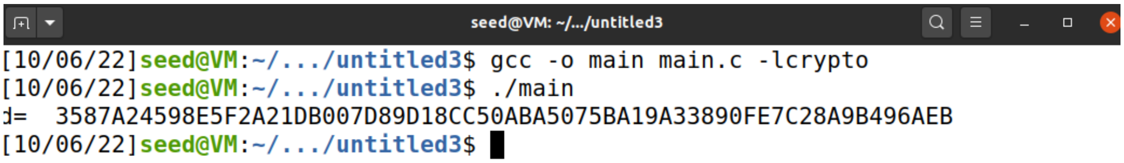
Task1

```
#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 128

//p = F7E75FDC469067FFDC4E847C51F452DF

//q = E85CED54AF57E53E092113E62F436F4F

//e = 0D88C3

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 *n = BN_new();

   BIGNUM *phi = BN_new();

   BIGNUM *p_sub_1 = BN_new();

   BIGNUM *q_sub_1 = BN_new();

   BIGNUM *res= BN_new();


   // Initialize p,q,e

   BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
```

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


    //sub p,q
    BN_sub(p_sub_1, p, BN_value_one());
    BN_sub(q_sub_1, q, BN_value_one());
    //phi(n)=(p-1)*(q-1)
    BN_mul(phi, p_sub_1, q_sub_1, ctx);
    BN_mul(n, p, q, ctx);


    BN_gcd(res, phi, e, ctx);
    if (!BN_is_one(res)){
        exit(0);
    }
    BN_mod_inverse(d, e, phi, ctx);
    printBN("d= ", d);
    return 0;
}
```



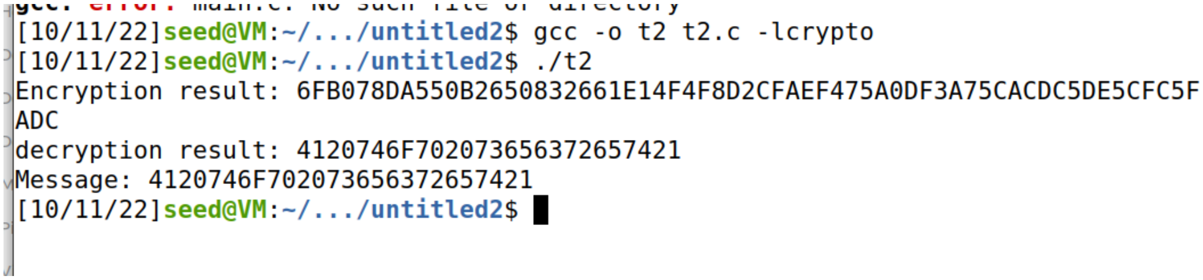
```
[10/06/22]seed@VM:~/.../untitled3$ gcc -o main main.c -lcrypto
[10/06/22]seed@VM:~/.../untitled3$ ./main
d=  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[10/06/22]seed@VM:~/.../untitled3$ 
```

Task 2

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128
//$ python -c ’print("A top secret!".encode("hex"))’
//4120746f702073656372657421
```

```
//n= DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
//e=010001 (this hex value equals to decimal 65537)
//M=A top secret!
//d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
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 *c = BN_new();
   BIGNUM *e = BN_new();
   BIGNUM *d = BN_new();
   BIGNUM *n = BN_new();
   BIGNUM *m = BN_new();
   BIGNUM *D = BN_new();

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

   BN_mod_exp(c, m, e, n, ctx);
   printBN("Encryption result:", c);
   BN_mod_exp(D, c, d, n, ctx);
```

```
    printBN("decryption result:", D);

    printBN("Message:", m);

    return 0;

}
```



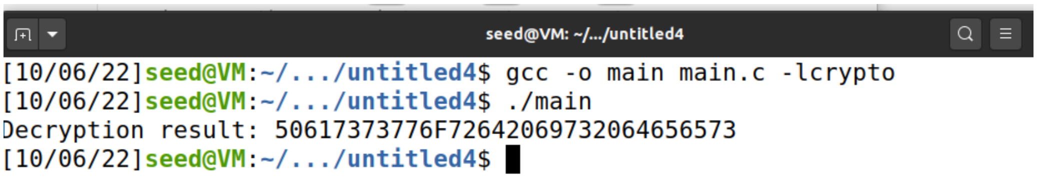
```
gcc: error: main.c: No such file or directory
[10/11/22]seed@VM:~/.../untitled2$ gcc -o t2 t2.c -lcrypto
[10/11/22]seed@VM:~/.../untitled2$ ./t2
Encryption result: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5F
ADC
decryption result: 4120746F702073656372657421
Message: 4120746F702073656372657421
[10/11/22]seed@VM:~/.../untitled2$ 
```

I use the given private key to verify the encyption message, the decyption result is same as the plaintext.

