

# **Crypto RSA**

*Seed Lab*

By Aneesh Nagdev

October 26, 2025

## Task 1:

Code for Task1 file:

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

```
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 128

void printBN(char *msg, BIGNUM * a) {
    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(); // Prime p
    BIGNUM *q = BN_new(); // Prime q
    BIGNUM *e = BN_new(); // Public exponent e
    BIGNUM *n = BN_new(); // Modulus n
    BIGNUM *phi_n = BN_new(); // Euler's Totient (n)
    BIGNUM *d = BN_new(); // Private exponent d

    // Set values for p, q, e (hexadecimal to BIGNUM conversion)
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");

    // Compute n = p * q
    BN_mul(n, p, q, ctx);
    printBN("Modulus n = ", n);
```

```

BN_sub(p, p, BN_value_one()); // p = p - 1
BN_sub(q, q, BN_value_one()); // q = q - 1
BN_mul(phi_n, p, q, ctx);
printBN("Euler's Totient φ(n) = ", phi_n);

BN_mod_inverse(d, e, phi_n, ctx); // Calculate d (private key)
printBN("Private key d = ", d);

BN_free(p);
BN_free(q);
BN_free(e);
BN_free(n);
BN_free(phi_n);
BN_free(d);
BN_CTX_free(ctx);

return 0;
}

:

```

### Compiling task1.c screenshot:

```
[10/25/25] seed@VM:~/.../Labsetup$ gcc task1.c -o task1 -lcrypto
```

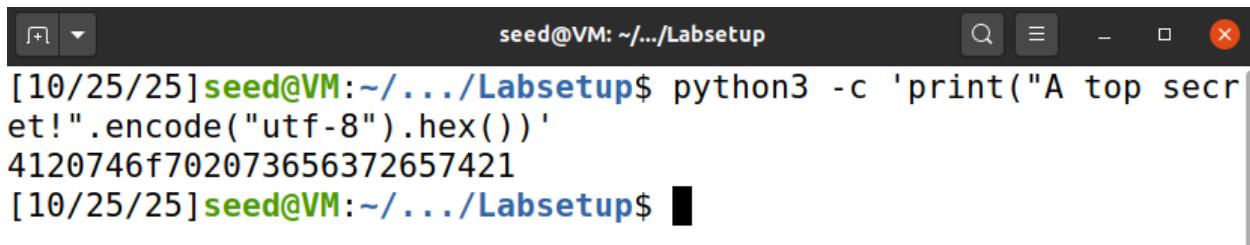
### Running task1 screenshot:

```
[10/25/25] seed@VM:~/.../Labsetup$ ./task1
Modulus n = E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0
AA1968DBB143D1
Euler's Totient (n) = E103ABD94892E3E74AFD724BF28E78348D52298B
D687C44DEB3A81065A7981A4
Private key d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33
890FE7C28A9B496AEB
[10/25/25] seed@VM:~/.../Labsetup$
```

**Description:** In this file, we created task1 file and derived the private key d from the public key components e and n. The output for the file can be seen in the screenshot above.

## Task 2:

Running the python3 code snippet screenshot:



```
seed@VM: ~/.../Labsetup$ python3 -c 'print("A top secret!".encode("utf-8").hex())'
4120746f702073656372657421
[10/25/25] seed@VM:~/.../Labsetup$
```

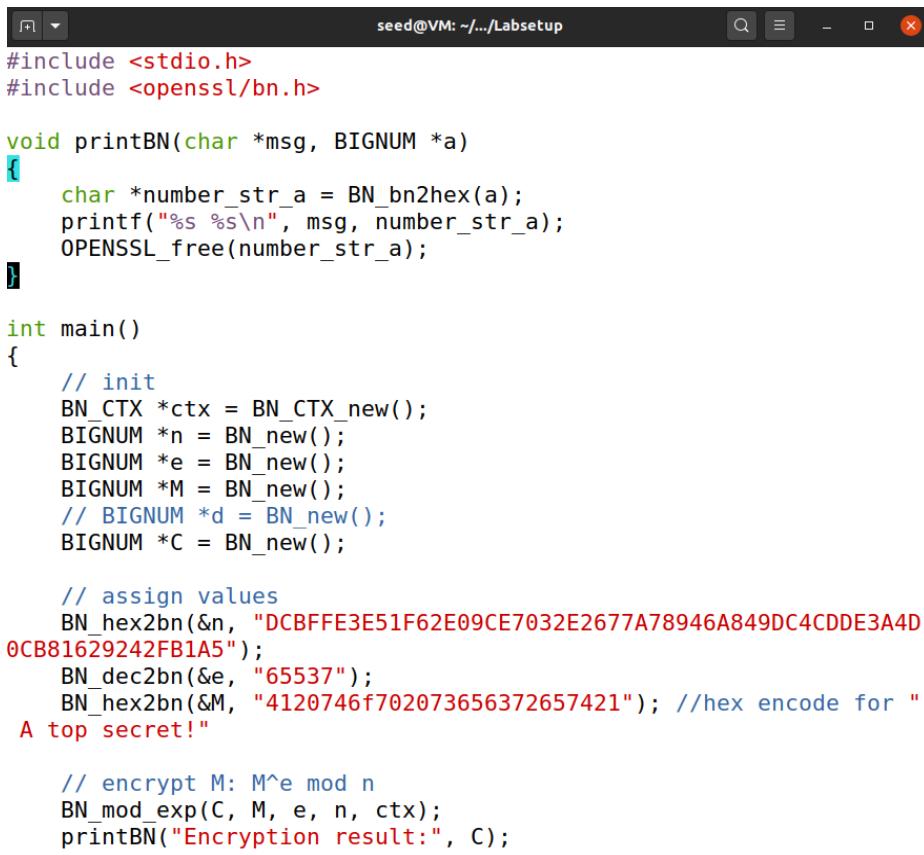
**Description:** The hex string returned represents the ASCII values of the message “A top secret!”.

Creating task2.c file screenshot:



```
seed@VM: ~/.../Labsetup$ vim task2.c
```

Partial screenshot of task2 code snippet:



```
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str_a = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str_a);
    OPENSSL_free(number_str_a);
}

int main()
{
    // init
    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();

    // assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D
0CB81629242FB1A5");
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&M, "4120746f702073656372657421"); //hex encode for "
A top secret!"

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

**Compiling task2.c file screenshot:**



```
seed@VM: ~/.../Labsetup$ gcc task2.c -o task2 -lcrypto
[10/25/25]seed@VM:~/.../Labsetup$
```

**Running task2 file screenshot:**



```
[10/25/25]seed@VM:~/.../Labsetup$ ./task2
Encryption result: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75C
ACDC5DE5CFC5FADC
[10/25/25]seed@VM:~/.../Labsetup$
```

**Description:** The message “A top secret!” was encrypted using the public key ( $e, n$ ). The message was first converted to a hexadecimal string then to a BIGNUM, and encrypted using the modular exponentiation formula. The encrypted message was printed as a hexadecimal string as seen in the screenshot above.

## Task 3:

**Creating task3.c decrypting file screenshot:**



```
seed@VM: ~/.../Labsetup$ vim task3.c
[10/25/25]seed@VM:~/.../Labsetup$
```

Screenshot of task3.c decrypting file:

```
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str_a = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str_a);
    OPENSSL_free(number_str_a);
}

int main()
{
    // init
    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();

    // assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D
0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AAC
BC26AA381CD7D30D");
    BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB6
7396567EA1E2493F");

    // decrypt C: C^d mod n
    BN_mod_exp(M, C, d, n, ctx);
    printBN("Decryption result:", M);

    // clear sensitive data
    BN_clear_free(n);
    BN_clear_free(d);
    BN_clear_free(M);
    BN_clear_free(C);

    return 0;
}
```

1,1 Top

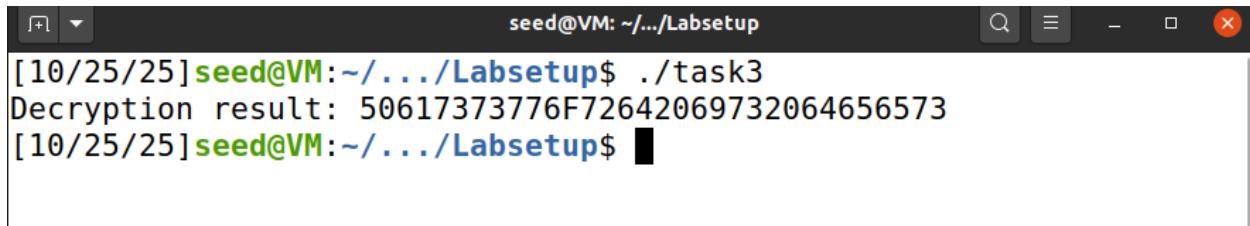
37,1 Bot

**Compiling task3.c file screenshot:**



```
seed@VM: ~/.../Labsetup$ gcc task3.c -o task3 -lcrypto
[10/25/25]seed@VM:~/.../Labsetup$
```

**Running task3 file screenshot:**



```
seed@VM: ~/.../Labsetup$ ./task3
Decryption result: 50617373776F72642069732064656573
[10/25/25]seed@VM:~/.../Labsetup$
```

**Description and explanation:** First we decrypt the cipherText using the private key and modulus. The decryption process was done using modular exponentiation. The decrypted result can be seen in the screenshot above. Next we will convert this hexadecimal to ASCII using python to get the original message.

**Converting Hex string to Plain ASCII:**



```
seed@VM: ~/.../Labsetup$ python3 -c 'print(bytes.fromhex("50617373776F72642069732064656573").decode("utf-8"))'
Password is dees
[10/25/25]seed@VM:~/.../Labsetup$
```

**Explanation and description:** Initially, we encrypted the message “A top secret!” using RSA encryption resulting in a ciphertext. Then, the encryption was done using the public key components e and n. Afterwards, we decrypted the ciphertext using the private key and recovering the original message. The decrypted result, which was in hex was converted to the plain ASCII string, revealing the message “Password is dees”, showing that the RSA encryption and decryption process.

## Task 4:

Getting the Hex for two string screenshot:



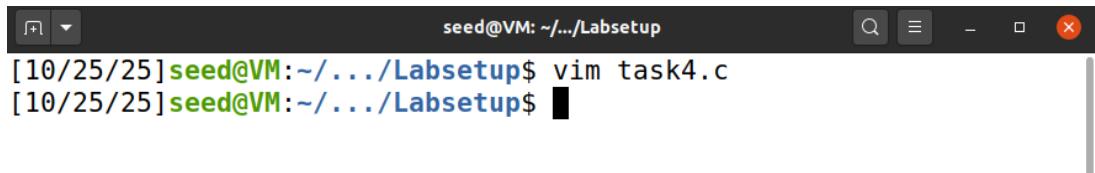
```
seed@VM: ~/.../Labsetup$ python3 -c 'print("I owe you $200\\0".encode("utf-8").hex())'
49206f776520796f75202432303030
```



```
seed@VM: ~/.../Labsetup$ python3 -c 'print("I owe you $300\\0".encode("utf-8").hex())'
49206f776520796f75202433303030
[10/25/25]seed@VM:~/.../Labsetup$
```

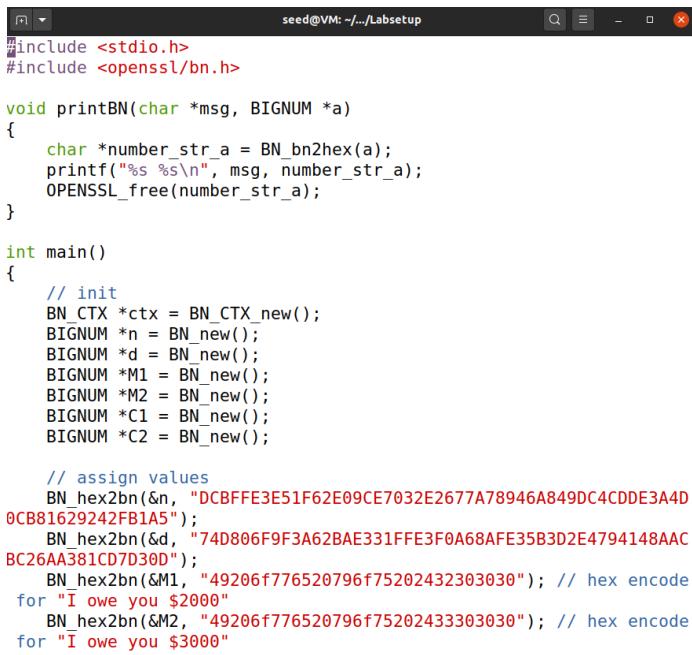
**Explanation:** This command gives us the hex for the strings.

Creating task4.c file screenshot:



```
seed@VM: ~/.../Labsetup$ vim task4.c
[10/25/25]seed@VM:~/.../Labsetup$
```

Screenshot of task4.c file:



