signing algorithm $M^d \bmod n$

Code

```
#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 *m = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *sign = BN_new();
    // Initialize
    BN_hex2bn(&m,"<< Hex value of M >>");
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    // Signing
    BN_mod_exp(sign,m,d,n,ctx);
    printBN("Sign =",sign);
    return 0;
}
```

Commands \$./task5

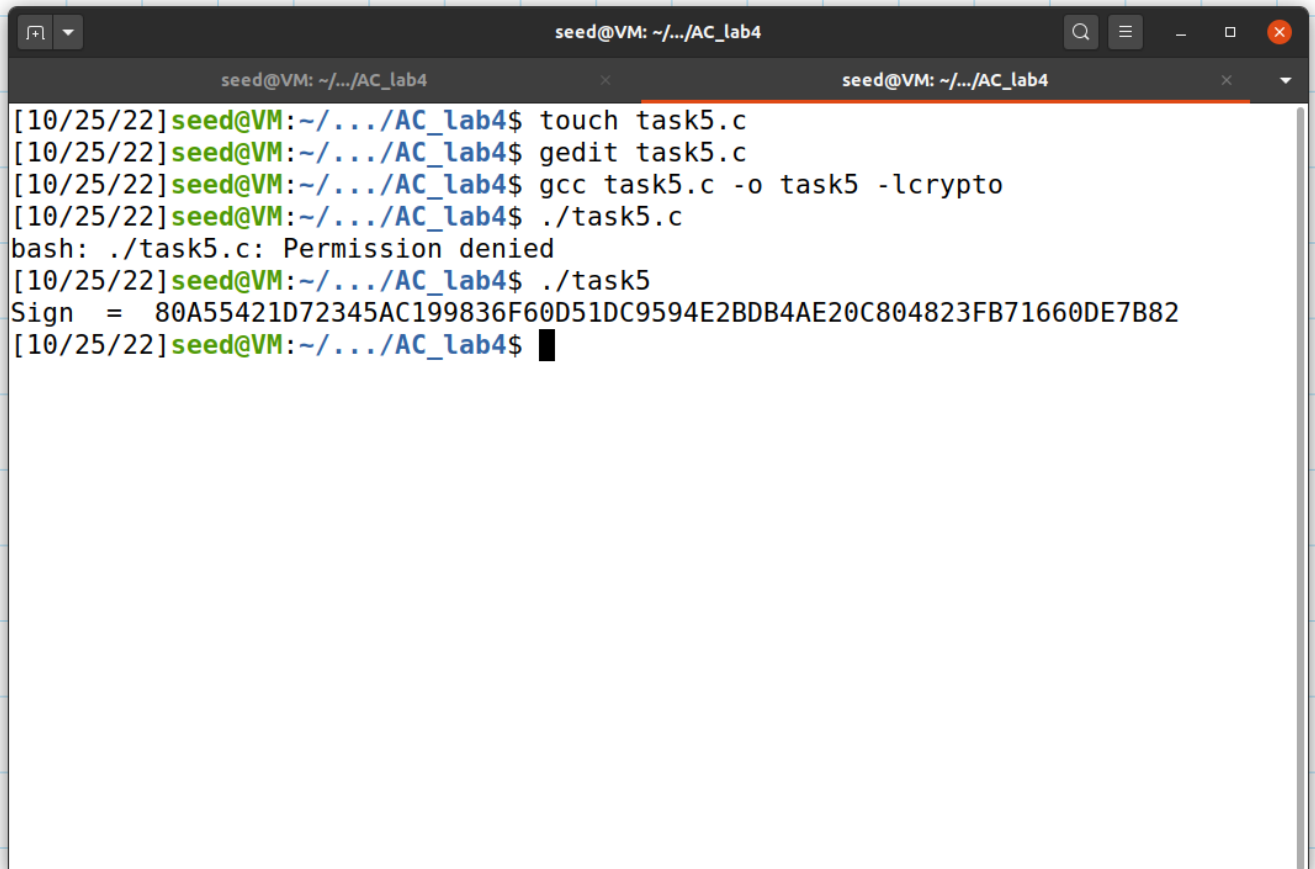
\$ gcc task5.c -o

Step 3

task5 -lcrypto

Execute steps 1 and 2 for the message "I owe \$3000"

```
>>> msg1='I owe you $2000'  
>>> msg1.encode().hex()  
'49206f776520796f75202432303030'  
>>> █
```



```
seed@VM: ~/.../AC_lab4  
[10/25/22] seed@VM: ~/.../AC_lab4$ touch task5.c  
[10/25/22] seed@VM: ~/.../AC_lab4$ gedit task5.c  
[10/25/22] seed@VM: ~/.../AC_lab4$ gcc task5.c -o task5 -lcrypto  
[10/25/22] seed@VM: ~/.../AC_lab4$ ./task5.c  
bash: ./task5.c: Permission denied  
[10/25/22] seed@VM: ~/.../AC_lab4$ ./task5  
Sign = 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82  
[10/25/22] seed@VM: ~/.../AC_lab4$ █
```

We are generating the signature of the given plain text

Sign = $m^d \bmod n$

Task 6: Verifying a signature

The objective of this task is to verify if the signature received by Bob is Alice's or not. Given are the Message M, signature S, Alice public key e and n.

Code

```
#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 *s = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *message = BN_new();
    // Initialize
    BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802
F");
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F1811611
5");
    BN_hex2bn(&e, "010001");
    // Signing
    BN_mod_exp(message,s,e,n,ctx);
    printBN("Message =",message);
    return 0;
}
```

Commands

```
$ gcc task6.c -o task6 -lcrypto
$ ./task6
$ python3 -c "print(bytes.fromhex('<< output of
task6 >>'))"
```

```
seed@VM: ~/.../AC_lab4
[10/25/22] seed@VM:~/.../AC_lab4$ touch task6.c
[10/25/22] seed@VM:~/.../AC_lab4$ gedit task6.c
[10/25/22] seed@VM:~/.../AC_lab4$ gcc task6.c -o task6 -lcrypto
[10/25/22] seed@VM:~/.../AC_lab4$ ./task6
Message = 4C61756E63682061206D697373696C652E
[10/25/22] seed@VM:~/.../AC_lab4$ ^C
[10/25/22] seed@VM:~/.../AC_lab4$ █
```

```
>>> bytes.fromhex('4C61756E63682061206D697373696C652E')
b'Launch a missile.'
>>> █
```

We are getting back the message by using the signature and the by decrypting using public key.

Task 7: Manually Verifying X.509 Certificate

The objective of this task is to verify the signature of a public key certificate from a server and show that the signature matches

To verify that a certificate was signed by a specific certificate authority we need the following details

1. Public key of the certificate authority (issuer).

- signature and algorithm used to generate signature from the server's certificate.

Step 1

Download the certificate from any website (each student use a different website)

```
$ openssl s_client -connect www.example.org:443 -showcerts
```

