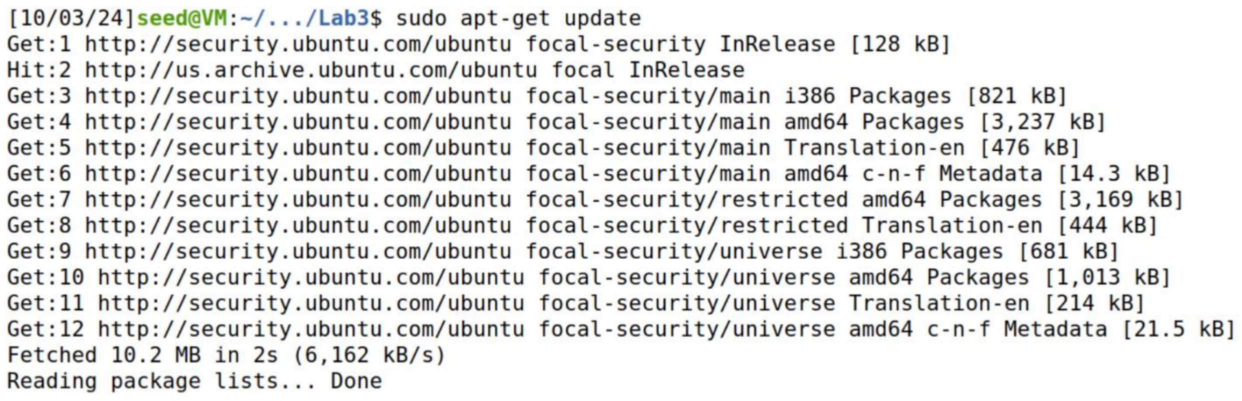
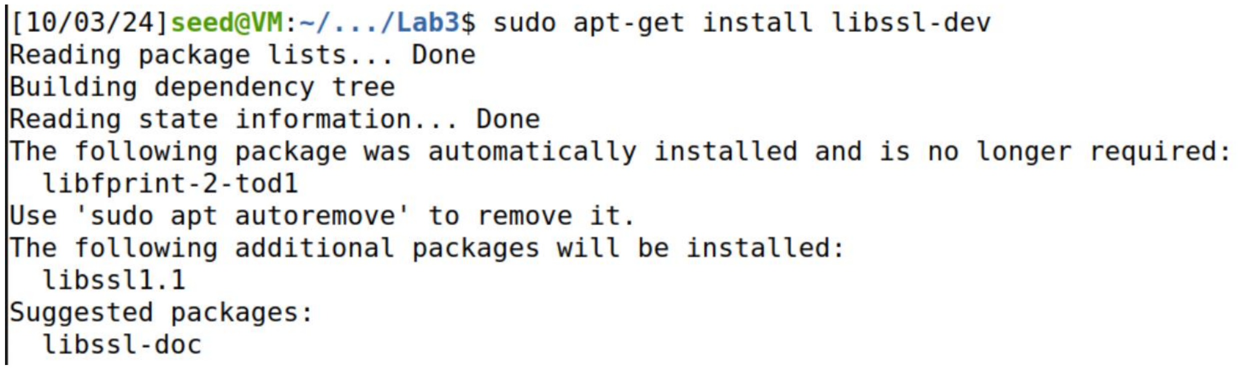
RSA Public-Key Encryption and Signature Lab

Lab setup



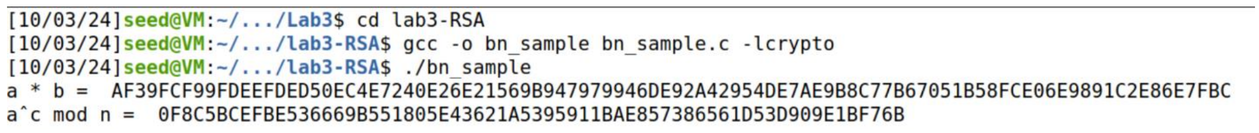
```
[10/03/24]seed@VM:~/.../Lab3$ sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
Hit:2 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:3 http://security.ubuntu.com/ubuntu focal-security/main i386 Packages [821 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3,237 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [476 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [14.3 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3,169 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/restricted Translation-en [444 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/universe i386 Packages [681 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1,013 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/universe Translation-en [214 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-n-f Metadata [21.5 kB]
Fetched 10.2 MB in 2s (6,162 kB/s)
Reading package lists... Done
```



```
[10/03/24]seed@VM:~/.../Lab3$ sudo apt-get install libssl-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libfprint-2-tod1
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libssl1.1
Suggested packages:
  libssl-doc
```

Background

Compilation of bn_sample



```
[10/03/24]seed@VM:~/.../Lab3$ cd lab3-RSA
[10/03/24]seed@VM:~/.../lab3-RSA$ gcc -o bn_sample bn_sample.c -lcrypto
[10/03/24]seed@VM:~/.../lab3-RSA$ ./bn_sample
a * b =  AF39FCF99FDEEFDED50EC4E7240E26E21569B947979946DE92A42954DE7AE9B8C77B67051B58FCE06E9891C2E86E7FBC
a^c mod n =  0F8C5BCEFBE536669B551805E43621A5395911BAE857386561D53D909E1BF76B
```

Task 01: Deriving the Private Key

Let p, q, and e be three prime numbers. Let n = p*q. We will use (e, n) as the public key. Please calculate the private key d. The hexadecimal values of p, q, and e are listed in the following. It should be noted that although p and q are used in this task are quite large numbers, they are not large enough to be secure. We intentionally make them small for the sake of simplicity. In practice, these numbers should be at least

512 bits long (the ones used here is only 128 bits).

p = F7E75FDC469067FFDC4E847C51F452DF

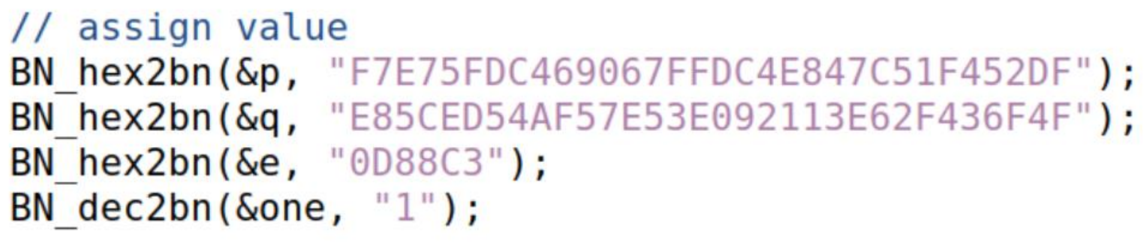
q = E85CED54AF57E53E092113E62F436F4F

e = 0D88C3

```
/* task_1.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);

}

/*
  To find d: ed mod(p-1)(q-1) = 1

*/
int main (){

  BN_CTX *ctx = BN_CTX_new();
```