```
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str_a = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str_a);
    OPENSSL_free(number_str_a);
}

int main()
{
    // init
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *M1 = BN_new();
    BIGNUM *M2 = BN_new();
    BIGNUM *C1 = BN_new();
    BIGNUM *C2 = BN_new();

    // assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D
0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AAC
BC26AA381CD7D30D");
    BN_hex2bn(&M1, "49206f776520796f75202432303030"); // hex encode
    for ("I owe you $2000"
        BN_hex2bn(&M2, "49206f776520796f75202433303030"); // hex encode
    for ("I owe you $3000"
```

```

// encrypt M: M^d mod n
BN_mod_exp(C1, M1, d, n, ctx);
BN_mod_exp(C2, M2, d, n, ctx);
printBN("Signature of M1:", C1);
printBN("Signature of M2:", C2);

// clear sensitive data
BN_clear_free(n);
BN_clear_free(d);
BN_clear_free(M1);
BN_clear_free(M2);
BN_clear_free(C1);
BN_clear_free(C2);

return 0;
}

```

43,1

Bot

### Compiling task4.c file screenshot:

```
[10/25/25] seed@VM:~/.../Labsetup$ gcc task4.c -o task4 -lcrypto
[10/25/25] seed@VM:~/.../Labsetup$
```

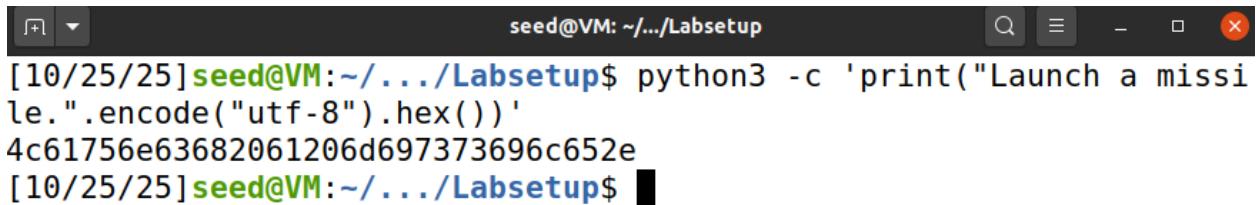
### Running task4 file screenshot:

```
[10/25/25] seed@VM:~/.../Labsetup$ task4
Signature of M1: 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C80482
3FB71660DE7B82
Signature of M2: 04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F
58E94B1ECEA2EB
[10/25/25] seed@VM:~/.../Labsetup$
```

**Description and Explanation:** The signatures for both messages calculated using the private key can be seen in the screenshot above. As expected, they both are completely different. This shows how RSA ensures message integrity. Any small alteration in the message results in a completely different signature. By comparing the signature, we can confirm whether the message has been changed.

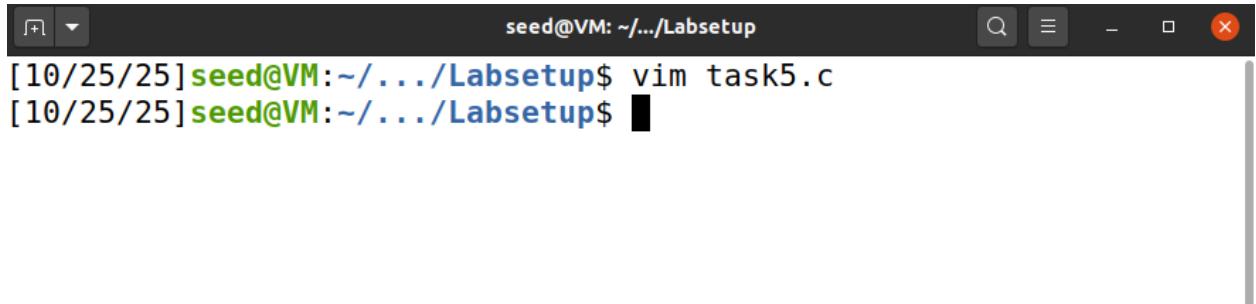
## Task 5:

Getting the Hex for the Bob's string screenshot:



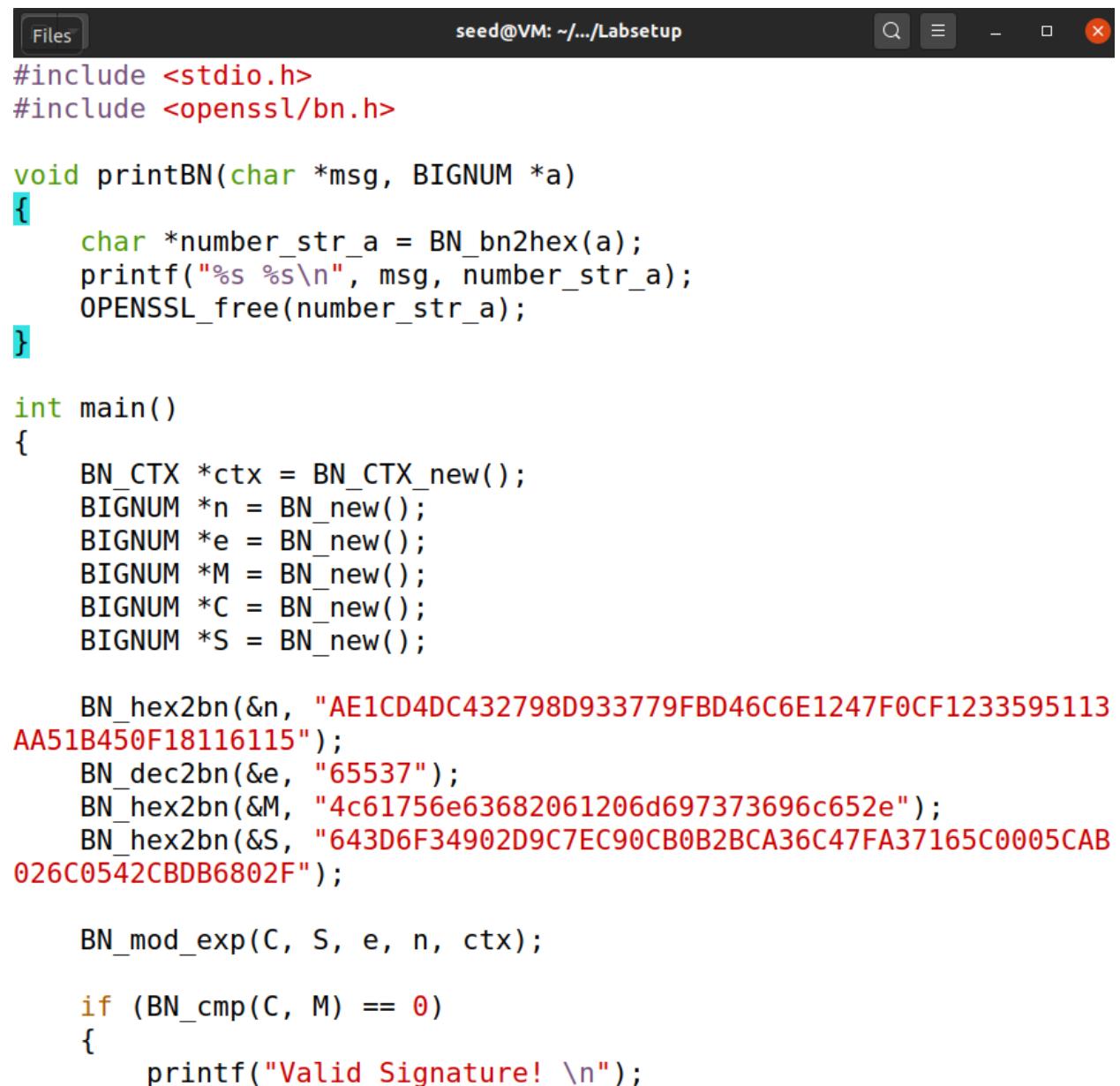
```
seed@VM: ~/.../Labsetup
[10/25/25] seed@VM:~/.../Labsetup$ python3 -c 'print("Launch a missle.".encode("utf-8").hex())'
4c61756e63682061206d697373696c652e
[10/25/25] seed@VM:~/.../Labsetup$
```

Creating task5.c file screenshot:



```
seed@VM: ~/.../Labsetup
[10/25/25] seed@VM:~/.../Labsetup$ vim task5.c
[10/25/25] seed@VM:~/.../Labsetup$
```

Screenshot of task4.c file:



The screenshot shows a terminal window with a dark theme. The title bar says "seed@VM: ~/.../Labsetup". The window contains the source code for task4.c, which includes headers for stdio.h and openssl/bn.h, defines printBN and main functions, and performs RSA operations on large integers n, e, M, C, and S.

```
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str_a = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str_a);
    OPENSSL_free(number_str_a);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();
    BIGNUM *S = BN_new();

    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113
AA51B450F18116115");
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB
026C0542CBDB6802F");

    BN_mod_exp(C, S, e, n, ctx);

    if (BN_cmp(C, M) == 0)
    {
        printf("Valid Signature! \n");
    }
}
```

```

    {
        printf("Verification fails! \n");
    }

BN_clear_free(n);
BN_clear_free(e);
BN_clear_free(M);
BN_clear_free(C);
BN_clear_free(S);

return 0;
}

```

43.1

Rot

**Description:** This code basically verifies an RSA signature by decrypting the given signature S using Alice's public key. The public exponent is provided as 65537 and used in the code as seen in the screenshot. The modules and the message M are also written and message M is represented as a hexadecimal string which we got from the python snippet. Next, we will run the file where the signature is decrypted using the modular exponentiation, and if the result is "Valid Signature!" it means the signature is valid else it will print "Verification fails!".

#### Compiling task5.c file screenshot:

```

seed@VM: ~/.../Labsetup
[10/25/25] seed@VM:~/.../Labsetup$ gcc task5.c -o task5 -lcrypto
[10/25/25] seed@VM:~/.../Labsetup$ 

```

#### Running task5 file screenshot:

```

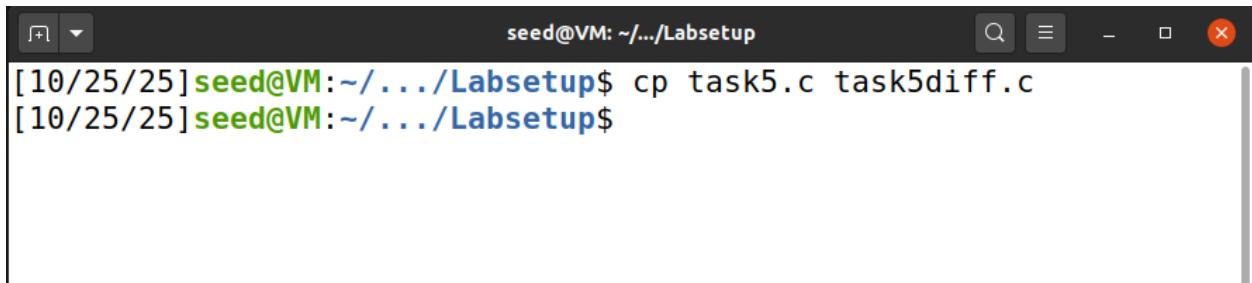
seed@VM: ~/.../Labsetup
[10/25/25] seed@VM:~/.../Labsetup$ task5
Valid Signature!
[10/25/25] seed@VM:~/.../Labsetup$ 

```

**Description and Explanation:** As it can be seen in the screenshot, "Valid Signature!" was outputted. This means the signature correctly matches the message, confirming that the message "Launch a missile." is authentic and has not been changed. The decryption of the signature using Alice's public key was successful, verifying the integrity of the message is valid.

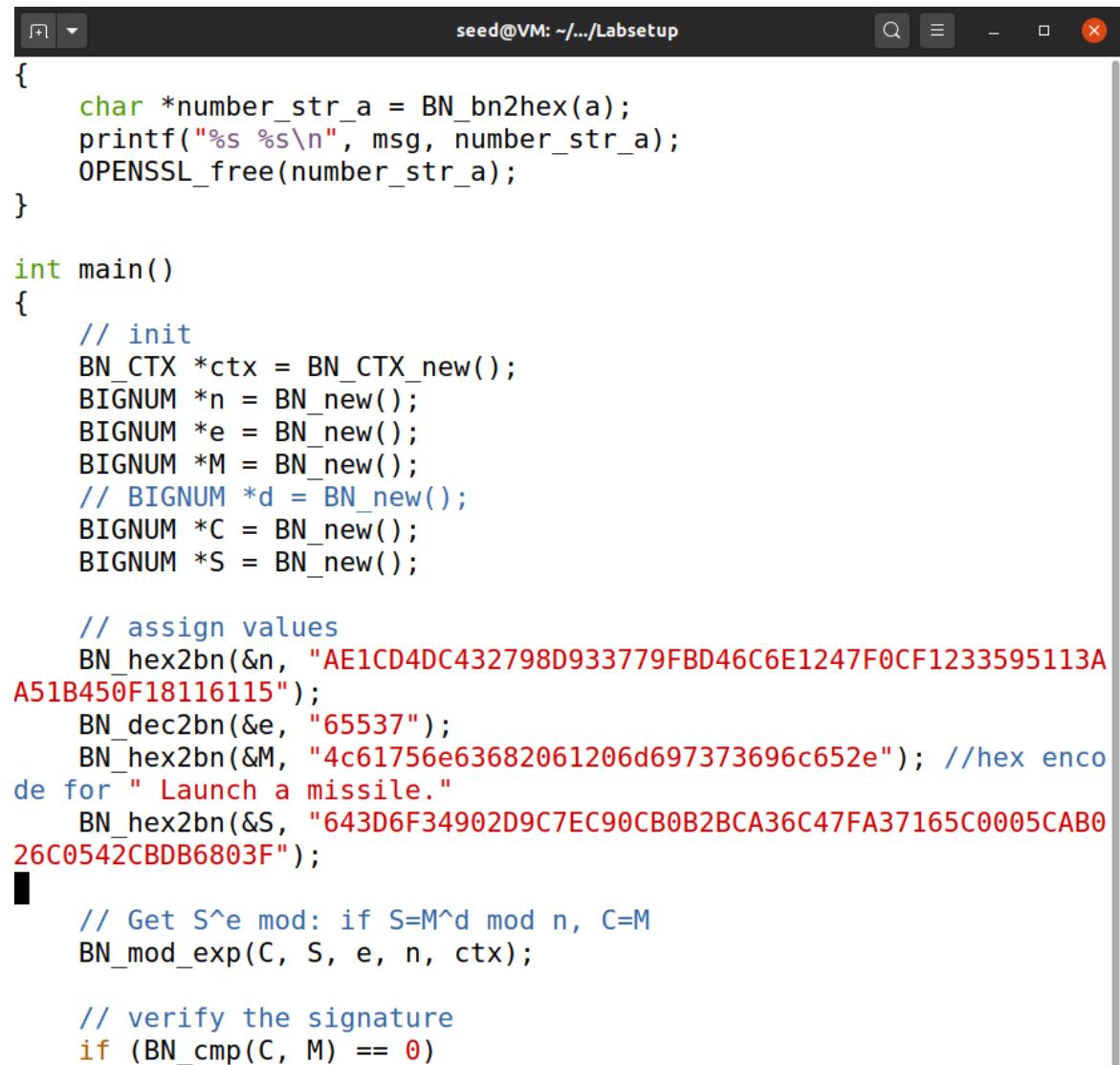
## Redoing the whole task with 3F in signature.

Copied tak5.c to task5diff.c file screenshot:



```
seed@VM: ~/.../Labsetup
[10/25/25] seed@VM:~/.../Labsetup$ cp task5.c task5diff.c
[10/25/25] seed@VM:~/.../Labsetup$
```

Changing the Signature in the code screenshot:



```
{
    char *number_str_a = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str_a);
    OPENSSL_free(number_str_a);
}

int main()
{
    // init
    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 *S = BN_new();

    // assign values
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113A
A51B450F18116115");
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&M, "4c61756e63682061206d697373696c652e"); //hex enco
de for "Launch a missile."
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB0
26C0542CBDB6803F");

    // Get S^e mod: if S=M^d mod n, C=M
    BN_mod_exp(C, S, e, n, ctx);

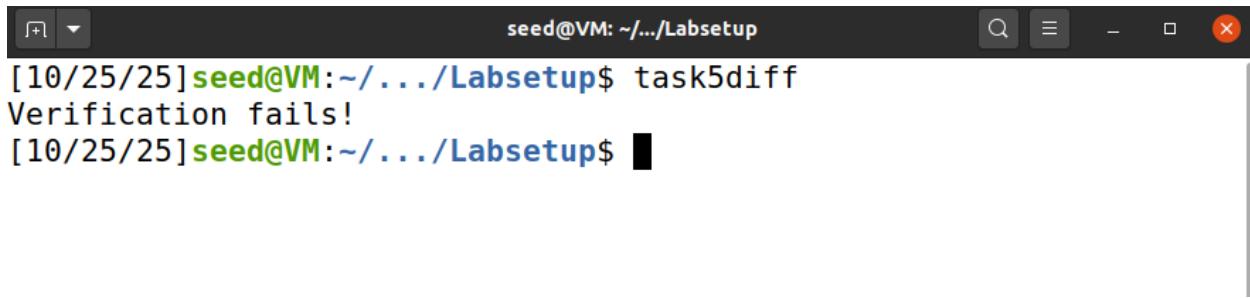
    // verify the signature
    if (BN_cmp(C, M) == 0)
```

Compiling task5diff.c file screenshot:



```
seed@VM: ~/.../Labsetup$ gcc task5diff.c -o task5diff -lcr
ypto
[10/25/25]seed@VM:~/.../Labsetup$
```

Running the task5diff file screenshot:



```
seed@VM: ~/.../Labsetup$ task5diff
Verification fails!
[10/25/25]seed@VM:~/.../Labsetup$
```

**Explanation:** As seen in the screenshot above, after corrupting the signature by changing the last byte from **2F** to **3F**, the verification results in "Verification fails!". This shows that even a small change to the signature causes the verification process to fail, demonstrating how RSA ensures the integrity of the message by detecting any alterations.

## Task 5:

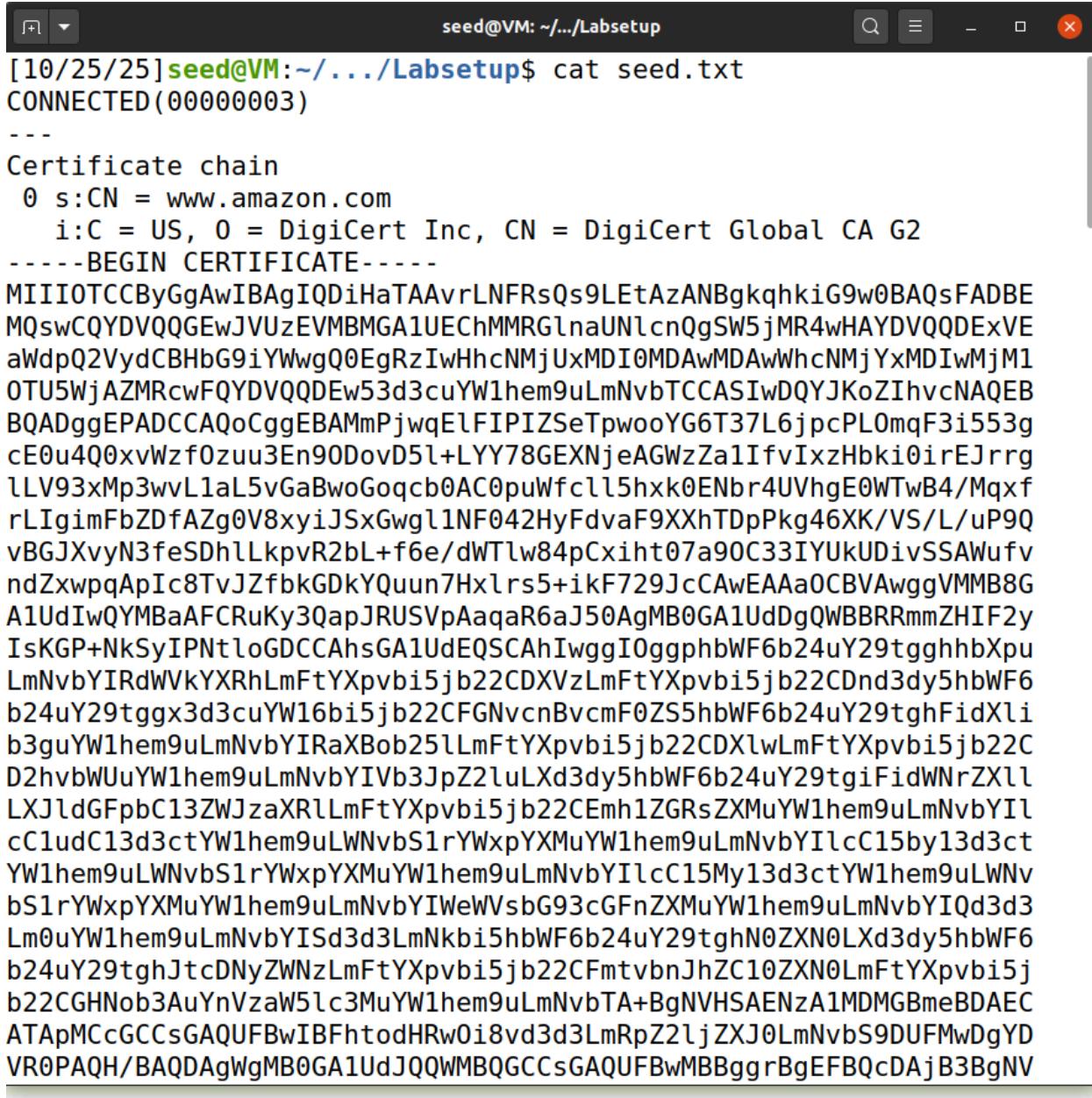
Using [www.google.com](http://www.google.com) for certificate.

Generating certificate screenshot:



```
seed@VM: ~/.../Labsetup$ openssl s_client -connect www.amazon.com:443 -showcerts > seed.txt
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiC
ert Global Root G2
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=1 C = US, O = DigiCert Inc, CN = DigiCert Global CA G2
verify return:1
depth=0 CN = www.amazon.com
verify return:1
[10/25/25]seed@VM:~/.../Labsetup$
```

### Viewing the certificate using Cat screenshot:



```
[10/25/25] seed@VM:~/.../Labsetup$ cat seed.txt
CONNECTED(00000003)
---
Certificate chain
0 s:CN = www.amazon.com
    i:C = US, O = DigiCert Inc, CN = DigiCert Global CA G2
-----BEGIN CERTIFICATE-----
MIIOTCCByGgAwIBAgIQDiHaTAAvrLNFRsQs9LEtAzANBkqhkiG9w0BAQsFADBE
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMR4wHAYDVQQDExVE
aWdpQ2VydCBHbG9iYWwgQ0EgRzIwHhcNMjUxMDI0MDAwMDAwWhcNMjYxMDIwMjM1
OTU5WjAZMRcwFQYDVQQDEw53d3cuYW1hem9uLmNvbTCCASIwDQYJKoZIhvcNAQEB
BQADggEPADCCAQoCggEBAMmPjwqElFIPIZSeTpwooYG6T37L6jpcPL0mqF3i553g
cE0u4Q0xvWzfOzuu3En90DovD5l+LYY78GEXNjeAGWzZa1IfvIxzhbki0irEJrrg
tLV93xMp3wvL1aL5vGaBwoGoqcb0AC0puWfc1l5hxk0ENbr4UVhgE0WTwB4/Mqxf
rLigimFbZDfAZg0V8xyiJSxGwg1NF042HyFdvaF9XXhTDpPkg46XK/VS/L/uP9Q
vBGJXvyN3feSDh1LkpvR2bL+f6e/dWTlw84pCxih07a90C33IYUkUDivSSAwufv
ndZxwpqApIc8TvJZfbkGdkYQuun7Hxlrs5+ikF729JcCAwEAAs0CBVAwggVMMB8G
A1UdIwQYMBaAFCRuKy3QapJRUSVpAaqaR6aJ50AgMB0GA1UdDgQWBRRmmZHIF2y
IsKGP+NkSyIPNtloGDCCAhsGA1UdEQSCAhIwggI0ggphbWF6b24uY29tghhbXpu
LmNvbYIRdWVkYXRhLmFtYXpvbi5jb22CDXVzLmFtYXpvbi5jb22CDnd3dy5hbWF6
b24uY29tggx3d3cuYW16bi5jb22CFGNvcnBvcnF0ZS5hbWF6b24uY29tghFidXli
b3guYW1hem9uLmNvbYIRaXBob25lLmFtYXpvbi5jb22CDXlwLmFtYXpvbi5jb22C
D2hvbWUuYW1hem9uLmNvbYIVb3JpZ2luLXd3dy5hbWF6b24uY29tgiFidWNrZXll
LXJldGFpbC13ZWJzaXR1LmFtYXpvbi5jb22CEmh1ZGRsZXMuYW1hem9uLmNvbYI1
c1udC13d3ctYW1hem9uLWNvbS1rYWxpYXMuYW1hem9uLmNvbYI1cC15by13d3ct
YW1hem9uLWNvbS1rYWxpYXMuYW1hem9uLmNvbYI1cC15My13d3ctYW1hem9uLWNv
bS1rYWxpYXMuYW1hem9uLmNvbYI1WeWVsB93cGFnZXMuYW1hem9uLmNvbYI1Qd3d3
Lm0uYW1hem9uLmNvbYISd3d3LmNkbi5hbWF6b24uY29tghN0ZXN0LXd3dy5hbWF6
b24uY29tghJtcDNyZWNzLmFtYXpvbi5jb22CFmtvbnJhZC10ZXN0LmFtYXpvbi5j
b22CGHNob3AuYnVzaW5l3MuYW1hem9uLmNvbTA+BgNVHSAENzA1MDMGBmeBDAEC
ATApMCcGCCsGAQUFBwIBFhtodHRw0i8vd3d3LmRpZ2ljkZXJ0LmNvbS9DUFMwDgYD
VR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjB3BgnV
```

HR8EcDBuMDWgM6Axhi9odHRw0i8vY3JsMy5kaWdpY2VydC5jb20vRGlnaUNlcnRH  
bG9iYWxDQuCyLmNybDA1oD0gMYYvaHR0cDovL2NybdQuZGlnaWNlcnQuY29tL0Rp  
Z21DZXJ0R2xvYmfQ0FHMi5jcmwwdAYIKwYBBQUHAQEEaDBmMCQGCCsGAQUFBzAB  
hhodHRw0i8vb2NzcC5kaWdpY2VydC5jb20wPgYIKwYBBQUHMAKGmh0dHA6Ly9j  
YWNlcnRzLmRpZ21jZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbENBRzIuY3J0MAwGA1Ud  
EwEB/wQCMAAwggF9BgorBgEEAdZ5Ag0QCBIIbBqSCAWkBzWb2ANdtfDRRp/V3wsfp  
X9cAv/mCyTNAzeHQswFzF8DIxWl3AAABmhRnX5gAAAQDAEcwRQIhALy+XEal28nG  
YnA/PEMwvCB+/Pg20Uq8UNA6ehC3RnZiAiBmndMjaXzmJEBSFa9Zy4Ls13PukA9  
y8yIFKkgwvfK8QB1AMIXfldFGaNf7n843rKQQevHwiFaIr9/1bWtdprZDlLNAAAB  
mhRnX1YAAAQDAEYwRAIgTYr5VQEfk7RpQICi0tLWWF7GxNDVHaQHurkl0fPKTvkC  
IDY1Mkj6KDsaAsWfwoyPQfP8iK37ZV1QvV0/hXsJqkyVAHYA1E5Dh/rswe+B8xkk  
JqgYZQHH0184AgE/cmd9VTcuGdgAAAGaFGdfZAAABAMARzBFAiEA6Y3TR6gpvn1b  
qftSw2Y4V05LWoe/vfxqGwqLo/e8r0CIHZpk4qWcpk606X9Vum7fVdLuD8I0yE  
mgBYbzAN1MtzaMA0GCSqGSIB3DQEBCwUA4IBAQAmphQ3sGV4wbQshoV0PZiP9Zi  
W3YxHU7F3i040bi9JEI/DvLy5mK+RU2RYr1ZlofQ2jkQF86nCX0jGHPdXF/eSd6  
Y5M46kNDSomv9pN8aC9mMD+wDafwbAu9zFdjAPio3o8x40ejnbkK4nn9A4iXR  
0FK5sMixgMDTBmqcA53JrUS7V+Ox7uTQ+NSIQY7D9fprm/xs1fnQHDiTe8PQ+bWT  
m2A6HTYXDbxzTkHE80v2jjimML0p8bsWau3Up6ruchpPIrfvcsjux8JE6KA8cc//  
Ioy/6CZ5wgM0v2NCN0p2eze+wPPSvMLnA511bKBhWIPLsh50zm0G/10eqjNU  
-----END CERTIFICATE-----  
1 s:C = US, 0 = DigiCert Inc, CN = DigiCert Global CA G2  
i:c = US, 0 = DigiCert Inc, OU = www.digicert.com, CN = DigiCert  
Global Root G2  
-----BEGIN CERTIFICATE-----  
MIIEizCCA30gAwIBAgIQDI7gy1qjRWIBAYe4kh5rzANBgkqhkiG9w0BAQsFADBh  
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3  
d3cuZGlnaWNlcnQuY29tMSAwHgYDVQDExdEaWdpQ2VydCBhbG9iYwggUm9vdCBH  
MjAeFw0xMzA4MDExMjAwMDBaFw0y0DA4MDExMjAwMDBaMEQxCzAJBgNVBAYTA1VT  
MRUwEwYDVQQKEwxEaWdpQ2VydCBjbMxHjAcBgNVBAMTFURpZ21DZXJ0IEdsb2Jh  
bCBDQSBSHmjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANNIfL7zBYzd  
W9UvhU5L4IatFaxhz1uvPmoKR/uadPfgC4przc/cv35gmAvkVNlW7SHMarZagV+X  
au4CLyMnuG3Us0cGAngLH1ypmTb+u6wbBfpXzYEQQGfWMItYNdSWYb7QjHqXnxr5

IuYUL6nG6AEfq/gmD6y0TSwy0R2Bm40cZbIc22GoiS9g5+vCShjEbyrpEJIJ7rfR  
ACvmfe8EiRR0M6GyD5eHn70gzS+8L0y4g2gxPR/VSpAQGQuBldYpd1H5NnbQtwl6  
0ErXb4y/E3w57bqukPyV93t4CTZedJMjeJfD/1K2uaGvG/w/VNfFVbkhJ+P1474j4  
8V4Rd6rfArMCawEEAA0CAvowggFWMBIGA1UdEwEB/wQIMAYBAf8CAQAwDgYDVROp  
AQH/BAQDAGGMDQGQCsGAQUFBwEBBCGwJjAkBggrBgeFBQcwAYYAHR0cDovL29j  
c3AuZGlnaWNlcnQuY29tMhsGA1udHwR0MHIwN6A1oD0GMWh0dHA6Ly9jcmw0LmRp  
Z21jZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwN6A1oD0GMWh0dHA6  
Ly9jcmwzLmRpZ21jZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwPQYD  
VR0gBDYwNDAYBgRVHSAAMCowKAYIKwYBBQUHAgEWGHg0dBz0i8vd3d3LmRpZ21j  
ZXJ0LmNvbS9DUFMwHQYDVR00BBYECFRuKy3QapJRUsVpAqaR6aJ50AgMB8GA1Ud  
IwQYMBaAFE4iVCAYlebjbuYP+vq5Eu0GF485MA0GCSqGSIB3DQEBCwUA4IBAQAL  
0YSR+ZfrqoGvh0la0JL84mxZvbIRaxHhBscMsdaVsnaT0AC9aHes03ewPj2  
dZ12uYf+QYB6z13jAMZbAuabeGLJ3LhimmnftiQjXs8X9Q9ViIyfEBFltcT8jW+rZ  
8uckJ2/01Ydb1zkIVp6hnV0WXu0LRg9eFhSvGNoVwvdRLNXSmDmyHBwW4co  
atc7TLJFGa8kBpJIERqLrqwYElesA8u4L3KJg6nw3jM+/AVTAN1VlOnAM2BvjA  
jx5ZnE0qnsHhfTuvcqdfuh0Wku4Z0BqYbvQ3lBetoxi6PrABDJXWKTUgNX31EGdk  
92hiHuwZ4STyhG6Qia  
-----END CERTIFICATE-----  
2 s:C = US, 0 = DigiCert Inc, OU = www.digicert.com, CN = DigiCert  
Global Root G2  
i:c = US, 0 = "VeriSign, Inc.", OU = VeriSign Trust Network, OU  
= "(c) 2006 VeriSign, Inc. - For authorized use only", CN = VeriSig  
n Class 3 Public Primary Certification Authority - G5  
-----BEGIN CERTIFICATE-----  
MIIE3zCCA8egAwIBAgIQSysBFc3lx0gbPN3+3hEWnjANBgkqhkiG9w0BAQsFADCB  
yjELMAKGA1UEBhMCVVmxFzAVBgvNVAoTDLZlcm1TaWduLCBjbMuMR8wHQYDVQQL  
ExZWZXJpU2lnbiBUcnVzdCB0ZXr3b3JrMTow0AYDVQQLezEoYykgMjAwNiBWZXJp  
U2lnbiwgSW5jLiAtIEZvciBhdXRrob3JpemVkiHVzZSBvbm5MUUwQwYDVQDExzW  
ZXJpU2lnbiBDGFzcyAzIFBLYmpYyB0cmLtYXJ5IEEnlcRpZmljYXRpb24gQXV0  
aG9yaXR5IC0gRzUwHhcNMTgwNDAzMDAwMDAwWhcNMjgwNDAYMjM10TU5WjBhMQsw  
CQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3d3cu  
ZGlnaWNlcnQuY29tMSAwHgYDVQDExdEaWdpQ2VydCBhbG9iYwggUm9vdCBHmjCC

ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALs3zTTce2vJsmiQrUp1/0a6  
IQoIjfUZVMn7iNvzrvI6iZE8euarBhprz6wt6F4JJES6Ypp+1q0ofuBUdSAFrFC3  
nGMabDDc2h8Zsdce3v3X4MuUgzeu7B9DTt17LNK9LqUv5Km4rTrUmaS2JembawBg  
kmD/TyFJGPdnkKthBpyP8rrpt0mSMmu181foXRvNjB2rlQSfM1LzbjSW3dd+P7  
SUu0rFUhqY+Vs7Qju0xtRfD2qbKVMLT9TFWMJ0pXFHyCnc1zktMWsgYMjFDRjx4J  
vheh5iHK/YPlELyDpQrEZyj2cxQUPUZ2w4cUiSE0Ta8PRQymSaG6u5zFsT0DKYUC  
AwEAAa0CAScwggEjMB0GA1UdDgQWBBr0IlQgGJXm427mD/r6uRLtBheP0TAPBgNV  
HRMBAf8EBTADAQH/MF8GA1UdIARYMFYwVAYEVROgADBMMCMGCCsGAQUFBwIBFhdo  
dHRwczovL2QuC3ltY2IuY29tL2NwczAlBggRBgEFBQcCAjAZDBdodHRwczovL2Qu  
c3ltY2IuY29tL3JwYTAvgNVHR8EKDAmMCsgIqAghh5odHRw0i8vcy5zeW1jYi5j  
b20vcGNhMy1nNS5jcmwwDgYDVR0PAQH/BAQDAgGGMC4GCCsGAQUFBwEBBCIwIDAe  
BggRBgEFBQcwAYYSaHR0cDovL3Muc3ltY2QuY29tMB8GA1UdIwQYMBaAFH/TZafC  
3ey78DAJ80M5+gKvMzEzMA0GCSqGSIB3DQEBCwUAA4IBAQCOStidCZ2ebdzF1Jkd  
ijPBIAyeed+Chn4dWDm0ptwghHtZ7Q1DRFGLl0QQgXmcFBZtRCcGRgC2ZZDTjFYE  
fs1mnDD3J+fKxm+cM6rR5JTLLxP4j3v6k91tk1l3TtZhucff5qs+SNmpp/QQtneN  
JoyWHawD8Knb1BZ0UA9sxrFeFdJz4uAz+ld6Fdm1ZtU5gMQXU0KHL6ZGry8+Qnax  
9A35U6m4eroyFSImrbteD0W5jkZtDuu3E33vx7Di7yLZV/Pfto/28wAcHxnukPk  
CmMNKS5wFI5j7dStFFn6QWYpQX5SGuu7ywTg0E/nbsDXwqcUs0+elydJvwFEwtwX  
b0ES

-----END CERTIFICATE-----

---

Server certificate

subject=CN = www.amazon.com

issuer=C = US, O = DigiCert Inc, CN = DigiCert Global CA G2

---

No client certificate CA names sent

Peer signing digest: SHA256

Peer signature type: RSA-PSS

Server Temp Key: X25519, 253 bits

---

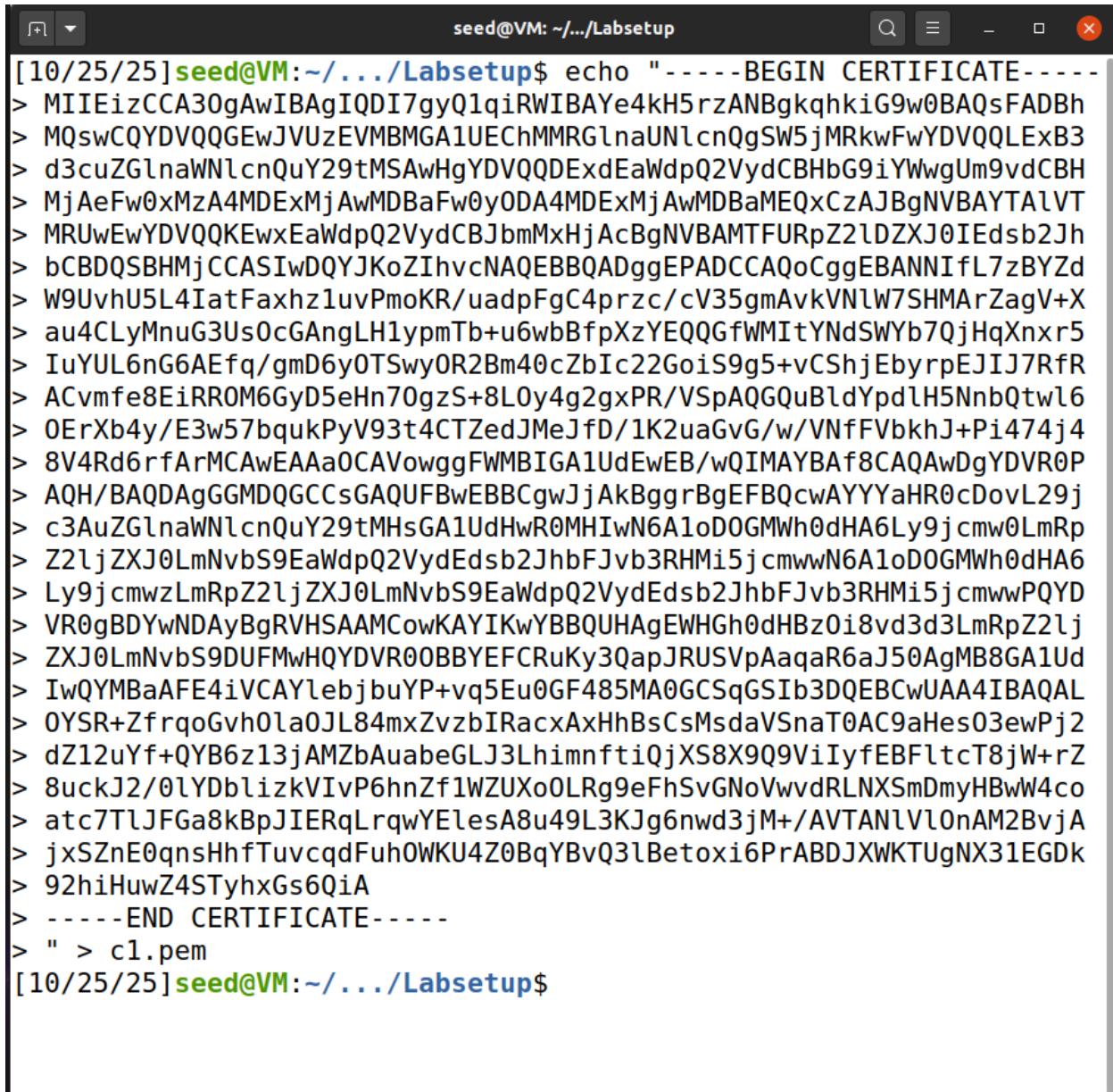
```
SSL handshake has read 5081 bytes and written 380 bytes
Verification error: unable to get local issuer certificate
---
New, TLSv1.3, Cipher is TLS_AES_128_GCM_SHA256
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data 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_AES_128_GCM_SHA256
    Session-ID: F0453C9C9572C64327D8C63D52FCF93DC43A2815A6690AB4533
3655FC33A3C42
    Session-ID-ctx:
    Resumption PSK: 868FF979D95ABA878A2E7864B7A796F5EEA3B49E9473E57
6D3C25514ECEC71E3
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 64864 (seconds)
    TLS session ticket:
        0000 - 01 31 37 36 31 34 32 32-32 33 34 30 30 30 00 00 .17614
22234000..
        0010 - 00 a5 26 ff 29 78 81 1a-be bf 5e 8f 32 21 30 02 ..&.)x
....^.2!0.
        0020 - 51 bd a9 2c fd be a1 9f-63 37 e0 cd 8f 68 80 62 Q...,...
```