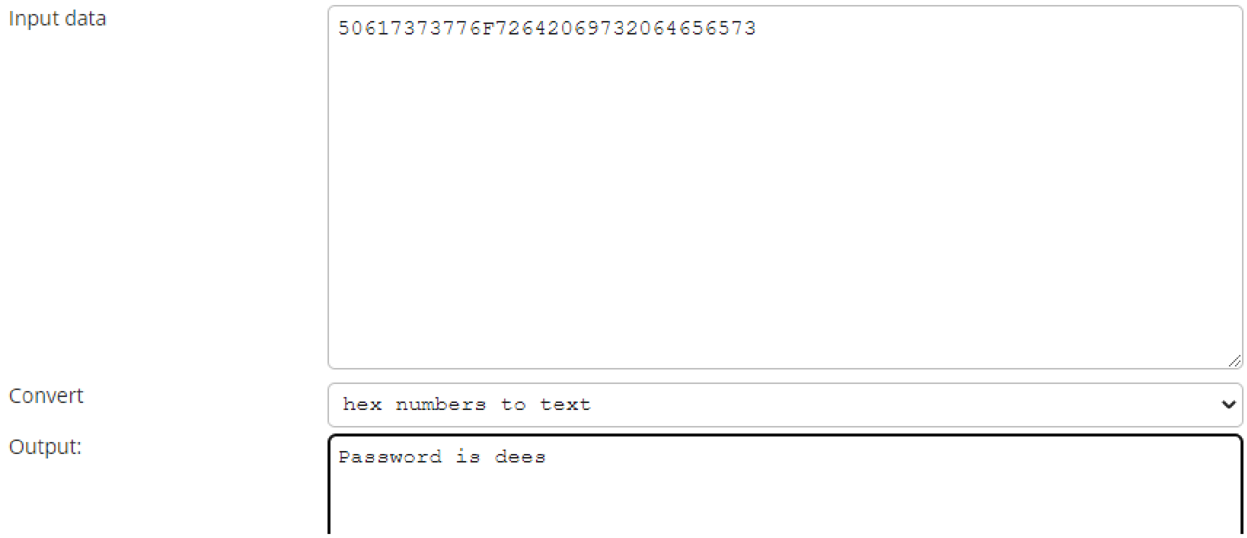
Task 3

```
#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 128

//$ python -c ’print("A top secret!".encode("hex"))’

//4120746f702073656372657421

//n= DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

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

//M=A top secret!

//c = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

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 *c = BN_new();

    BIGNUM *e = BN_new();

    BIGNUM *d = BN_new();

    BIGNUM *n = BN_new();

    BIGNUM *D = BN_new();


    // Initialize p,q,e

    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

    BN_hex2bn(&e, "010001");

    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");


    BN_mod_exp(D, c, d, n, ctx);

    printBN("Decryption result:", D);

    return 0;

}
```

//My system is python3 which cannot convert hex to text or text to hex directly, so I use online tool to convert the plaintext.



```
[10/06/22]seed@VM:~/.../untitled4$ gcc -o main main.c -lcrypto
[10/06/22]seed@VM:~/.../untitled4$ ./main
Decryption result: 50617373776F72642069732064656573
[10/06/22]seed@VM:~/.../untitled4$ 
```



Input data

```
50617373776F72642069732064656573
```

Convert: hex numbers to text

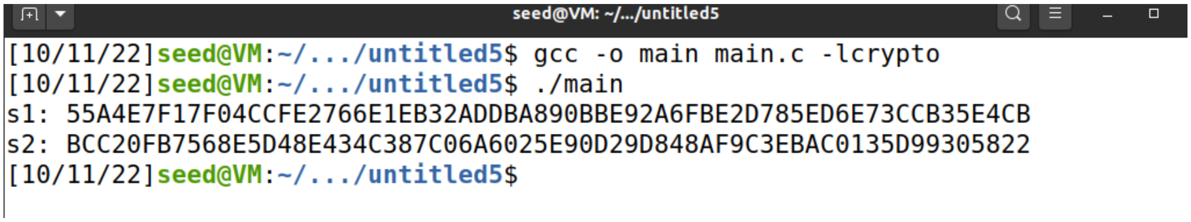
Output:

```
Password is dees
```

Task 4

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 128
//n= DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
//e=010001 (this hex value equals to decimal 65537)
//d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
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 *s1 = BN_new();
   BIGNUM *s2 = BN_new();
```