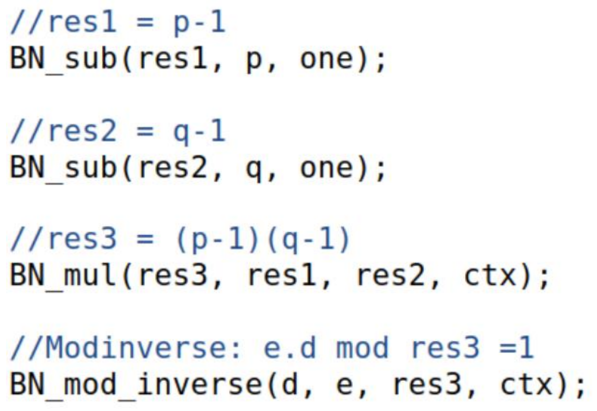
```
    // create variables
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *one = BN_new();
    BIGNUM *res1 = BN_new(); // for p-1
    BIGNUM *res2 = BN_new(); // for q-1
    BIGNUM *res3 = BN_new(); // for (p-1)*(q-1)

    // assign value
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");
    BN_dec2bn(&one, "1");

    //res1 = p-1
    BN_sub(res1, p, one);
    //res2 = q-1
    BN_sub(res2, q, one);
     //res3 = (p-1)(q-1)
    BN_mul(res3, res1, res2, ctx);
     //Modinverse: e.d mod res3 =1
    BN_mod_inverse(d, e, res3, ctx);
    printBN("d = ", d);
    return 0;
// Code must be compiled with '-lcrypto'
// gcc bn_sample.c -lcrypto
}
```

Assign the numbers which is given for p, q, e



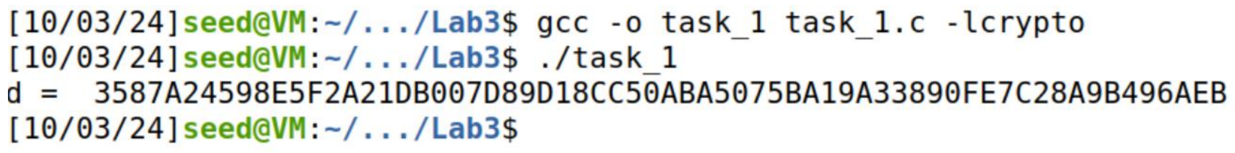
```
// assign value
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3");
BN_dec2bn(&one, "1");
```

Compute functions: res1=p-1, res2=q-1, res3=res1*res2, d*e mod res3=1



```
//res1 = p-1
BN_sub(res1, p, one);

//res2 = q-1
BN_sub(res2, q, one);

//res3 = (p-1)(q-1)
BN_mul(res3, res1, res2, ctx);

//Modinverse: e.d mod res3 =1
BN_mod_inverse(d, e, res3, ctx);
```

There is private key:

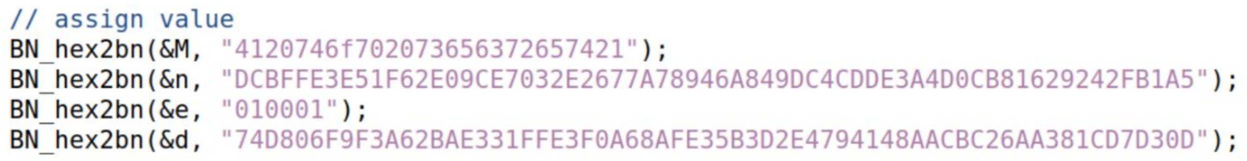


```
[10/03/24]seed@VM:~/.../Lab3$ gcc -o task_1 task_1.c -lcrypto
[10/03/24]seed@VM:~/.../Lab3$ ./task_1
d =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[10/03/24]seed@VM:~/.../Lab3$
```

d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

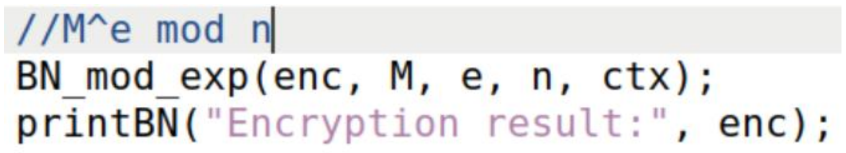
Task 2: Encrypting a Message

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 ().



```
// assign value
BN_hex2bn(&M, "4120746f702073656372657421");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

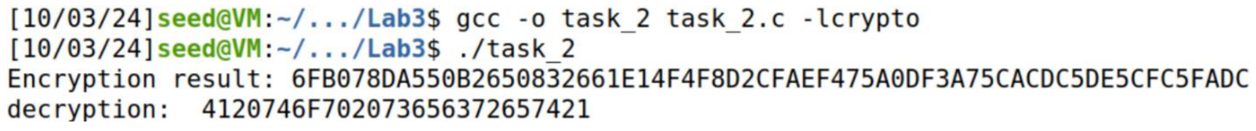
Encrypt the message:



```
//M^e mod n
BN_mod_exp(enc, M, e, n, ctx);
printBN("Encryption result:", enc);
```

Encrypted value:

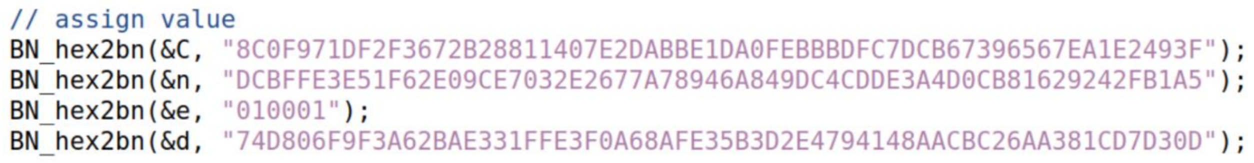
6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC



```
[10/03/24]seed@VM:~/.../Lab3$ gcc -o task_2 task_2.c -lcrypto
[10/03/24]seed@VM:~/.../Lab3$ ./task_2
Encryption result: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
decryption:  4120746F702073656372657421
```

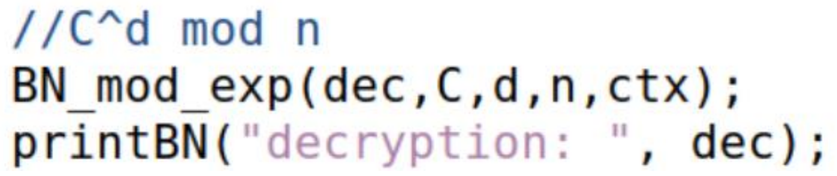
Task 3: Decrypting a Message

First setup the values for C, n, e and d,



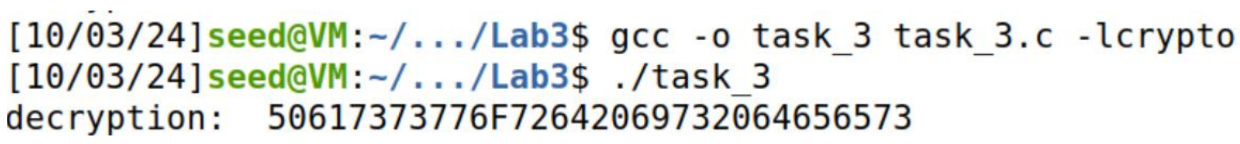
```
// assign value
BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

Then define the decryption:



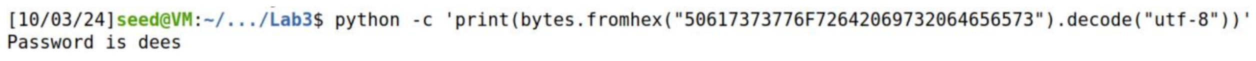
```
//C^d mod n
BN_mod_exp(dec,C,d,n,ctx);
printBN("decryption: ", dec);
```