```
..c7...h.b
    0030 - 15 77 69 2d 0d 21 dc 8d-8c 40 29 59 55 07 4d 9e .wi-.!.
...@)YU.M.
    0040 - db 9e 7a 03 3a ea 2d f8-00 43 59 75 d0 f0 47 f8 ..z:..
-..CYu..G.
    0050 - aa a2 f1 d3 d4 e4 8f 95-12 66 bf 12 31 50 e3 b4 .....
...f..1P..
    0060 - bd 30 42 3b 10 14 e8 7b-c4 53 e6 6a 39 7f eb cb .0B;..
.{.S.j9...
    0070 - 4b 1a 8f c8 d8 74 30 fb-65 19 34 c7 92 6f 7c 18 K....t
0.e.4..o|.
    0080 - 77 76 ce 74 a2 bb 03 0c-ac 09 wv.t..
.....
Start Time: 1761443768
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
[10/25/25]seed@VM:~/.../Labsetup$
```

### Saving the first certificate as c0.pem:

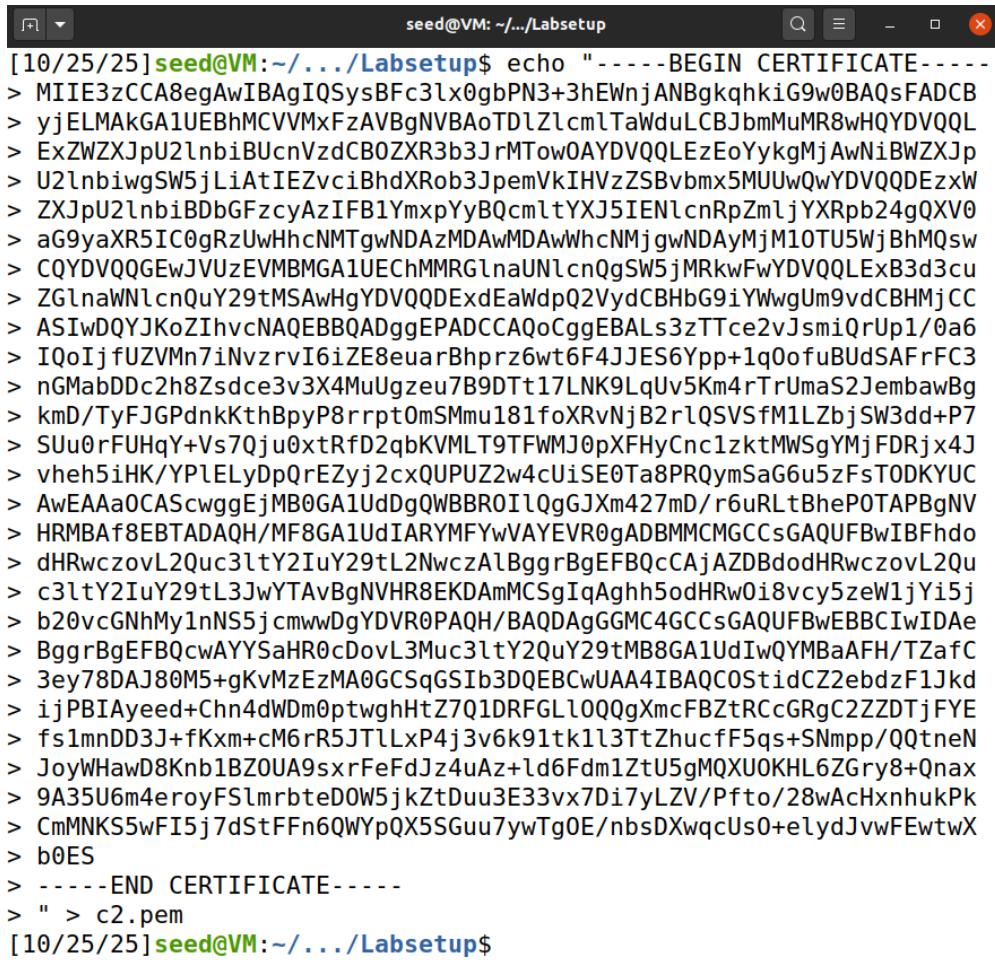
```
[10/25/25]seed@VM:~/.../Labsetup$ echo "-----BEGIN CERTIFICATE-----" > MIIIOTCCByGgAwIBAgIQDiHaTAAvRLNFRsQs9LEtAzANBgkqhkiG9w0BAQsFADBE > MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMR4wHAYDVQQDExVE > aWdpQ2VydCBhbG9iYWwgQ0EgRzIwHhcNMjUxMDI0MDAwMDAwWhcNMjYxMDIwMjM1 > OTU5WjAZMRCwFQYDVQQDEw53d3cuYW1hem9uLmNvbTCASIwDQYJKoZIhvcNAQEB > BQADggEPADCCAQoCggEBAMmPjwqElFIPIZSeTpwooyG6T37L6jpcPL0mqF3i553g > cE0u4Q0xvWzfOzuu3En90DovD5l+LYY78GEXNjeAGWzZa1IfvIxzbki0irEJrrg > lLV93xMp3wvL1aL5vGaBwoGoqcb0AC0puWfc1l5hxk0ENbr4UVhgE0TwB4/Mqxf > rLIgimFbZDfAZg0V8xyiJSxGwgl1NF042HyFdvaF9XXhTDpPkg46XK/VS/L/uP9Q > vBGJXvyN3feSDh1LkpvR2bL+f6e/dWTlw84pCxiht07a90C33IYUkUDivSSAWufv > ndZxwpqApIc8TvJZfbkGDkYQuun7Hxlrs5+ikF729JcCAwEAAsOCBVAwggVMMB8G > A1UdIwQYMBaAFCRuKy3QapJRUSVpAqaqA6aJ50AgMB0GA1UdDgQWBRRmmZHIF2y > IsKGP+NkSyIPNtloGDCCAhsGA1UdEQSCAhIwggI0ggphbWF6b24uY29tgghbXpu > LmNvbYIRdWVkYXRhLmFtYXpibi5jb22CDXVzLmFtYXpibi5jb22CDnd3dy5hbWF6 > b24uY29tggx3d3cuYW16bi5jb22CFGNvcnBvcmF0ZS5hbWF6b24uY29tghFidXli > b3guYW1hem9uLmNvbYIRaXBob25lLmFtYXpibi5jb22CDXlwLmFtYXpibi5jb22C > D2hvbWUuYW1hem9uLmNvbYIVb3JpZ2luLXd3dy5hbWF6b24uY29tgiFidWNrZXll > LXJldGFpbC13ZWJzaXR1LmFtYXpibi5jb22CEmh1ZGRsZXMuYW1hem9uLmNvbYI1 > cCludC13d3ctYW1hem9uLWNvbS1rYWxpYXMuYW1hem9uLmNvbYI1cC15by13d3ct > YW1hem9uLWNvbS1rYWxpYXMuYW1hem9uLmNvbYI1cC15My13d3ctYW1hem9uLWNv > bS1rYWxpYXMuYW1hem9uLmNvbYIWeWVsbg93cGFnZXMuYW1hem9uLmNvbYI0d3d3 > Lm0uYW1hem9uLmNvbYISd3d3LmNkb15hbWF6b24uY29tghN0ZXN0LXd3dy5hbWF6 > b24uY29tghJtcDNyZWNzLmFtYXpibi5jb22CFmtvbnJhZC10ZXN0LmFtYXpibi5j > b22CGHNob3AuYnVzaW5lc3MuYW1hem9uLmNvbTA+BgNVHSAEnzA1MDMGBmeBDAEC > ATApMCCGCCsGAQUFBwIBFhtodHRw0i8vd3d3LmRpZ2lJZXJ0LmNvbS9DUFMwDgYD > VR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgfEFBQcDAjB3BgNV > HR8EcDBuMDWgM6Axhi9odHRw0i8vY3JsMy5kaWdpY2VydC5jb20vRGlnaUNlcnRH > bG9iYWxDQUcyLmNybDA1oD0gMYYvaHR0cDovL2NybdQuZGlnaWNlcnQuY29tL0Rp > Z2lDZXJ0R2xvYmFsQ0FHMi5jcmwwdAYIKwYBBQUHAQEEaDBmMCQGCCsGAQUFBzAB > hhodHRw0i8vb2NzcC5kaWdpY2VydC5jb20wPgYIKwYBBQUHMAKGmh0dHA6Ly9j > YWNlcnRzLmRpZ2lJZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbENBRzIuY3J0MAwGA1Ud > EwEB/wQCMAAwggF9BgorBgEEAdZ5AgQCBIBbQSCAWkBZwB2ANdtfDRRp/V3wsfp > X9cAv/mCyTNaZeHQswFzF8DIxWl3AAABmhRnX5gAAAQDAEcwRQIhALy+XEa128nG > YnA/PEMwvCB+/Pg20Uq8UNA6ehC3RnZiAiBmndMjaXZmJEbSSFa9Zy4Ls13PuKA9 > y8yIFKkgwwfK8QB1AMIxflDFGaNF7n843rKQQevHwiFaIr9/1bWtdprZD1LNAAAB > mhRnX1YAAAQDAEYwRAIgTYr5VQEfk7RpQIC10tLWWF7GxNDVHaQHurk10fPKTvkC > IDY1MKj6KDsaAsWfwoyPQfp8iK37ZV1QvV0/hXsJqkyVAHY1E5Dh/rswe+B8xkk > JqgYZQHH0184AgE/cmd9VTcuGdgAAAGaFGdfZAAABAMRzBFAiEA6Y3TR6gpvN1b > qftSw2Y4V05LWoe/vfxQgwgLo/e8rQCIHZpk4qWcpk606X9VumJ7fVdLuD8I0yE > mgBYbzAN1MtzMA0GCSqGSIB3DQEBCwUAA4IBAQAmphQ3sGV4wbQshoV0PZiP9Zi > W3YxHU7F3i04Qbi9JEI/DvLY5mK+RU2RYr1ZlofQ2jkvQF86nCXQjGHPdXF/eSd6 > Y5M46kNDSomv9pN8aC9mMD+wDafwbAUa9zFdotJAPio3o8x4QejnbkK4nn9A4iXR > OFK5sMixgMDTBmqcA53JrUS7V+0x7uTQ+NSI0Y7D9fprm/xs1fnQHDiTe8PQ+bWT > m2A6HTYXDBxzTKHE80v2jjiMML0p8bsWau3Up6ruchpPIrfvcsjux8JE6KA8cC// > Ioy/6CZ5wgM0v2NCN0p2eze+wPPSvMLnA511bKBhWIPLsh50zm0G/10eqjNU > -----END CERTIFICATE----- > " > c0.pem [10/25/25]seed@VM:~/.../Labsetup$
```

**Saving the second certificate as c1.pem:**



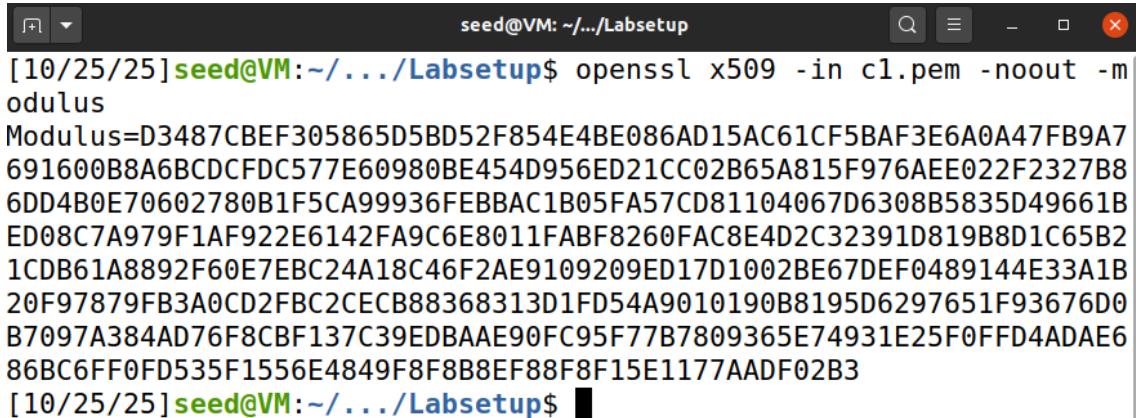
```
seed@VM: ~/.../Labsetup$ echo "-----BEGIN CERTIFICATE-----> MIIEizCCA30gAwIBAgIQDI7gyQ1qiRWIBAYe4kH5rzANBgkqhkiG9w0BAQsFADBh> MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLEXB3> d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBhbG9iYWwgUm9vdCBH> MjAeFw0xMzA4MDExMjAwMDBaFw0yODA4MDExMjAwMDBaMEQxCzAJBgNVBAYTA1VT> MRUwEwYDVQQKEwxEaWdpQ2VydCBjbmMxHjAcBgNVBAMTFURpZ2lDZXJ0IEDsb2Jh> bCBDQSBHMjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANNIfL7zBYZd> W9UvhU5L4IatFaxhz1uvPmoKR/uadpFgC4przc/cV35gmAvkVNlW7SHMARZagV+X> au4CLyMnuG3Us0cGAngLH1ypmTb+u6wbBfpXzYEQQGfWMItYNdSWYb70jHqXnxr5> IuYUL6nG6AEfq/gmD6y0TSwyOR2Bm40cZbIc22Gois9g5+vCSjhEbyrpEJIJ7RfR> ACvmfe8EiRR0M6GyD5eHn70gzS+8L0y4g2gxPR/VSpAQGQuBldYpd1H5NnbQtwl6> 0ErXb4y/E3w57bqukPyV93t4CTZedJMeJfD/1K2uaGvG/w/VNffVbkhJ+Pi474j4> 8V4Rd6rfArMCAwEAAs0CAVowggFWMBIGA1UdEwEB/wQIMAYBa8CAQAwDgYDVR0P> AQH/BAQDAgGGMDQGCCsGAQUFBwEBBCgwJjAkBggrBgEFBQcwAYYYaHR0cDovL29j> c3AuZGlnaWNlcnQuY29tMHsGA1UdHwR0MHlwN6A1oDOGMWh0dHA6Ly9jcmw0LmRp> Z2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwN6A1oDOGMWh0dHA6> Ly9jcmwzLmRpZ2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwPQYD> VR0gBDYwNDAyBgRVHSAAMCowKAYIKwYBBQUHAgeWHGh0dBz0i8vd3d3LmRpZ2lj> ZXJ0LmNvbS9DUFMwHQYDVR00BBYEFCRuKy3QapJRUSVpAqqaR6aJ50AgMB8GA1Ud> IwQYMBaAFE4iVCAYlebjbuYP+vq5Eu0GF485MA0GCSqGSIb3DQEBCwUA4IBAQAL> OYSR+ZfrqoGvh0la0JL84mxZvzbIRacxAxHhBsCsMsdaVSnaT0AC9aHes03ewPj2> dZ12uYf+QYB6z13jAMZbAuabeGLJ3LhimnftiQjXS8X9Q9ViIyfEBFltcT8jW+rZ> 8uckJ2/0LYDbлизkVIvP6hnZf1WZUXo0LRg9eFhSvGNoVwvdRLNXSmDmyHBwW4co> atc7T1JFGa8kBpJIERqlrqwYElesA8u49L3KJg6nwd3jM+/AVTANlVl0nAM2BvjA> jxSZnE0qnsHhfTuvcqdfuh0Wku4Z0BqYBvQ3lBetoxi6PrABDJXWKTUgNX31EGDk> 92hiHuwZ4STyhxs6QiA> -----END CERTIFICATE-----> " > c1.pem[10/25/25] seed@VM:~/.../Labsetup$
```

### Saving the third certificate as c2.pem screenshot:



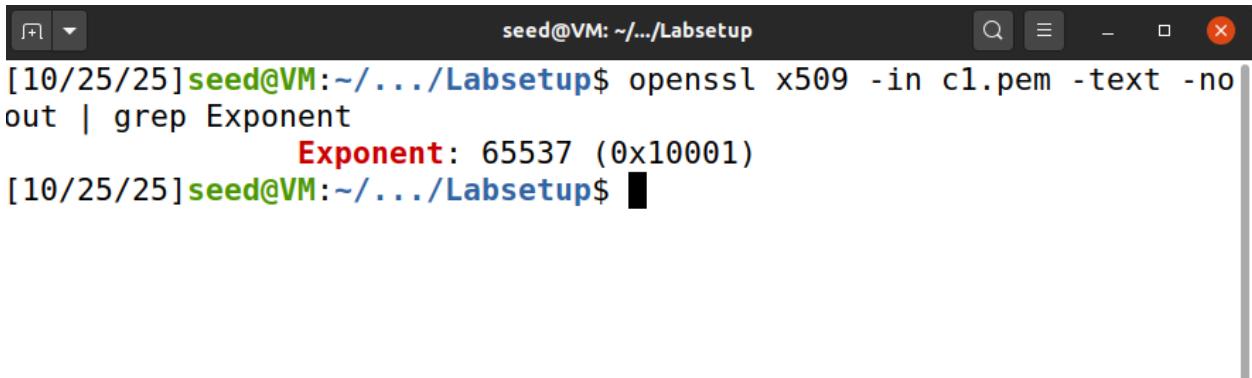
```
[10/25/25] seed@VM:~/.../Labsetup$ echo "-----BEGIN CERTIFICATE-----> MIIE3zCCA8egAwIBAgIQSysBFc3lx0gbPN3+3hEwnjANBgkqhkiG9w0BAQsFADCB> yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDLZlcm1TaWduLCBjbmMuMR8wHQYDVQQL> ExZWZXJpU2lnbiBUcnVzdCB0ZXr3b3JrMTowOAYDVQQLEzEoYkgMjAwNiBWZXJp> U2lnbiwgSw5jLiAtIEZvcibhdXRob3JpemVkiHVzzSBvbxm5MUUwQwYDVQQDEzxW> ZXJpU2lnbiBDGFzcAzIFB1YmxpYyBQcm1tYXJ5IENlcnRpZmljYXRpb24gQXV0> aG9yaXR5IC0gRzUwHhcNMTgwNDAzMDAwMDAwWhcNMjgwNDAYMjM10TU5WjBhMQsw> CQYDVQQGEwJVUzEVMBMGA1UEChMMRG1naUNlcnQgSw5jMRkwFwYDVQQLExB3d3cu> ZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBhbG9iYlwgUm9vdCBhmjCC> ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALs3zTTce2vJsmiQrUp1/0a6> IQoIjfUZVMn7iNzrvI6iZE8euarBhprz6wt6F4JJES6Ypp+1qoofuBUdSAFrFC3> nGMabDDc2h8Zsdce3v3X4MuUgzeu7B9DTt17LNK9LqUv5Km4rTrUmaS2JembawBg> kmD/TyFJGPdnkKthBpyP8rrpt0mSMmu181foXRvNjB2rlQSVsfM1LzbjSW3dd+P7> SUu0rFUHqY+Vs7Qju0xtRfD2qbKVMLT9TFWMJ0pXFHyCnc1zktMWSgYMjFDRjx4J> vheh5iHK/YPlELyDpQrEZyj2cxQUPUZ2w4cUiSE0Ta8PRQymSaG6u5zFsT0DKYUC> AwEAAaOCAScwggejMB0GA1UdDgQWBROILQgGJXm427mD/r6uRLtBheP0TAPBgNV> HRMBAf8EBTADAQH/MF8GA1UdIARYMFYwVAYEVROgADBMMCAGCCsGAQUFBwIBFhdo> dHRwczovL2Qu3ltY2IuY29tL2NwczaLBgrBgeFBQcCAjAZDBdodHRwczovL2Qu> c3ltY2IuY29tL3JwYTAvgNVHR8EKDAmMCSgIqAghh5odHRw0i8vcy5zeW1jYi5j> b20vcGNhMy1nNS5jcmwwDgYDVR0PAQH/BAQDAggGMC4GCCsGAQUFBwEBBCIwIDAe> BggrBgeFBQcwAYYSaHR0cDovL3Muc3ltY2QuY29tMB8GA1UdIwQYMBaAFH/TZafC> 3ey78DAJ80M5+gKvMzEzMA0GCSqGSIb3DQEBCwUAA4IBAQCOStidCZ2ebdzF1Jkd> ijPBIAYeed+Chn4dWDm0ptwghHtZ701DRFGLl0QQgXmcFBzTRCcGRgC2ZZDTjFYE> fs1mnDD3J+fKxm+cM6rR5JTLxP4j3v6k91tk1l3TtZhucff5qs+SNmpp/QQtneN> JoyWHawD8Kn1BZ0UA9sxrFeFdJz4uAz+ld6Fdm1ZtU5gMQXU0KHL6ZGry8+Qnax> 9A35U6m4eroyFSImrbteD0W5jkZtDuu3E33vx7Di7yLZV/Pfto/28wAcHxnukPk> CmMNKS5wFI5j7dStFFn6QWYpQX5SGuu7ywTg0E/nbsDXwqcUs0+elydJvwFEwtwX> b0ES> -----END CERTIFICATE-----> " > c2.pem[10/25/25] seed@VM:~/.../Labsetup$
```

### Finding Modulus from the c1.pem file screenshot:



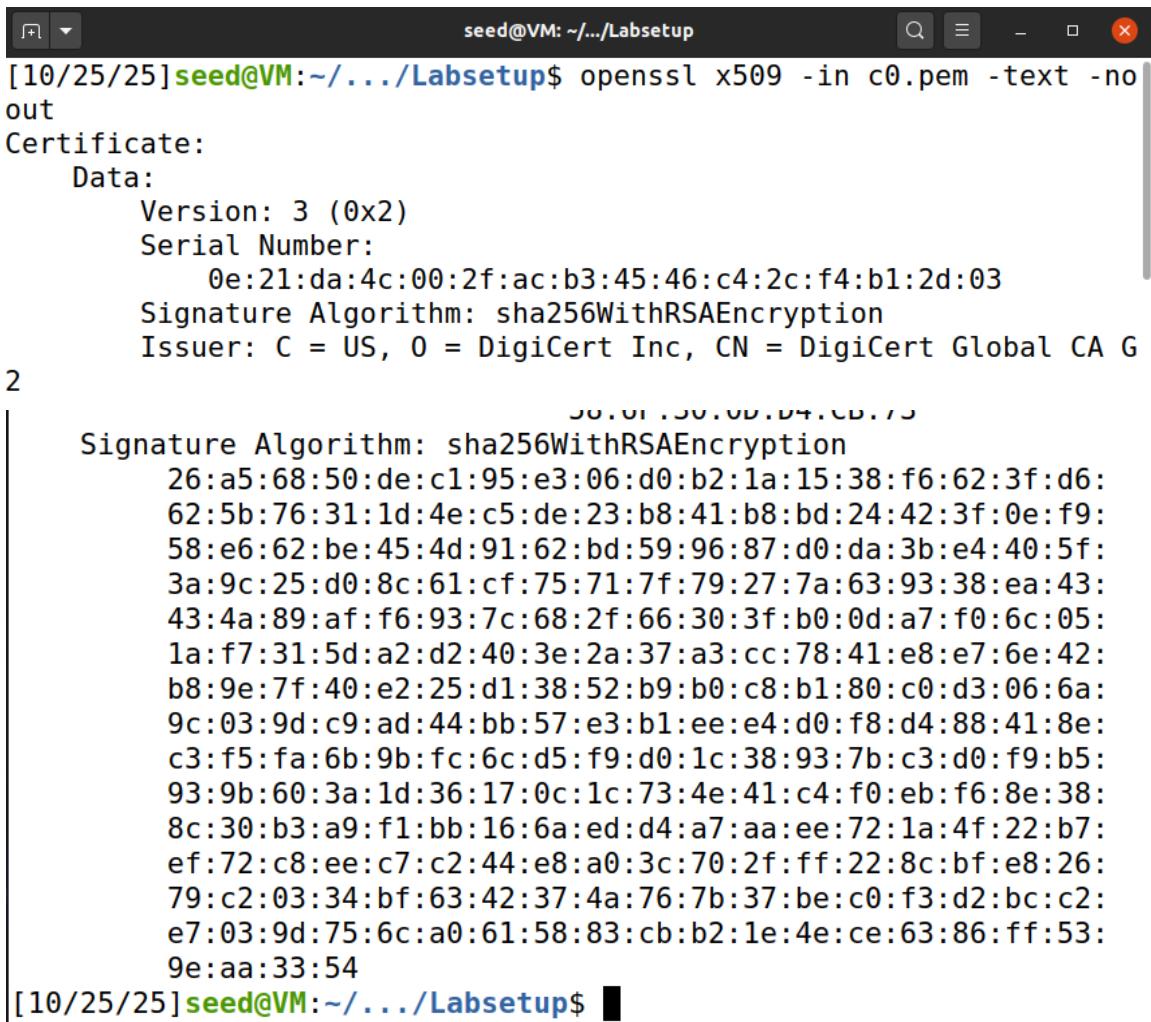
```
[10/25/25] seed@VM:~/.../Labsetup$ openssl x509 -in c1.pem -noout -modulusModulus=D3487CBEF305865D5BD52F854E4BE086AD15AC61CF5BAF3E6A0A47FB9A7691600B8A6BCDCFDC577E60980BE454D956ED21CC02B65A815F976AEE022F2327B86DD4B0E70602780B1F5CA99936FEBBAC1B05FA57CD81104067D6308B5835D49661BED08C7A979F1AF922E6142FA9C6E8011FABF8260FAC8E4D2C32391D819B8D1C65B21CDB61A8892F60E7EBC24A18C46F2AE9109209ED17D1002BE67DEF0489144E33A1B20F97879FB3A0CD2FBC2CECB88368313D1FD54A9010190B8195D6297651F93676D0B7097A384AD76F8CBF137C39EDBAE90FC95F77B7809365E74931E25F0FFD4ADAE686BC6FF0FD535F1556E4849F8F8B8EF88F8F15E1177AADF02B3[10/25/25] seed@VM:~/.../Labsetup$
```

Getting Exponent from the c1.pem file screenshot:



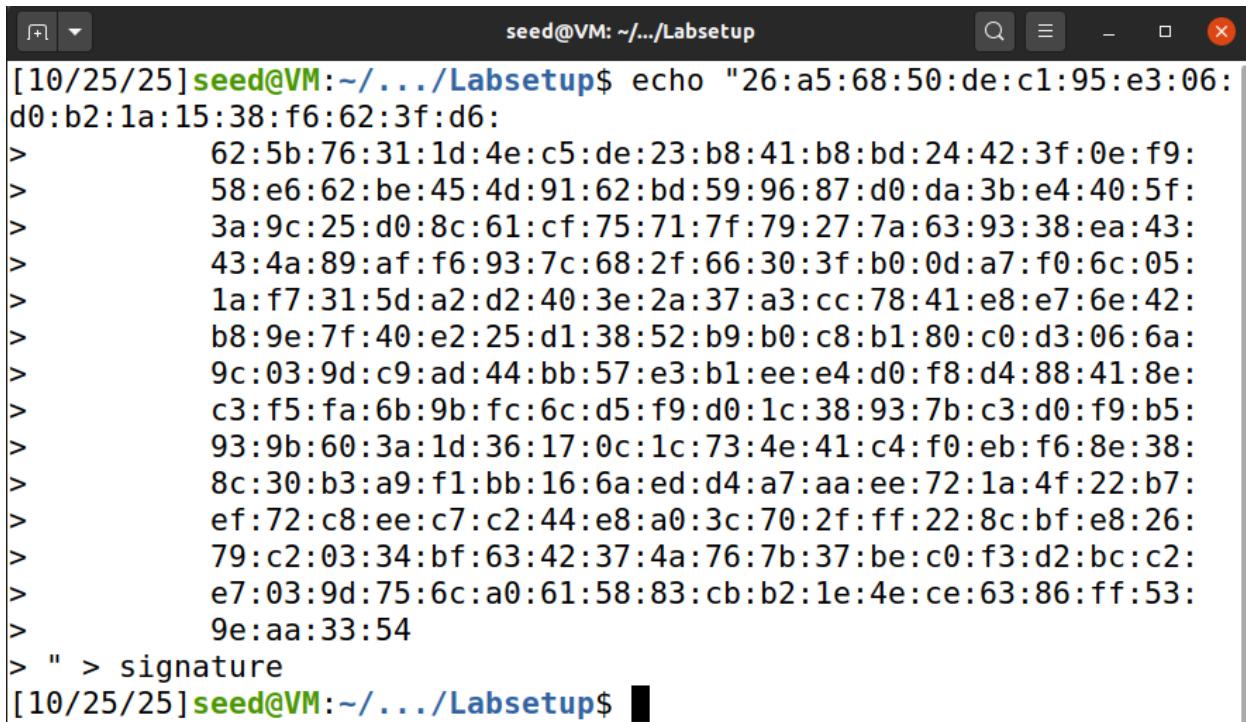
```
seed@VM: ~/.../Labsetup$ openssl x509 -in c1.pem -text -noout | grep Exponent
Exponent: 65537 (0x10001)
[10/25/25] seed@VM:~/.../Labsetup$
```

Extracting the signature from the server's certificate:



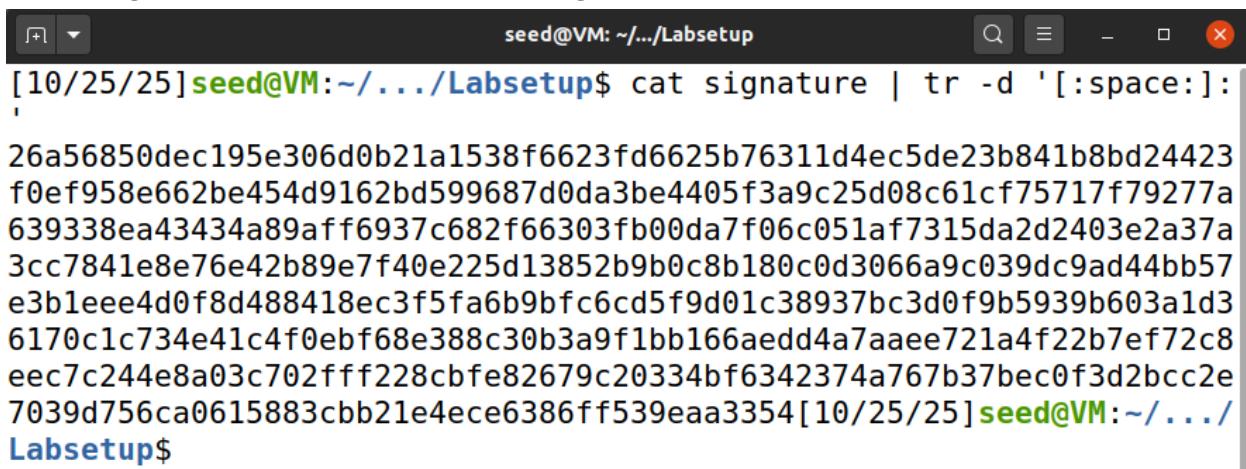
```
seed@VM: ~/.../Labsetup$ openssl x509 -in c0.pem -text -noout
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        0e:21:da:4c:00:2f:ac:b3:45:46:c4:2c:f4:b1:2d:03
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = US, O = DigiCert Inc, CN = DigiCert Global CA G
2
    Signature Algorithm: sha256WithRSAEncryption
        26:a5:68:50:de:c1:95:e3:06:d0:b2:1a:15:38:f6:62:3f:d6:
        62:5b:76:31:1d:4e:c5:de:23:b8:41:b8:bd:24:42:3f:0e:f9:
        58:e6:62:be:45:4d:91:62:bd:59:96:87:d0:da:3b:e4:40:5f:
        3a:9c:25:d0:8c:61:cf:75:71:7f:79:27:7a:63:93:38:ea:43:
        43:4a:89:af:f6:93:7c:68:2f:66:30:3f:b0:0d:a7:f0:6c:05:
        1a:f7:31:5d:a2:d2:40:3e:2a:37:a3:cc:78:41:e8:e7:6e:42:
        b8:9e:7f:40:e2:25:d1:38:52:b9:b0:c8:b1:80:c0:d3:06:6a:
        9c:03:9d:c9:ad:44:bb:57:e3:b1:ee:e4:d0:f8:d4:88:41:8e:
        c3:f5:fa:6b:9b:fc:6c:d5:f9:d0:1c:38:93:7b:c3:d0:f9:b5:
        93:9b:60:3a:1d:36:17:0c:1c:73:4e:41:c4:f0:eb:f6:8e:38:
        8c:30:b3:a9:f1:bb:16:6a:ed:d4:a7:aa:ee:72:1a:4f:22:b7:
        ef:72:c8:ee:c7:c2:44:e8:a0:3c:70:2f:ff:22:8c:bf:e8:26:
        79:c2:03:34:bf:63:42:37:4a:76:7b:37:be:c0:f3:d2:bc:c2:
        e7:03:9d:75:6c:a0:61:58:83:cb:b2:1e:4e:ce:63:86:ff:53:
        9e:aa:33:54
[10/25/25] seed@VM:~/.../Labsetup$
```

**Saving it in a signature file screenshot:**