```
    BIGNUM *e = BN_new();

    BIGNUM *d = BN_new();

    BIGNUM *n = BN_new();

    BIGNUM *m1 = BN_new();

    BIGNUM *m2 = BN_new();

    BIGNUM *D = BN_new();


    // Initialize p,q,e

    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

    BN_hex2bn(&e, "010001");

    BN_hex2bn(&m1, "49206f776520796f752024323030302e"); // "I owe you $2000."

    BN_hex2bn(&m2, "49206f776520796f752024333030302e"); // "I owe you $3000."

    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");


    BN_mod_exp(s1, m1, d, n, ctx);

    BN_mod_exp(s2, m2, d, n, ctx);

    printBN("s1:", s1);

    printBN("s2:", s2);

    return 0;

}
```



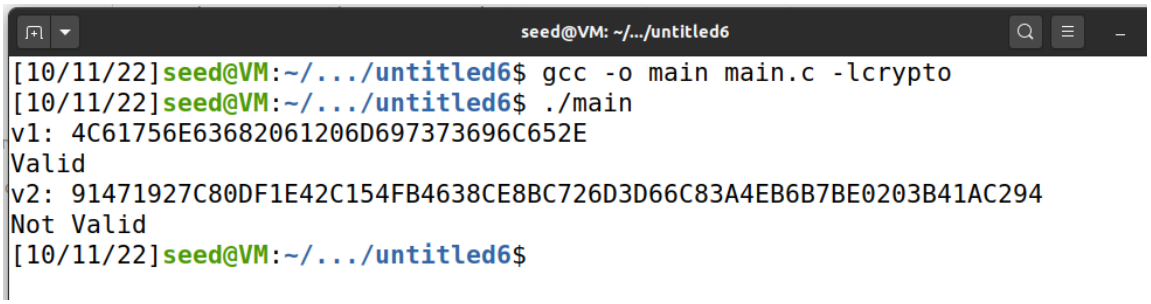
```
[10/11/22]seed@VM:~/.../untitled5$ gcc -o main main.c -lcrypto
[10/11/22]seed@VM:~/.../untitled5$ ./main
s1: 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
s2: BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
[10/11/22]seed@VM:~/.../untitled5$
```

S1 is the signature of “I owe you $2000.”, S2 is the signature of “I owe you $3000.” It can be noticed that the signatures are so different although the plaintext only have 1 number different.

Task 5

```
#include <stdio.h>
```

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

#define NBITS 128

//m = Launch a missile.

//s= 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F

//s1 = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F

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

//n =AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115

//$ python -c 'print("Launch a missile.".encode("hex"))'

//4c61756e63682061206d697373696c652e

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 *v1 = BN_new();

    BIGNUM *v2 = BN_new();

    BIGNUM *e = BN_new();

    BIGNUM *d = BN_new();

    BIGNUM *n = BN_new();

    BIGNUM *m1 = BN_new();

    BIGNUM *m2 = BN_new();

    BIGNUM *m = BN_new();


    // Initialize p,q,e

    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&m1,
"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    BN_hex2bn(&m2,
"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");//change that the
last byte of the signature changes from 2F to 3F
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&m, "4c61756e63682061206d697373696c652e");


    BN_mod_exp(v1, m1, e, n, ctx);
    BN_mod_exp(v2, m2, e, n, ctx);
    printBN("v1:", v1);
    if (BN_cmp(v1, m) == 0){
        printf("Valid\n");
    }else{
        printf("Not Valid\n");
    }
    printBN("v2:", v2);
        if (BN_cmp(v2, m) == 0){
        printf("Valid\n");
    }else{
        printf("Not Valid\n");
    }
    return 0;
}
```