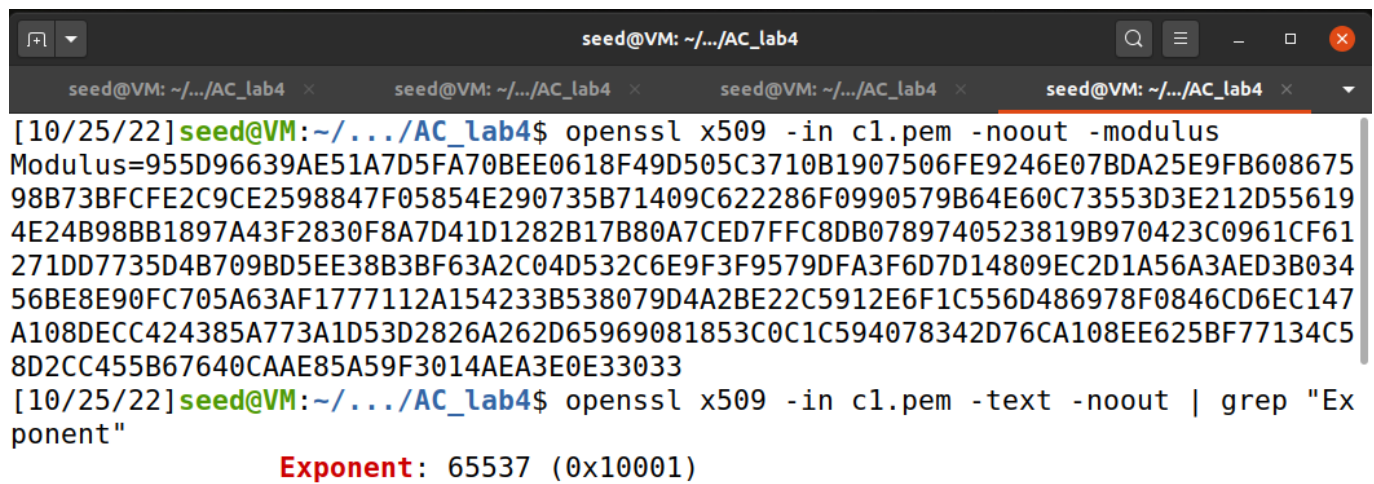
Copy server certificate to c0.pem file and root certificate of the issuer to c1.pem

```

[10/25/22] seed@VM: ~/.../AC_lab4$ openssl s_client -connect www.example.org:443 -showcerts
CONNECTED(00000003)
depth=1 C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = www.example.org
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = www.example.org
  i:C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
-----BEGIN CERTIFICATE-----
MIHRzCCBi+gAwIBAgIQD6pJEMHvD1BSJJkDM1NmjANBgkqhkiG9w0BAQsFADBP
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMSkwJwYDVQQDEyBE
aWdpQ2VydCBUTFMgU1NBIFNIQTl1NiAyMDIwIENBMTAeFw0yMjAzMTQwMDAwMDBa
Fw0yMzAzMTQyMzU5NTlaMIGWMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZv
cm5pYTEuMBIGA1UEBxMLTG9zIEFuZ2VsZXN0QjBBAgNVBAOM0UudGVybmV0wqBD
b3J3bW3JhdGlvb3KgZm9yYwqBBc3NpZ25lZMKGtMftZXPcCoGFuZMKGtNvYtYmVyczEY
MBYGA1UEAxMPd3d3LmV4YW1wbGUub3JnMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
MIIBCgKCAQEALV2Yw5rLGN1fpwvuBhj0nVBcNxCxkHUG/pJG4HvaJen7YIZ1mLc7
/P4sn0JZiEfWFTikHNbcUCcYiKG8JkFebZ0YMc1U9PiEtVWGU4kuYuxiXpD8oMP

```

[illegible]



```
[10/25/22] seed@VM:~/.../AC_lab4$ openssl x509 -in c1.pem -noout -modulus
Modulus=955D96639AE51A7D5FA70BEE0618F49D505C3710B1907506FE9246E07BDA25E9FB608675
98B73BFCFE2C9CE2598847F05854E290735B71409C622286F0990579B64E60C73553D3E212D55619
4E24B98BB1897A43F2830F8A7D41D1282B17B80A7CED7FFC8DB0789740523819B970423C0961CF61
271DD7735D4B709BD5EE38B3BF63A2C04D532C6E9F3F9579DFA3F6D7D14809EC2D1A56A3AED3B034
56BE8E90FC705A63AF1777112A154233B538079D4A2BE22C5912E6F1C556D486978F0846CD6EC147
A108DECC424385A773A1D53D2826A262D65969081853C0C1C594078342D76CA108EE625BF77134C5
8D2CC455B67640CAAEE85A59F3014AEA3E0E33033
[10/25/22] seed@VM:~/.../AC_lab4$ openssl x509 -in c1.pem -text -noout | grep "Exponent"
Exponent: 65537 (0x10001)
```

Give your observation with screenshot

Step 3

Extract the signature from the server's certificate. 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 (note: if the signature algorithm used in the certificate is not based on RSA, find another certificate).