Decrypted value: 50617373776F72642069732064656573



```
[10/03/24]seed@VM:~/.../Lab3$ gcc -o task_3 task_3.c -lcrypto
[10/03/24]seed@VM:~/.../Lab3$ ./task_3
decryption:  50617373776F72642069732064656573
```

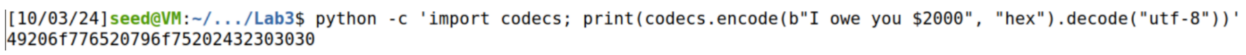
Decoded text: Password is dees



```
[10/03/24]seed@VM:~/.../Lab3$ python -c 'print(bytes.fromhex("50617373776F72642069732064656573").decode("utf-8"))'
Password is dees
```

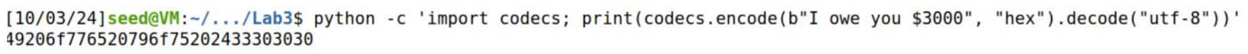
Task 4: Signing a Message

First, get the hex strings of 2 strings:



```
[10/03/24]seed@VM:~/.../Lab3$ python -c 'import codecs; print(codecs.encode(b"I owe you $2000", "hex").decode("utf-8"))'
49206f776520796f75202432303030
```

49206f776520796f75202432303030

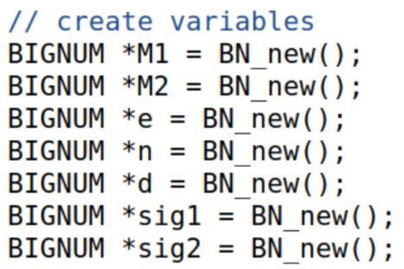


```
[10/03/24]seed@VM:~/.../Lab3$ python -c 'import codecs; print(codecs.encode(b"I owe you $3000", "hex").decode("utf-8"))'
49206f776520796f75202433303030
```

49206f776520796f75202433303030

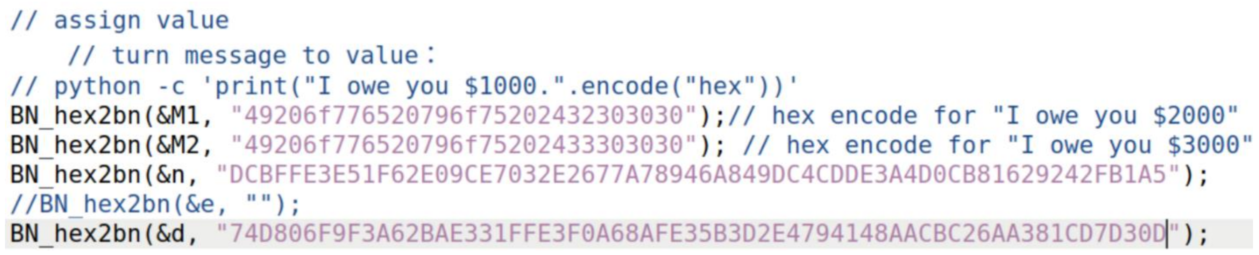
Adapted some subtle modification:

Create variables for M1 and M2 and sig1 and sig2,



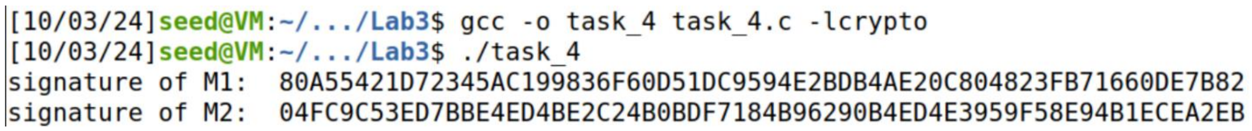
```
// create variables
BIGNUM *M1 = BN_new();
BIGNUM *M2 = BN_new();
BIGNUM *e = BN_new();
BIGNUM *n = BN_new();
BIGNUM *d = BN_new();
BIGNUM *sig1 = BN_new();
BIGNUM *sig2 = BN_new();
```

Assign the values for M1 and M2, n and d:



```
// assign value
    // turn message to value:
// python -c 'print("I owe you $1000.".encode("hex"))'
BN_hex2bn(&M1, "49206f776520796f75202432303030");// hex encode for "I owe you $2000"
BN_hex2bn(&M2, "49206f776520796f75202433303030"); // hex encode for "I owe you $3000"
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
//BN_hex2bn(&e, "");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

Generated signature for messages:



```
[10/03/24]seed@VM:~/.../Lab3$ gcc -o task_4 task_4.c -lcrypto
[10/03/24]seed@VM:~/.../Lab3$ ./task_4
signature of M1:  80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
signature of M2:  04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB
```

For “I owe you $2000” =

80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82

For “I owe you $3000” =

04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB

Task 5: Verifying a Signature

Bob receives a message M = "Launch a missile." from Alice, with her signature S. We know that Alice's public key is (e, n).

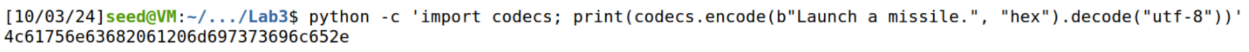
M = Launch a missile.

S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F

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

n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115

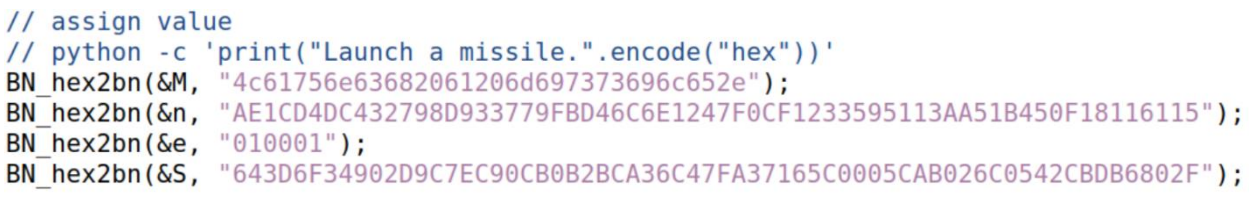
First, obtain the hex string for M :



```
[10/03/24]seed@VM:~/.../Lab3$ python -c 'import codecs; print(codecs.encode(b"Launch a missile.", "hex").decode("utf-8"))'
4c61756e63682061206d697373696c652e
```

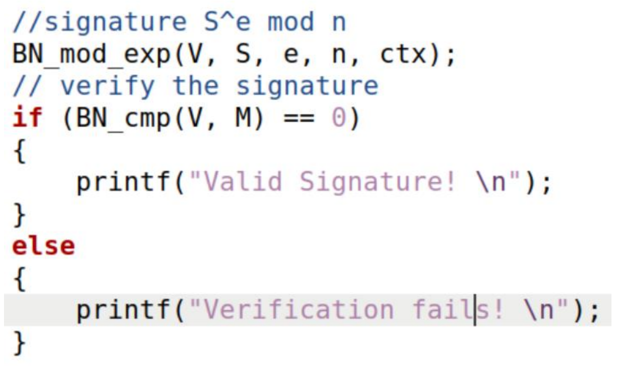
4c61756e63682061206d697373696c652e

Assign values for M and n, e , s:



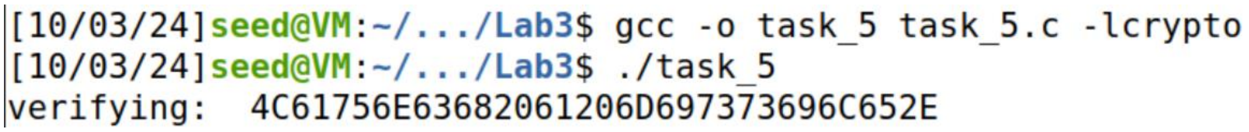
```
// assign value
// python -c 'print("Launch a missile.".encode("hex"))'
BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&e, "010001");
BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
```

Signature:



```
//signature S^e mod n
BN_mod_exp(V, S, e, n, ctx);
// verify the signature
if (BN_cmp(V, M) == 0)
{
    printf("Valid Signature! \n");
}
else
{
    printf("Verification fails! \n");
}
```

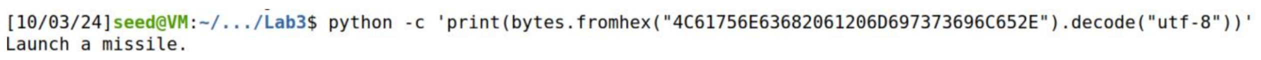
Verify the signature:



```
[10/03/24]seed@VM:~/.../Lab3$ gcc -o task_5 task_5.c -lcrypto
[10/03/24]seed@VM:~/.../Lab3$ ./task_5
verifying:  4C61756E63682061206D697373696C652E
```

4C61756E63682061206D697373696C652E

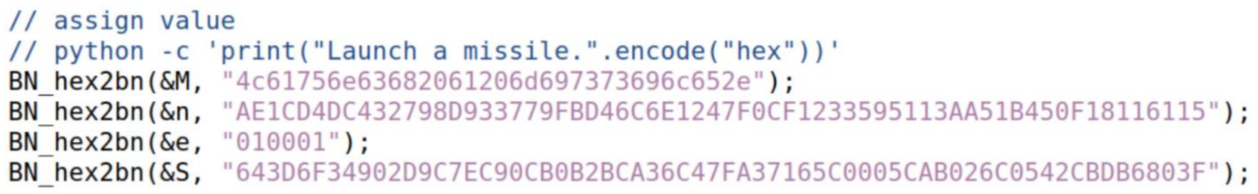
Get original message:



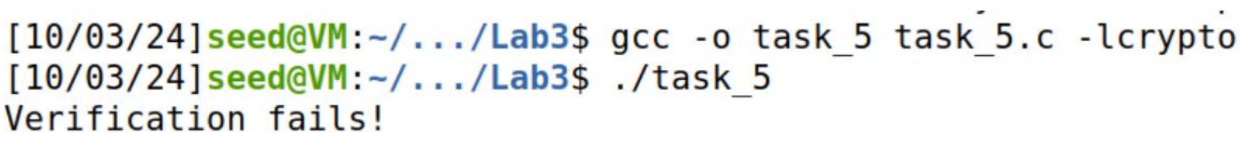
```
[10/03/24]seed@VM:~/.../Lab3$ python -c 'print(bytes.fromhex("4C61756E63682061206D697373696C652E").decode("utf-8"))'
Launch a missile.
```

It can be able to get original message.

Suppose that the signature in is corrupted, such that the last byte of the signature changes from 2F to 3F.



```
// assign value
// python -c 'print("Launch a missile.".encode("hex"))'
BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&e, "010001");
BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
```

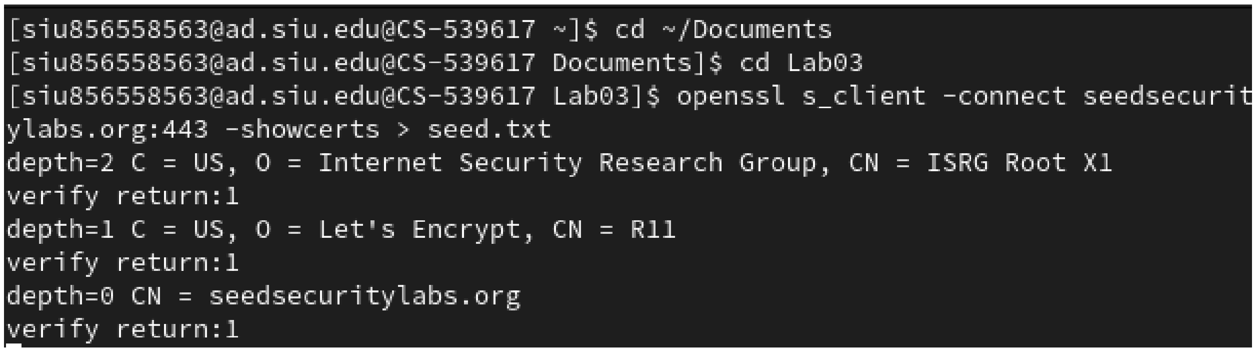


```
[10/03/24]seed@VM:~/.../Lab3$ gcc -o task_5 task_5.c -lcrypto
[10/03/24]seed@VM:~/.../Lab3$ ./task_5
Verification fails!
```

If a valid S happens to be corrupted, even just in one byte, our program will also reject the message.

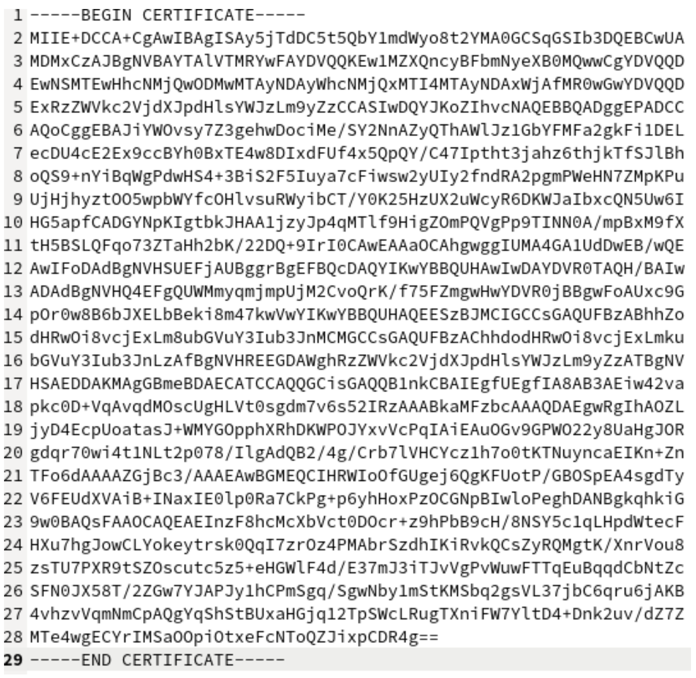
Task 6: Manually Verifying an X.509 Certificate

Download certificate for https://seedsecuritylabs.org/



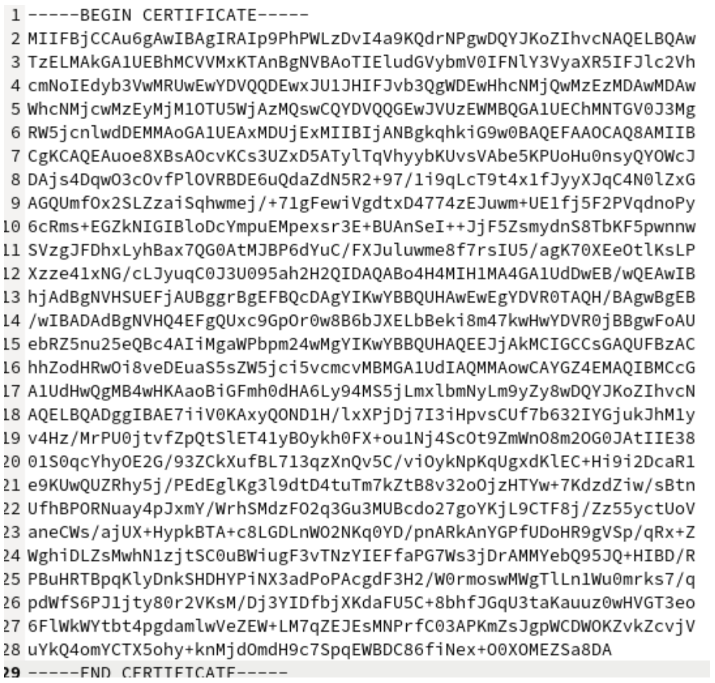
```
[siu856558563@ad.siu.edu@CS-539617 ~]$ cd ~/Documents
[siu856558563@ad.siu.edu@CS-539617 Documents]$ cd Lab03
[siu856558563@ad.siu.edu@CS-539617 Lab03]$ openssl s_client -connect seedsecurit
ylabs.org:443 -showcerts > seed.txt
depth=2 C = US, O = Internet Security Research Group, CN = ISRG Root X1
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = R11
verify return:1
depth=0 CN = seedsecuritylabs.org
verify return:1
```

Save the first certification (server's CA) as `c0.pem`:



```
-----BEGIN CERTIFICATE-----
MIIE+DCCA+CgAwIBAgISAy5jTdDC5t5QbY1mdWyo8t2YMA0GCSqGSIb3DQEBCwUA
MDMxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQwwCgYDVQQD
EwNSMTEwHhcNMjQwODMwMTAyNDAyWhcNMjQxMTI4MTAyNDAxWjAfMR0wGwYDVQQD
ExRzZWVkc2VjdXJpdHlsYWJzLm9yZzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBAJiYWOvsy7Z3gehwDociMe/SY2NnAZyQThAWlJz1GbYFMFa2gkFi1DEL
ecDU4cE2Ex9ccBYh0BxTE4w8DIxdFUf4x5QpQY/C47Iptht3jahz6thjkTfSJlBh
oQS9+nYiBqWgPdwHS4+3BiS2F5Iuya7cFiwsw2yUIy2fndRA2pgmPWeHN7ZMpKPu
UjHjhyztOO5wpbWYfcOHlvsuRWyibCT/Y0K25HzUX2uWcyR6DKWJaIbxcQN5Uw6I
HG5apfCADGYNpKIgtbkJHAA1jzyJp4qMTlf9HigZOmPQVgPp9TINN0A/mpBxM9fX
tH5BSLQFqo73ZTaHh2bK/22DQ+9IrI0CAwEAAaOCAhgwggIUMA4GA1UdDwEB/wQE
AwIFoDAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwDAYDVR0TAQH/BAIw
ADAdBgNVHQ4EFgQUWMmyqmjmpUjM2CvoQrK/f75FZmgwHwYDVR0jBBgwFoAUxc9G
pOr0w8B6bJXELbBeki8m47kwVwYIKwYBBQUHAQEESzBJMCIGCCsGAQUFBzABhhZo
dHRwOi8vcjExLm8ubGVuY3Iub3JnMCMGCCsGAQUFBzAChhdodHRwOi8vcjExLmku
bGVuY3Iub3JnLzAfBgNVHREEGDAWghRzZWVkc2VjdXJpdHlsYWJzLm9yZzATBgNV
HSAEDDAKMAgGBmeBDAECATCCAQQGCisGAQQB1nkCBAIEgfUEgfIA8AB3AEiw42va
pkc0D+VqAvqdMOscUgHLVt0sgdm7v6s52IRzAAABkaMFzbcAAAQDAEgwRgIhAOZL
jyD4EcpUoatasJ+WMYGOpphXRhDKWPOJYxvVcPqIAiEAuOGv9GPWO22y8UaHgJOR
gdqr70wi4t1NLt2p078/IlgAdQB2/4g/Crb7lVHCYcz1h7o0tKTNuyncaEIKn+Zn
TFo6dAAAAZGjBc3/AAAEAwBGMEQCIHRWIoOfGUgej6QgKFUotP/GBOSpEA4sgdTy
V6FEUdXVAiB+INaxIE0lp0Ra7CkPg+p6yhHoxPzOCGNpBIwloPeghDANBgkqhkiG
9w0BAQsFAAOCAQEAEInzF8hcMcXbVct0DOcr+z9hPbB9cH/8NSY5c1qLHpdWtecF
HXu7hgJowCLYokeytrsk0QqI7zrOz4PMAbrSzdhIKiRvkQCsZyRQMgtK/XnrVou8
zsTU7PXR9tSZOscutc5z5+eHGWlF4d/E37mJ3iTJvVgPvWuwFTTqEuBqqdCbNtZc
SFN0JX58T/2ZGw7YJAPJy1hCPmSgq/SgwNby1mStKMSbq2gsVL37jbC6qru6jAKB
4vhzvVqmNmCpAQgYqShStBUxaHGjq12TpSWcLRugTXniFW7YltD4+Dnk2uv/dZ7Z
MTe4wgECYrIMSaOOpiOtxeFcNToQZJixpCDR4g==
-----END CERTIFICATE-----
```

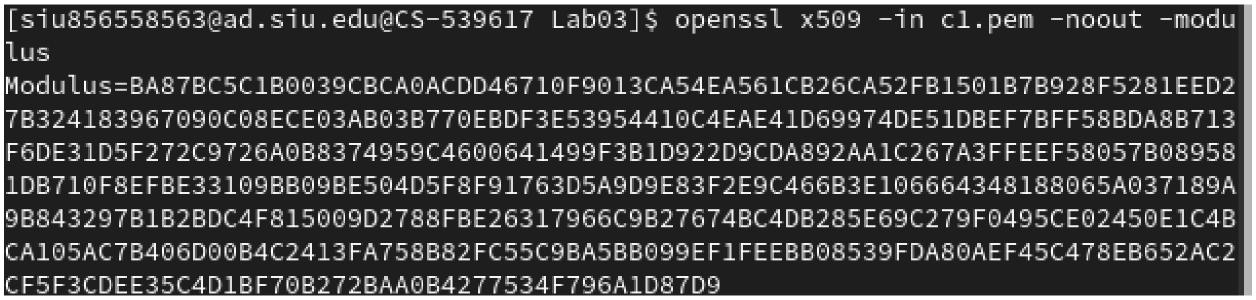
And the second one (immediate CA, issuer's CA) as `c1.pem`:



```
-----BEGIN CERTIFICATE-----
MIIFBjCCAu6gAwIBAgIRAIp9PhPWLzDvI4a9KQdrNPgwDQYJKoZIhvcNAQELBQAw
TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh
cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMjQwMzEzMDAwMDAw
WhcNMjcwMzEyMjM1OTU5WjAzMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNTGV0J3Mg
RW5jcnlwdDEMMAoGA1UEAxMDUjExMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAuoe8XBsAOcvKCs3UZxD5ATylTqVhyybKUvsVAbe5KPUoHu0nsyQYOWcJ
DAjs4DqwO3cOvfPlOVRBDE6uQdaZdN5R2+97/1i9qLcT9t4x1fJyyXJqC4N0lZxG
AGQUmfOx2SLZzaiSqhwmej/+71gFewiVgdtxD4774zEJuwm+UE1fj5F2PVqdnoPy
6cRms+EGZkNIGIBloDcYmpuEMpexsr3E+BUAnSeI++JjF5ZsmydnS8TbKF5pwnnw
SVzgJFDhxLyhBax7QG0AtMJBP6dYuC/FXJuluwme8f7rsIU5/agK70XEeOtlKsLP
Xzze41xNG/cLJyuqC0J3U095ah2H2QIDAQABo4H4MIH1MA4GA1UdDwEB/wQEAwIB
hjAdBgNVHSUEFjAUBggrBgEFBQcDAgYIKwYBBQUHAwEwEgYDVR0TAQH/BAgwBgEB
/wIBADAdBgNVHQ4EFgQUxc9GpOr0w8B6bJXELbBeki8m47kwHwYDVR0jBBgwFoAU
ebRZ5nu25eQBc4AIiMgaWPbpm24wMgYIKwYBBQUHAQEEJjAkMCIGCCsGAQUFBzAC
hhZodHRwOi8veDEuaS5sZW5jci5vcmcvMBMGA1UdIAQMMAowCAYGZ4EMAQIBMCcG
A1UdHwQgMB4wHKAaoBiGFmh0dHA6Ly94MS5jLmxlbmNyLm9yZy8wDQYJKoZIhvcN
AQELBQADggIBAE7iiV0KAxyQOND1H/lxXPjDj7I3iHpvsCUf7b632IYGjukJhM1y
v4Hz/MrPU0jtvfZpQtSlET41yBOykh0FX+ou1Nj4ScOt9ZmWnO8m2OG0JAtIIE38
01S0qcYhyOE2G/93ZCkXufBL713qzXnQv5C/viOykNpKqUgxdKlEC+Hi9i2DcaR1
e9KUwQUZRhy5j/PEdEglKg3l9dtD4tuTm7kZtB8v32oOjzHTYw+7KdzdZiw/sBtn
UfhBPORNuay4pJxmY/WrhSMdzFO2q3Gu3MUBcdo27goYKjL9CTF8j/Zz55yctUoV
aneCWs/ajUX+HypkBTA+c8LGDLnWO2NKq0YD/pnARkAnYGPfUDoHR9gVSp/qRx+Z
WghiDLZsMwhN1zjtSC0uBWiugF3vTNzYIEFfaPG7Ws3jDrAMMYebQ95JQ+HIBD/R
PBuHRTBpqKlyDnkSHDHYPiNX3adPoPAcgdF3H2/W0rmoswMWgTlLn1Wu0mrks7/q
pdWfS6PJ1jty80r2VKsM/Dj3YIDfbjXKdaFU5C+8bhfJGqU3taKauuz0wHVGT3eo
6FlWkWYtbt4pgdamlwVeZEW+LM7qZEJEsMNPrfC03APKmZsJgpWCDWOKZvkZcvjV
uYkQ4omYCTX5ohy+knMjdOmdH9c7SpqEWBDC86fiNex+O0XOMEZSa8DA
-----END CERTIFICATE-----
```

Extract the public key (e,n) from the issuer's certificate

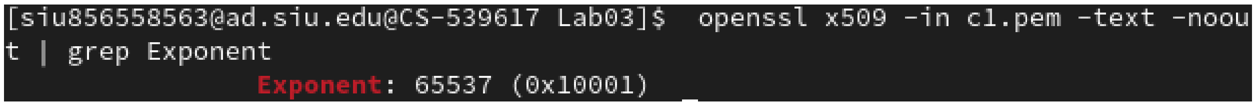
Extract the modulus `n` from an `x509` certificate



```
[siu856558563@ad.siu.edu@CS-539617 Lab03]$ openssl x509 -in c1.pem -noout -modu
lus
Modulus=BA87BC5C1B0039CBCA0ACDD46710F9013CA54EA561CB26CA52FB1501B7B928F5281EED2
7B324183967090C08ECE03AB03B770EBDF3E53954410C4EAE41D69974DE51DBEF7BFF58BDA8B713
F6DE31D5F272C9726A0B8374959C4600641499F3B1D922D9CDA892AA1C267A3FFEEF58057B08958
1DB710F8EFBE33109BB09BE504D5F8F91763D5A9D9E83F2E9C466B3E106664348188065A037189A
9B843297B1B2BDC4F815009D2788FBE26317966C9B27674BC4DB285E69C279F0495CE02450E1C4B
CA105AC7B406D00B4C2413FA758B82FC55C9BA5BB099EF1FEEBB08539FDA80AEF45C478EB652AC2
CF5F3CDEE35C4D1BF70B272BAA0B4277534F796A1D87D9
```

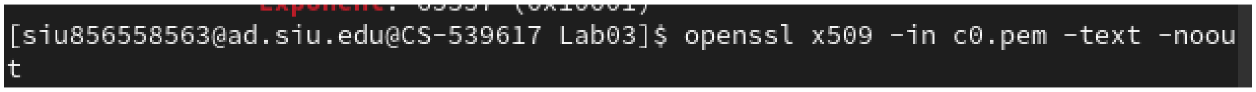
Modulus=BA87BC5C1B0039CBCA0ACDD46710F9013CA54EA561CB26CA52FB1501B7B928F5281EED27B324183967090C08ECE03AB03B770EBDF3E53954410C4EAE41D69974DE51DBEF7BFF58BDA8B713F6DE31D5F272C9726A0B8374959C4600641499F3B1D922D9CDA892AA1C267A3FFEEF58057B089581DB710F8EFBE33109BB09BE504D5F8F91763D5A9D9E83F2E9C466B3E106664348188065A037189A9B843297B1B2BDC4F815009D2788FBE26317966C9B27674BC4DB285E69C279F0495CE02450E1C4BCA105AC7B406D00B4C2413FA758B82FC55C9BA5BB099EF1FEEBB08539FDA80AEF45C478EB652AC2CF5F3CDEE35C4D1BF70B272BAA0B4277534F796A1D87D9

Print all attributes of the certificate, and then find the `exponent`, which is public key `e`:



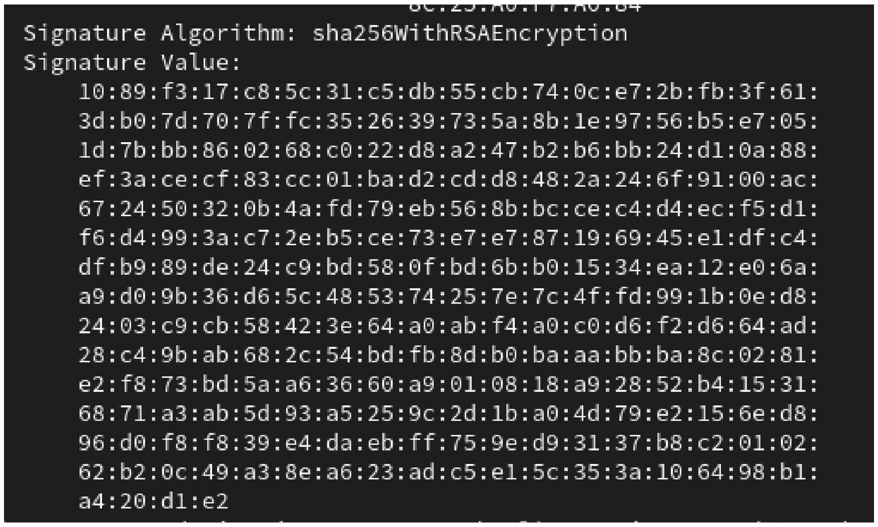
```
[siu856558563@ad.siu.edu@CS-539617 Lab03]$  openssl x509 -in c1.pem -text -noou
t | grep Exponent
                Exponent: 65537 (0x10001)
```

Extract the signature from the server's certificate



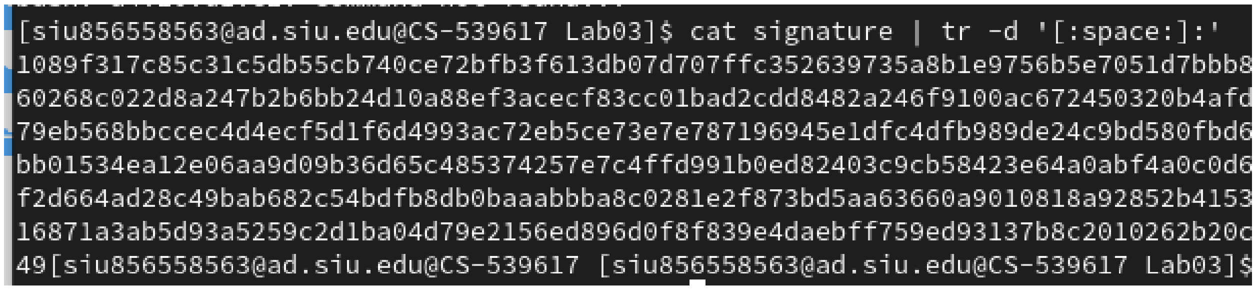
```
[siu856558563@ad.siu.edu@CS-539617 Lab03]$ openssl x509 -in c0.pem -text -noou
t
```

Find the location of last `"Signature Algorithm"`, the hex string is the body of the signature.