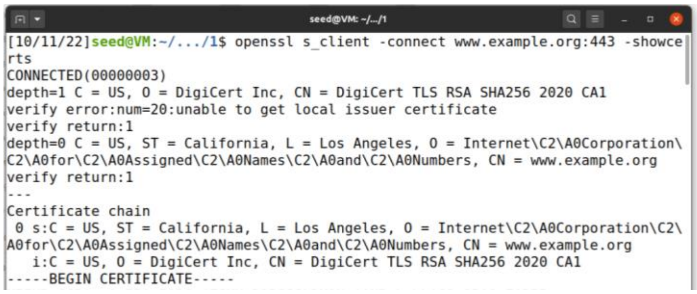


```
[10/11/22]seed@VM:~/.../untitled6$ gcc -o main main.c -lcrypto
[10/11/22]seed@VM:~/.../untitled6$ ./main
v1: 4C61756E63682061206D697373696C652E
Valid
v2: 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
Not Valid
[10/11/22]seed@VM:~/.../untitled6$
```

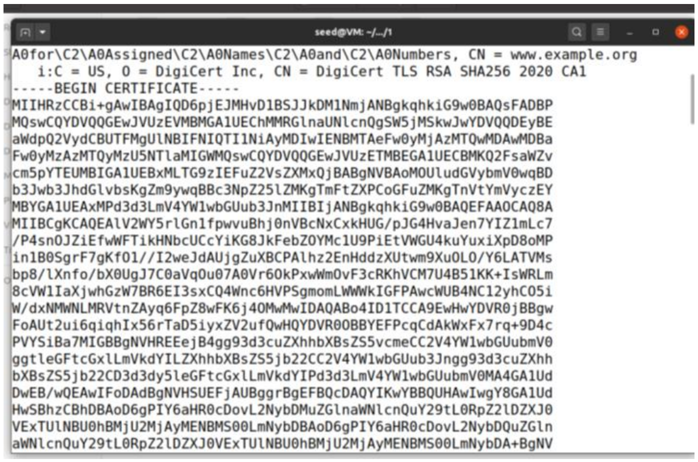
V1 is the validation of correct signature, V2 is the validation of incorrect signature that change the last byte of the signature changes from 2F to 3F. It can be noticed the result will be so different although there are a small different. Therefore, the user will reject the signature even so one byte different.

Task 6

Step 1 download the certification

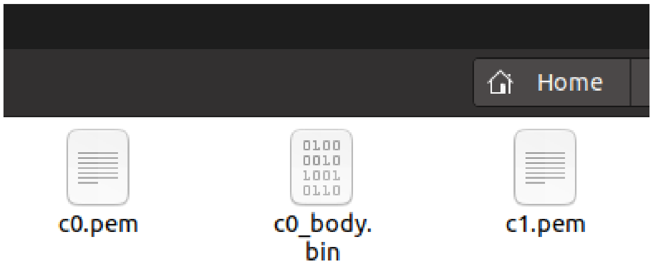


```
[10/11/22]seed@VM:~/.../1$ openssl s_client -connect www.example.org:443 -showce
rts
CONNECTED(00000003)
depth=1 C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, 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-----
```



```
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-----
MIIHRzCCBi+gAwIBAgIQD6pjEJMHvD1BSJJkDM1NmjANBgkqhkiG9w0BAQsFADBP
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMSkwJwYDVQQDEyBE
aWdpQ2VydCBUTFMgUlNBIFNIQTI1NiAyMDIwIENBMTAeFw0yMjAzMTQwMDAwMDBa
Fw0yMzAzMTQyMzU5NTlaMIGWMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZv
cm5pYTEUMBIGA1UEBxMLTG9zIEFuZ2VsZXMxQjBABgNVBAoMOUludGVybmV0wqBD
b3Jwb3JhdGlvbsKgZm9ywqBBc3NpZ25lZMKgTmFtZXPCoGFuZMKgTnVtYmVyczEY
MBYGA1UEAxMPd3d3LmV4YW1wbGUub3JnMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
MIIBCgKCAQEAlV2WY5rlGn1fpwvuBhj0nVBcNxCxkHUG/pJG4HvaJen7YIZ1mLc7
/P4snOJZiEfwWFTikHNbcUCcYiKG8JkFebZOYMc1U9PiEtVWGU4kuYuxiXpD8oMP
in1B0SgrF7gKfO1//I2weJdAUjgZuXBCPAlhz2EnHddzXUtwm9XuOLO/Y6LATVMs
bp8/lXnfo/bX0UgJ7C0aVqOu07A0Vr6OkPxwWmOvF3cRKhVCM7U4B51KK+IsWRLm
8cVW1IaXjwhGzW7BR6EI3sxCQ4Wnc6HVPSgmomLWWWkIGFPAwcWUB4NC12yhC05i
W/dxNMWNLMRVtnZAyq6FpZ8wFK6j4OMwMwIDAQABo4ID1TCCA9EwHwYDVR0jBBgw
FoAUt2ui6qiqhIx56rTaD5iyxZV2ufQwHQYDVR0OBBYEFPcqCdAkWxFx7rq+9D4c
PVYSiBa7MIGBBgNVHREEejB4gg93d3cuZXhhbXBsZS5vcmeCC2V4YW1wbGUubmV0
ggtleGFtcGxlLmVkdYILZXhhbXBsZS5jb22CC2V4YW1wbGUub3Jngg93d3cuZXhh
bXBsZS5jb22CD3d3dy5leGFtcGxlLmVkdYIPd3d3LmV4YW1wbGUubmV0MA4GA1Ud
DwEB/wQEAwIFoDAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwgY8GA1Ud
HwSBhzCBhDBAoD6gPIY6aHR0cDovL2NybDMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0
VExTUlNBU0hBMjU2MjAyMENBMS00LmNybDBAoD6gPIY6aHR0cDovL2NybDQuZGln
aWNlcnQuY29tL0RpZ2lDZXJ0VExTUlNBU0hBMjU2MjAyMENBMS00LmNybDA+BgNV
```