Commands

```
$openssl x509 -in c0.pem -text -noout
//extract only the signature part and paste it in
signature file
$ cat signature | tr -d '[:space:]:'
Give your observation with screenshot.
```

```
seed@VM: ~/.../AC_lab4
```

```
[10/25/22] seed@VM: ~/.../AC_lab4$ openssl x509 -in c0.pem -text -noout
```

```
Certificate:
```

```
Data:
```

```
Version: 3 (0x2)
```

```
Serial Number:
```

```
    06:d8:d9:04:d5:58:43:46:f6:8a:2f:a7:54:22:7e:c4
```

```
Signature Algorithm: sha256WithRSAEncryption
```

```
Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
```

```
Validity
```

```
Not Before: Apr 14 00:00:00 2021 GMT
```

```
Not After : Apr 13 23:59:59 2031 GMT
```

```
Subject: C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
```

```
Subject Public Key Info:
```

```
Public Key Algorithm: rsaEncryption
```

```
RSA Public-Key: (2048 bit)
```

```
Modulus:
```

```
00:c1:4b:b3:65:47:70:bc:dd:4f:58:db:ec:9c:ed:
c3:66:e5:1f:31:13:54:ad:4a:66:46:1f:2c:0a:ec:
64:07:e5:2e:dc:dc:b9:0a:20:ed:df:e3:c4:d0:9e:
9a:a9:7a:1d:82:88:e5:11:56:db:1e:9f:58:c2:51:
e7:2c:34:0d:2e:d2:92:e1:56:cb:f1:79:5f:b3:bb:
87:ca:25:03:7b:9a:52:41:66:10:60:4f:57:13:49:
f0:e8:37:67:83:df:e7:d3:4b:67:4c:22:51:a6:df:
0e:99:10:ed:57:51:74:26:e2:7d:c7:ca:62:2e:13:
1b:7f:23:88:25:53:6f:c1:34:58:00:8b:84:ff:f8:
be:a7:58:49:22:7b:96:ad:a2:88:9b:15:bc:a0:7c:
df:e9:51:a8:d5:b0:ed:37:e2:36:b4:82:4b:62:b5:
49:9a:ec:c7:67:d6:e3:3e:f5:e3:d6:12:5e:44:f1:
bf:71:42:7d:58:84:03:80:b1:81:01:fa:f9:ca:32:
b1:b4:8a:27:87:27:5f:2b:74:41:0a:07:1b:02:
```



```
seed@VM: ~/.../AC_lab4
73:3e:f3:a4
[10/25/22]seed@VM:~/.../AC_lab4$ touch signature.txt
[10/25/22]seed@VM:~/.../AC_lab4$ ^C
[10/25/22]seed@VM:~/.../AC_lab4$ cat signature.txt
80:32:ce:5e:0b:dd:6e:5a:0d:0a:af:e1:d6:84:cb:c0:8e:fa:
85:70:ed:da:5d:b3:0c:f7:2b:75:40:fe:85:0a:fa:f3:31:78:
b7:70:4b:1a:89:58:ba:80:bd:f3:6b:1d:e9:7e:cf:0b:ba:58:
9c:59:d4:90:d3:fd:6c:fd:d0:98:6d:b7:71:82:5b:cf:6d:0b:
5a:09:d0:7b:de:c4:43:d8:2a:a4:de:9e:41:26:5f:bb:8f:99:
cb:dd:ae:e1:a8:6f:9f:87:fe:74:b7:1f:1b:20:ab:b1:4f:c6:
f5:67:5d:5d:9b:3c:e9:ff:69:f7:61:6c:d6:d9:f3:fd:36:c6:
ab:03:88:76:d2:4b:2e:75:86:e3:fc:d8:55:7d:26:c2:11:77:
df:3e:02:b6:7c:f3:ab:7b:7a:86:36:6f:b8:f7:d8:93:71:cf:
86:df:73:30:fa:7b:ab:ed:2a:59:c8:42:84:3b:11:17:1a:52:
f3:c9:0e:14:7d:a2:5b:72:67:ba:71:ed:57:47:66:c5:b8:02:
4a:65:34:5e:8b:d0:2a:3c:20:9c:51:99:4c:e7:52:9e:f7:6b:
11:2b:0d:92:7e:1d:e8:8a:eb:36:16:43:87:ea:2a:63:bf:75:
3f:eb:de:c4:03:bb:0a:3c:f7:30:ef:eb:af:4c:fc:8b:36:10:
73:3e:f3:a4

[10/25/22]seed@VM:~/.../AC_lab4$ cat signature |tr -d '[:space:]:'
cat: signature: No such file or directory
tr: invalid option -- '['
Try 'tr --help' for more information.
[10/25/22]seed@VM:~/.../AC_lab4$ cat signature.txt |tr -d '[:space:]:'
tr: invalid option -- '['
Try 'tr --help' for more information.
[10/25/22]seed@VM:~/.../AC_lab4$ cat signature.txt |tr -d '[:space:]:'
8032ce5e0bdd6e5a0d0aafed1d684cbc08efa8570edda5db30cf72b7540fe850afaf33178b7704b1a8958ba80bdf36b1de97ecf0bba589c59d490d3fd6cfd0986db771825bcf6
d0b5a09d07bdec443d2aa4de9e41265fbb8f99cbddae1a86f9f87fe74b71f1b20abb14fc6f5675d5d9b3ce9ff69f7616cd6d9f3fd36c6ab038876d24b2e7586e3fcd8557d26
c21177df3e02b67cf3ab7b7a86366fb8f7d89371cf86df7330fa7babed2a59c842843b11171a52f3c90e147da25b7267ba71ed574766c5b8024a65345e8bd02a3c209c51994ce
[10/25/22]seed@VM:~/.../AC_lab4$
```