```
seed@VM: ~/.../Labsetup$ echo "26:a5:68:50:de:c1:95:e3:06:  
d0:b2:1a:15:38:f6:62:3f:d6:  
>      62:5b:76:31:1d:4e:c5:de:23:b8:41:b8:bd:24:42:3f:0e:f9:  
>      58:e6:62:be:45:4d:91:62:bd:59:96:87:d0:da:3b:e4:40:5f:  
>      3a:9c:25:d0:8c:61:cf:75:71:7f:79:27:7a:63:93:38:ea:43:  
>      43:4a:89:af:f6:93:7c:68:2f:66:30:3f:b0:0d:a7:f0:6c:05:  
>      1a:f7:31:5d:a2:d2:40:3e:2a:37:a3:cc:78:41:e8:e7:6e:42:  
>      b8:9e:7f:40:e2:25:d1:38:52:b9:b0:c8:b1:80:c0:d3:06:6a:  
>      9c:03:9d:c9:ad:44:bb:57:e3:b1:ee:e4:d0:f8:d4:88:41:8e:  
>      c3:f5:fa:6b:9b:fc:6c:d5:f9:d0:1c:38:93:7b:c3:d0:f9:b5:  
>      93:9b:60:3a:1d:36:17:0c:1c:73:4e:41:c4:f0:eb:f6:8e:38:  
>      8c:30:b3:a9:f1:bb:16:6a:ed:d4:a7:aa:ee:72:1a:4f:22:b7:  
>      ef:72:c8:ee:c7:c2:44:e8:a0:3c:70:2f:ff:22:8c:bf:e8:26:  
>      79:c2:03:34:bf:63:42:37:4a:76:7b:37:be:c0:f3:d2:bc:c2:  
>      e7:03:9d:75:6c:a0:61:58:83:cb:b2:1e:4e:ce:63:86:ff:53:  
>      9e:aa:33:54  
> " > signature  
[10/25/25]seed@VM:~/.../Labsetup$
```

**Removing colons and spaces from the signature:**

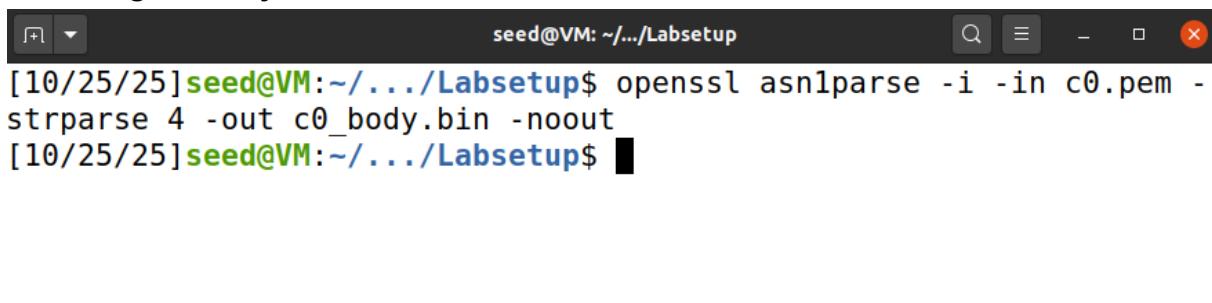