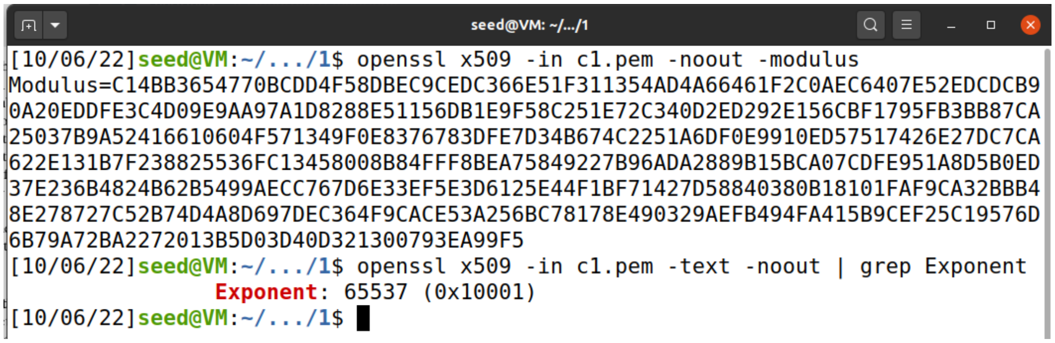
Copy and paste each of the to a file. Let us call the first one c0.pem and the second one c1.pem.



Step 2: extract (e,n)

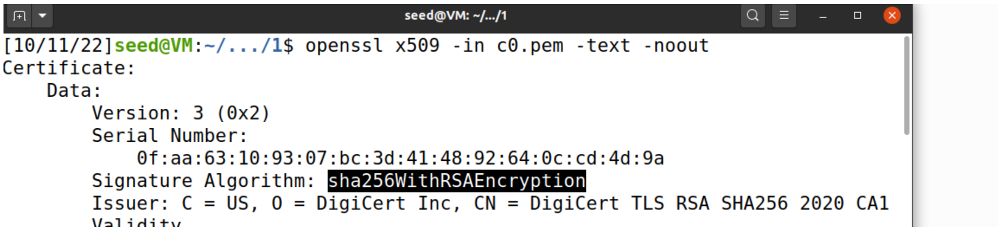
For modulus (n):

Find exponent (e):