Step 4

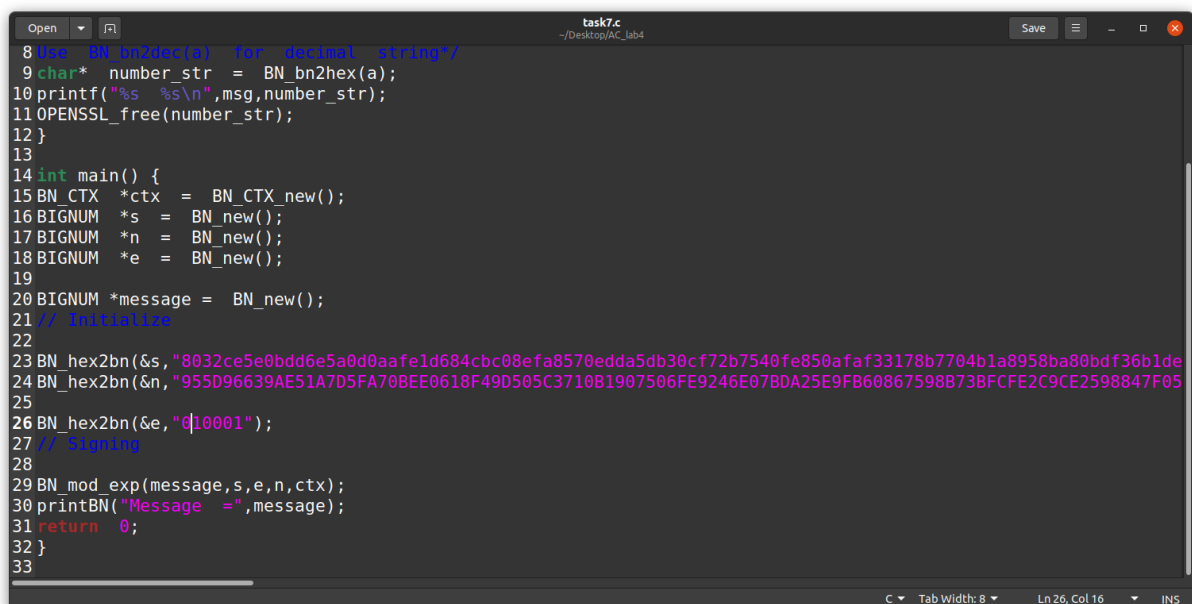
Verify the signature by substituting the values just found out, in the code given below and running it.