```
seed@VM: ~/.../Labsetup$ cat signature | tr -d '[:space:]':  
26a56850dec195e306d0b21a1538f6623fd6625b76311d4ec5de23b841b8bd24423  
f0ef958e662be454d9162bd599687d0da3be4405f3a9c25d08c61cf75717f79277a  
639338ea43434a89aff6937c682f66303fb00da7f06c051af7315da2d2403e2a37a  
3cc7841e8e76e42b89e7f40e225d13852b9b0c8b180c0d3066a9c039dc9ad44bb57  
e3b1eee4d0f8d488418ec3f5fa6b9bfc6cd5f9d01c38937bc3d0f9b5939b603a1d3  
6170c1c734e41c4f0ebf68e388c30b3a9f1bb166aedd4a7aaee721a4f22b7ef72c8  
eec7c244e8a03c702fff228cbfe82679c20334bf6342374a767b37bec0f3d2bcc2e  
7039d756ca0615883ccb21e4ece6386ff539eaa3354[10/25/25]seed@VM:~/.../  
Labsetup$
```

Extracting the body of the server's certificate:

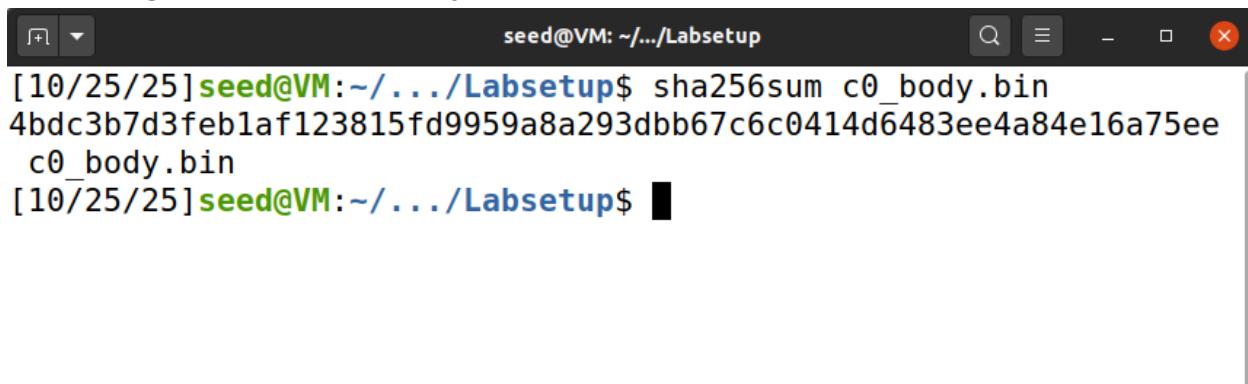
```
seed@VM:~/.../Labsetup$ openssl asn1parse -i -in c0.pem
 0:d=0  hl=4 l=2105 cons: SEQUENCE
  4:d=1  hl=4 l=1825 cons:  SEQUENCE
  8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
 10:d=3  hl=2 l=   1 prim:   INTEGER          :02
 13:d=2  hl=2 l=   16 prim:   INTEGER         :0E21DA4C002FACB34
546C42CF4B12D03
 31:d=2  hl=2 l=   13 cons:   SEQUENCE
 33:d=3  hl=2 l=    9 prim:   OBJECT          :sha256WithRSAEnc
ryption
 44:d=3  hl=2 l=    0 prim:   NULL
 46:d=2  hl=2 l=   68 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          :organizationNa
me
 70:d=5  hl=2 l=   12 prim:   PRINTABLESTRING :DigiCert Inc
 84:d=3  hl=2 l=   30 cons:   SET
 86:d=4  hl=2 l=   28 cons:   SEQUENCE
 88:d=5  hl=2 l=    3 prim:   OBJECT          :commonName
 93:d=5  hl=2 l=   21 prim:   PRINTABLESTRING :DigiCert Globa
l CA G2
116:d=2  hl=2 l=   30 cons:   SEQUENCE
118:d=3  hl=2 l=   13 prim:   UTCTIME        :251024000000Z
133:d=3  hl=2 l=   13 prim:   UTCTIME        :261020235959Z
148:d=2  hl=2 l=   25 cons:   SEQUENCE
150:d=3  hl=2 l=   23 cons:   SET
152:d=4  hl=2 l=   21 cons:   SEQUENCE
```

**Extracting the body of the certificate screenshot:**