```
 Signature Algorithm: sha256WithRSAEncryption
 Signature Value:
     10:89:f3:17:c8:5c:31:c5:db:55:cb:74:0c:e7:2b:fb:3f:61:
     3d:b0:7d:70:7f:fc:35:26:39:73:5a:8b:1e:97:56:b5:e7:05:
     1d:7b:bb:86:02:68:c0:22:d8:a2:47:b2:b6:bb:24:d1:0a:88:
     ef:3a:ce:cf:83:cc:01:ba:d2:cd:d8:48:2a:24:6f:91:00:ac:
     67:24:50:32:0b:4a:fd:79:eb:56:8b:bc:ce:c4:d4:ec:f5:d1:
     f6:d4:99:3a:c7:2e:b5:ce:73:e7:e7:87:19:69:45:e1:df:c4:
     df:b9:89:de:24:c9:bd:58:0f:bd:6b:b0:15:34:ea:12:e0:6a:
     a9:d0:9b:36:d6:5c:48:53:74:25:7e:7c:4f:fd:99:1b:0e:d8:
     24:03:c9:cb:58:42:3e:64:a0:ab:f4:a0:c0:d6:f2:d6:64:ad:
     28:c4:9b:ab:68:2c:54:bd:fb:8d:b0:ba:aa:bb:ba:8c:02:81:
     e2:f8:73:bd:5a:a6:36:60:a9:01:08:18:a9:28:52:b4:15:31:
     68:71:a3:ab:5d:93:a5:25:9c:2d:1b:a0:4d:79:e2:15:6e:d8:
     96:d0:f8:f8:39:e4:da:eb:ff:75:9e:d9:31:37:b8:c2:01:02:
     62:b2:0c:49:a3:8e:a6:23:ad:c5:e1:5c:35:3a:10:64:98:b1:
     a4:20:d1:e2
```

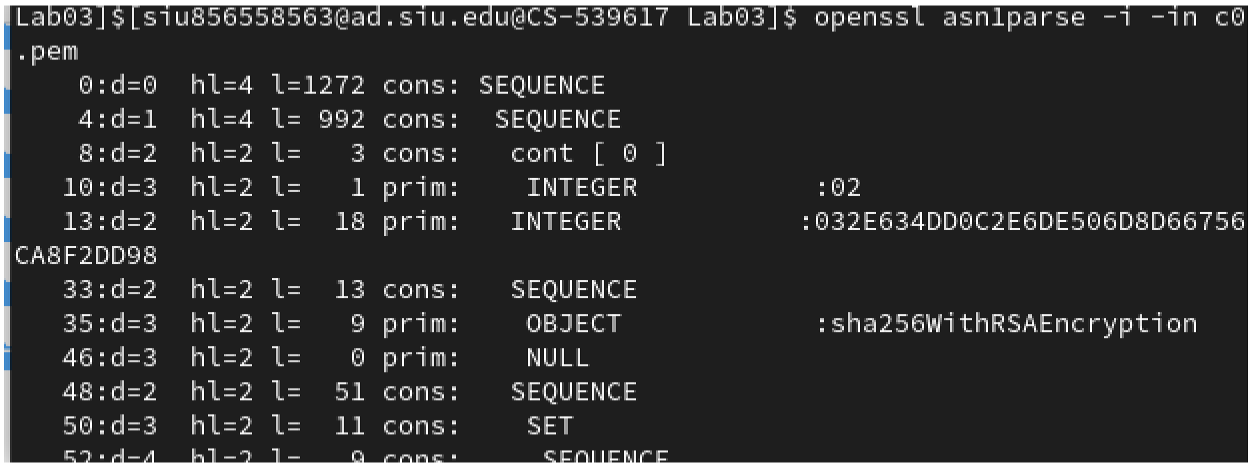
Export the body to `signature` and then trim all spaces and colons in it:



```
[siu856558563@ad.siu.edu@CS-539617 Lab03]$ cat signature | tr -d '[:space:]:'
1089f317c85c31c5db55cb740ce72bfb3f613db07d707ffc352639735a8b1e9756b5e7051d7bbb8
60268c022d8a247b2b6bb24d10a88ef3acecf83cc01bad2cdd8482a246f9100ac672450320b4afd
79eb568bbccec4d4ecf5d1f6d4993ac72eb5ce73e7e787196945e1dfc4dfb989de24c9bd580fbd6
bb01534ea12e06aa9d09b36d65c485374257e7c4ffd991b0ed82403c9cb58423e64a0abf4a0c0d6
f2d664ad28c49bab682c54bdfb8db0baaabbba8c0281e2f873bd5aa63660a9010818a92852b4153
16871a3ab5d93a5259c2d1ba04d79e2156ed896d0f8f839e4daebff759ed93137b8c2010262b20c
49[siu856558563@ad.siu.edu@CS-539617 [siu856558563@ad.siu.edu@CS-539617 Lab03]$
```

1089f317c85c31c5db55cb740ce72bfb3f613db07d707ffc352639735a8b1e9756b5e7051d
7bbb860268c022d8a247b2b6bb24d10a88ef3acecf83cc01bad2cdd8482a246f9100ac672
450320b4afd79eb568bbccec4d4ecf5d1f6d4993ac72eb5ce73e7e787196945e1dfc4dfb98
9de24c9bd580fbd6bb01534ea12e06aa9d09b36d65c485374257e7c4ffd991b0ed82403c9
cb58423e64a0abf4a0c0d6f2d664ad28c49bab682c54bdfb8db0baaabbba8c0281e2f873bd
5aa63660a9010818a92852b415316871a3ab5d93a5259c2d1ba04d79e2156ed896d0f8f83
9e4daebff759ed93137b8c2010262b20c49

Extract the body of the server's certificate

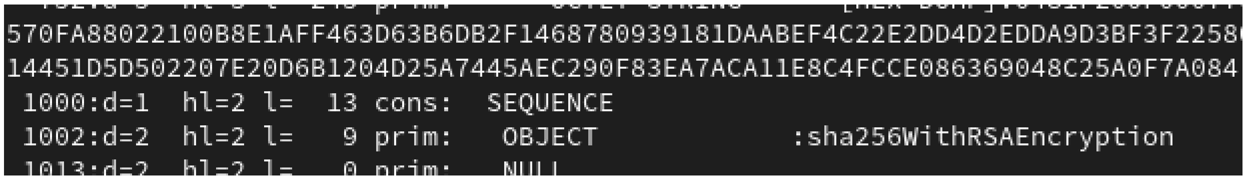


```
Lab03]$[siu856558563@ad.siu.edu@CS-539617 Lab03]$ openssl asn1parse -i -in c0
.pem
    0:d=0  hl=4 l=1272 cons: SEQUENCE
    4:d=1  hl=4 l= 992 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  18 prim:   INTEGER           :032E634DD0C2E6DE506D8D66756
CA8F2DD98
   33:d=2  hl=2 l=  13 cons:   SEQUENCE
   35:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   46:d=3  hl=2 l=   0 prim:    NULL
   48:d=2  hl=2 l=  51 cons:   SEQUENCE
   50:d=3  hl=2 l=  11 cons:    SET
   52:d=4  hl=2 l=   9 cons:     SEQUENCE
```

The field starting from

4:d=1 hl=4 l= 992 cons: SEQUENCE

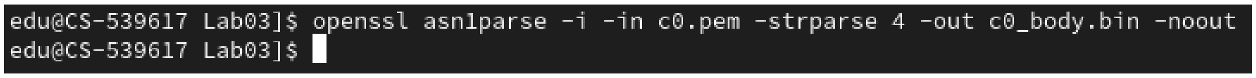
is the body of the certificate that is used to generate the hash. And the line before the last `"sha256WithRSAEncryption"` is the beginning of the signature block.



```
570FA88022100B8E1AFF463D63B6DB2F1468780939181DAABEF4C22E2DD4D2EDDA9D3BF3F2258
14451D5D502207E20D6B1204D25A7445AEC290F83EA7ACA11E8C4FCCE086369048C25A0F7A084
 1000:d=1  hl=2 l=  13 cons:  SEQUENCE
 1002:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1013:d=2  hl=2 l=   0 prim:   NULL
```

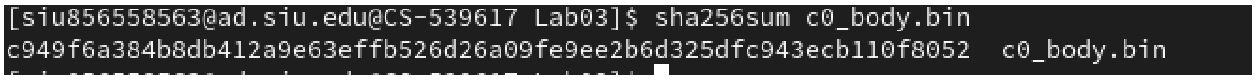
1000:d=1 hl=2 l= 13 cons: SEQUENCE

So, the certificate body starts from offset 4 to 999. Use `-strparse` to get the field from the offset 4, which is exactly the body of server's certificate:



```
edu@CS-539617 Lab03]$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
edu@CS-539617 Lab03]$
```

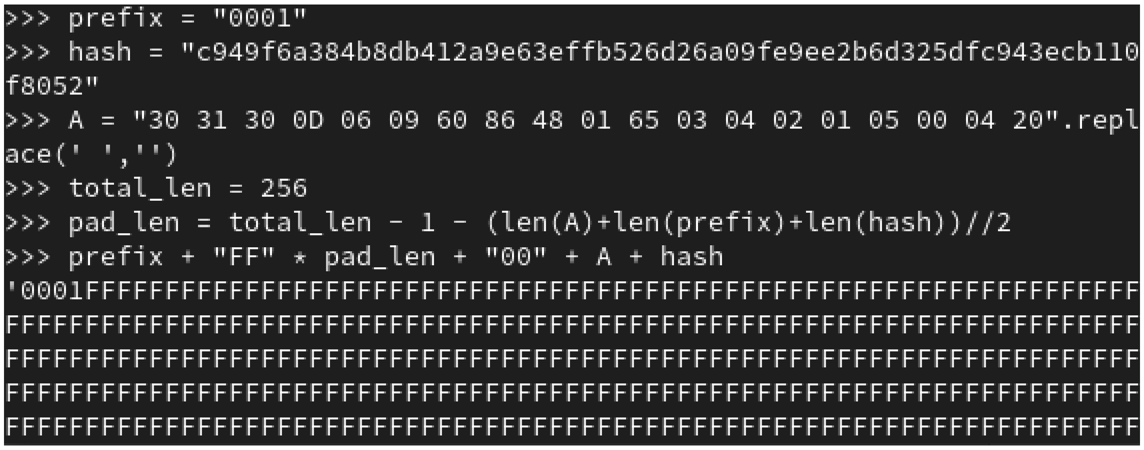
Then calculate its hash value:



```
[siu856558563@ad.siu.edu@CS-539617 Lab03]$ sha256sum c0_body.bin
c949f6a384b8db412a9e63effb526d26a09fe9ee2b6d325dfc943ecb110f8052  c0_body.bin
```

c949f6a384b8db412a9e63effb526d26a09fe9ee2b6d325dfc943ecb110f8052 c0_body.bin

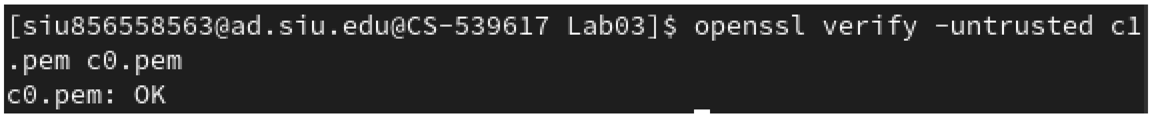
`SHA256withRSA` utilizes PKCS#1 v1.5 to pad hash. Since sizes of both the signature and the `n` are 256 bytes, we pad the hash into 256 bytes by:



```
>>> prefix = "0001"
>>> hash = "c949f6a384b8db412a9e63effb526d26a09fe9ee2b6d325dfc943ecb110
f8052"
>>> A = "30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20".repl
ace(' ','')
>>> total_len = 256
>>> pad_len = total_len - 1 - (len(A)+len(prefix)+len(hash))//2
>>> prefix + "FF" * pad_len + "00" + A + hash
'0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

'0001FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFF003031300D060960864801650304020105000420c949f6a384b8db412a9
e63effb526d26a09fe9ee2b6d325dfc943ecb110f8052'

Verify the signature.



```
[siu856558563@ad.siu.edu@CS-539617 Lab03]$ openssl verify -untrusted c1
.pem c0.pem
c0.pem: OK
```