Code

```
#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 *s = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *message = BN_new();
    // Initialize
    BN_hex2bn(&s,"<< signature >>");
    BN_hex2bn(&n,"<< modulus >>");
    BN_hex2bn(&e,"<< exponent >>");
    // Signing
    BN_mod_exp(message,s,e,n,ctx);
    printBN("Message =",message);
    return 0;
}
```

Commands

```
$ gcc task7.c -o task7 -lcrypto
$ ./task7
```



```
task7.c
~/Desktop/AC_lab4
8 /* Use BN_bn2dec(a) for decimal string*/
9 char* number_str = BN_bn2hex(a);
10 printf("%s %s\n",msg,number_str);
11 OPENSSL_free(number_str);
12 }
13
14 int main() {
15 BN_CTX *ctx = BN_CTX_new();
16 BIGNUM *s = BN_new();
17 BIGNUM *n = BN_new();
18 BIGNUM *e = BN_new();
19
20 BIGNUM *message = BN_new();
21 // Initialize
22
23 BN_hex2bn(&s,"8032ce5e0bdd6e5a0d0aafe1d684cbc08efa8570edda5db30cf72b7540fe850afaf33178b7704b1a8958ba80bdf36b1de");
24 BN_hex2bn(&n,"955D96639AE51A7D5FA70BEE0618F49D505C3710B1907506FE9246E07BDA25E9FB60867598B73BFCFE2C9CE2598847F05");
25
26 BN_hex2bn(&e,"010001");
27 // Signing
28
29 BN_mod_exp(message,s,e,n,ctx);
30 printBN("Message =",message);
31 return 0;
32 }
33
```

C Tab Width: 8 Ln 26, Col 16 INS

```
seed@VM: ~/.../AC_lab4
seed@VM: ~/.../AC_lab4 x seed@VM: ~/.../AC_lab4 x seed@VM: ~/.../AC_lab4 x seed@VM: ~/.../AC_lab4 x
[10/25/22] seed@VM: ~/.../AC_lab4$ gcc task7.c -o task7 -lcrypto
[10/25/22] seed@VM: ~/.../AC_lab4$ ./task7
Message = 566173E310115890E2F597F20108F7B1639F464A71564392257A22D51C83DEE6167E
BA89DBE4C77162546FBB3A712C2996066931301C2120ACE73400B6A7C4544378BDC5F336F6356EAB
AA2A877E3A249B33D38B1D32FD97B0E902FDA811A370581436A0DFB7785ABB30BDCA8A7142D29FAC
E9F2012A6C89429FC8E22C3D1A783A336E4F1BFB0DE121ECDD346BDE0B9736EBB2CF2187B74C312E
8F8AC68A8C205618E17B07095DE38655CDC19BE2936A5C773E0A478989EB67AA77E88EABF4B9418E
7917F0D9AE37FA2EF8D5CAFBDFFC61F7F575671D4210AA668EDEAD97B86C9286FE995E14EF43DDA
889223D1CED98B5F4CC9390E4D836E1652F1F96DC4FB
[10/25/22] seed@VM: ~/.../AC_lab4$ █
```

We will substitute
modulus,
exponent and
signature and
find the message