```
seed@VM: ~/.../Labsetup
[10/25/25] seed@VM:~/.../Labsetup$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[10/25/25] seed@VM:~/.../Labsetup$
```

**Calculating the hash for the body screenshot:**



```
seed@VM: ~/.../Labsetup
[10/25/25] seed@VM:~/.../Labsetup$ sha256sum c0_body.bin
4bdc3b7d3feb1af123815fd9959a8a293dbb67c6c0414d6483ee4a84e16a75ee
c0_body.bin
[10/25/25] seed@VM:~/.../Labsetup$
```

**Padding the hash so it becomes 256 byte:**

---

```
prefix = "0001"
hash = "4bdc3b7d3feb1af123815fd9959a8a293dbb67c6c0414d6483ee4a84e
16a75ee"
A = "3031300D060960864801650304020105000420"
total_len = 256

pad_len = total_len - 1 - (len(A) + len(prefix) + len(hash)) // 2
padded_message = prefix + 'FF' * pad_len + '00' + A + hash

print(padded_message)
```

~

**Description and explanation:** The python code is used to pad the hash of the certificate's body so it can be correctly verified using RSA encryption. RSA requires the data to be a specific size, so we add extra padding to make it the right length(256 bytes). Without padding, the verification would fail because the data wouldn't match the format expected by RSA.

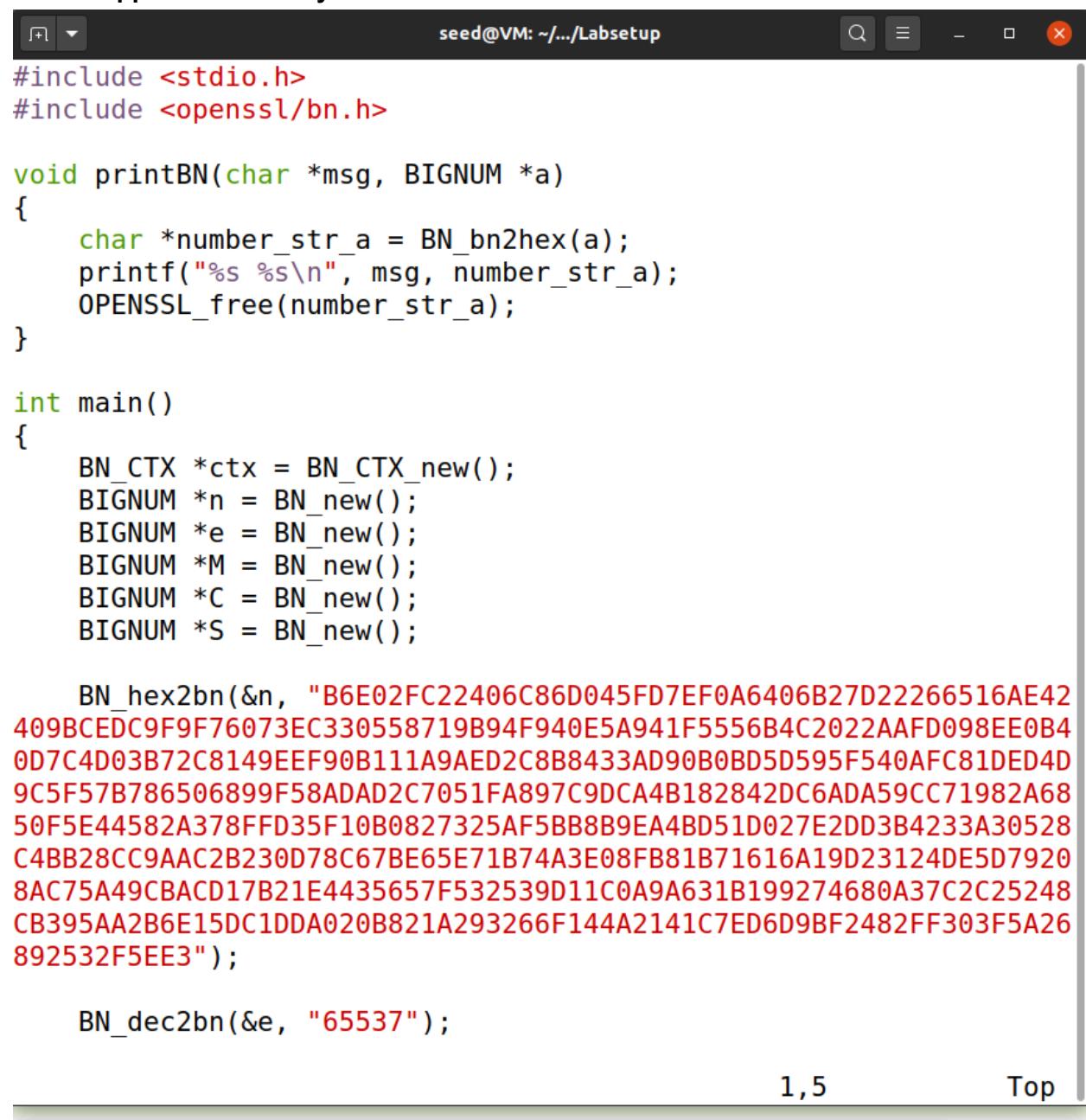
Padded hash output:

```
[10/25/25] seed@VM:~/.../Labsetup$ python3 pad_hash.py
0001FFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
FFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
FFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
FFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
FFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
FFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
FFFFFFFFFFFFFFFFFFF00000000000000000000000000000000
003031300D0609608648016503040201050004204bdc3b7
d3feb1af123815fd9959a8a293dbb67c6c0414d6483ee4a84e16a75ee
[10/25/25] seed@VM:~/.../Labsetup$
```

Creating file to verify:

```
[10/25/25] seed@VM:~/.../Labsetup$ vim verify.c
```

Code snippet for the verify.c file:



The screenshot shows a terminal window titled "seed@VM: ~/.../Labsetup". The code displayed is as follows:

```
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str_a = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str_a);
    OPENSSL_free(number_str_a);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();
    BIGNUM *S = BN_new();

    BN_hex2bn(&n, "B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42
409BCEDC9F9F76073EC330558719B94F940E5A941F5556B4C2022AAFD098EE0B4
0D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90B0BD5D595F540AFC81DED4D
9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC71982A68
50F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528
C4BB28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D7920
8AC75A49CBACD17B21E4435657F532539D11C0A9A631B199274680A37C2C25248
CB395AA2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF2482FF303F5A26
892532F5EE3");

    BN_dec2bn(&e, "65537");
```

The terminal window also displays the page numbers "1, 5" and "Top" at the bottom.

31.5

66%

47, 1

Bot

**Description and explanation:** The C code basically verifies the signature of the server certificate using RSA encryption. It begins by using the modulus n and exponent e, which are provided from the certificate's public key. The hash of the certificate's body is stored in M, and the signature from the certificate is stored in S. The code calculator C, which is the expected result if the signature is valid. If the computed value of C matches the original certificate hash M, it confirms the signature is valid meaning the certificate was not changed. In the screenshot below we can see that "valid signature" is printed hence the signature matches.

#### Compiling the verify.c file:

```
seed@VM: ~/.../Labsetup$ gcc verify.c -o verify -lcrypto  
[10/25/25] seed@VM:~/.../Labsetup$
```

#### Running the verify file:

```
seed@VM: ~/.../Labsetup$ ./verify  
Valid Signature!  
[10/25/25] seed@VM:~/.../Labsetup$
```