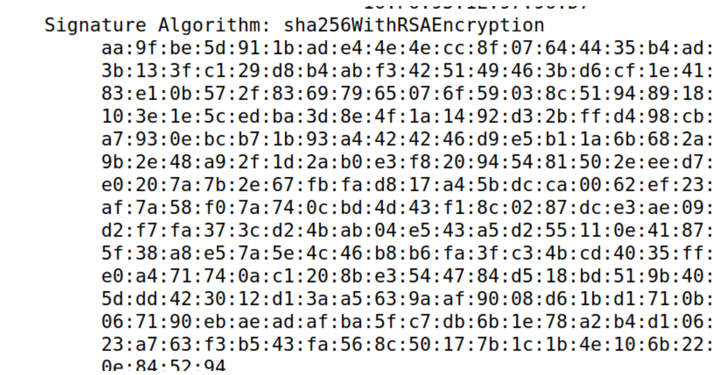


```
[10/06/22]seed@VM:~/.../1$ openssl x509 -in c1.pem -noout -modulus
Modulus=C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A66461F2C0AEC6407E52EDCDCB9
0A20EDDFE3C4D09E9AA97A1D8288E51156DB1E9F58C251E72C340D2ED292E156CBF1795FB3BB87CA
25037B9A52416610604F571349F0E8376783DFE7D34B674C2251A6DF0E9910ED57517426E27DC7CA
622E131B7F238825536FC13458008B84FFF8BEA75849227B96ADA2889B15BCA07CDFE951A8D5B0ED
37E236B4824B62B5499AECC767D6E33EF5E3D6125E44F1BF71427D58840380B18101FAF9CA32BBB4
8E278727C52B74D4A8D697DEC364F9CACE53A256BC78178E490329AEFB494FA415B9CEF25C19576D
6B79A72BA2272013B5D03D40D321300793EA99F5
[10/06/22]seed@VM:~/.../1$ openssl x509 -in c1.pem -text -noout | grep Exponent
                Exponent: 65537 (0x10001)
[10/06/22]seed@VM:~/.../1$ 
```

Step 3: extract the signature



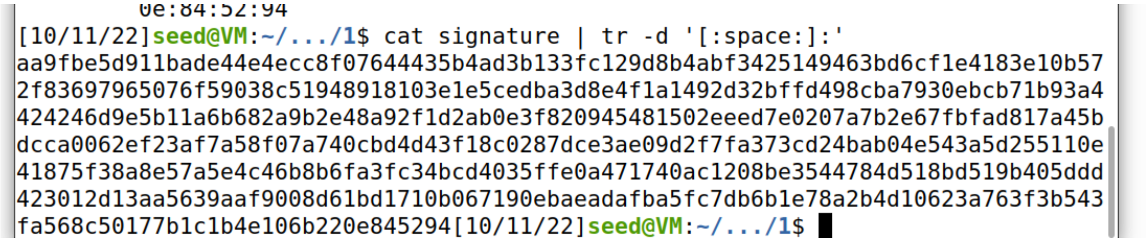
```
[10/11/22]seed@VM:~/.../1$ openssl x509 -in c0.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0f:aa:63:10:93:07:bc:3d:41:48:92:64:0c:cd:4d:9a
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
        Validity
```



```
    Signature Algorithm: sha256WithRSAEncryption
         aa:9f:be:5d:91:1b:ad:e4:4e:4e:cc:8f:07:64:44:35:b4:ad:
         3b:13:3f:c1:29:d8:b4:ab:f3:42:51:49:46:3b:d6:cf:1e:41:
         83:e1:0b:57:2f:83:69:79:65:07:6f:59:03:8c:51:94:89:18:
         10:3e:1e:5c:ed:ba:3d:8e:4f:1a:14:92:d3:2b:ff:d4:98:cb:
         a7:93:0e:bc:b7:1b:93:a4:42:42:46:d9:e5:b1:1a:6b:68:2a:
         9b:2e:48:a9:2f:1d:2a:b0:e3:f8:20:94:54:81:50:2e:ee:d7:
         e0:20:7a:7b:2e:67:fb:fa:d8:17:a4:5b:dc:ca:00:62:ef:23:
         af:7a:58:f0:7a:74:0c:bd:4d:43:f1:8c:02:87:dc:e3:ae:09:
         d2:f7:fa:37:3c:d2:4b:ab:04:e5:43:a5:d2:55:11:0e:41:87:
         5f:38:a8:e5:7a:5e:4c:46:b8:b6:fa:3f:c3:4b:cd:40:35:ff:
         e0:a4:71:74:0a:c1:20:8b:e3:54:47:84:d5:18:bd:51:9b:40:
         5d:dd:42:30:12:d1:3a:a5:63:9a:af:90:08:d6:1b:d1:71:0b:
         06:71:90:eb:ae:ad:af:ba:5f:c7:db:6b:1e:78:a2:b4:d1:06:
         23:a7:63:f3:b5:43:fa:56:8c:50:17:7b:1c:1b:4e:10:6b:22:
         0e:84:52:94
```

Copy this text to file “Signature”

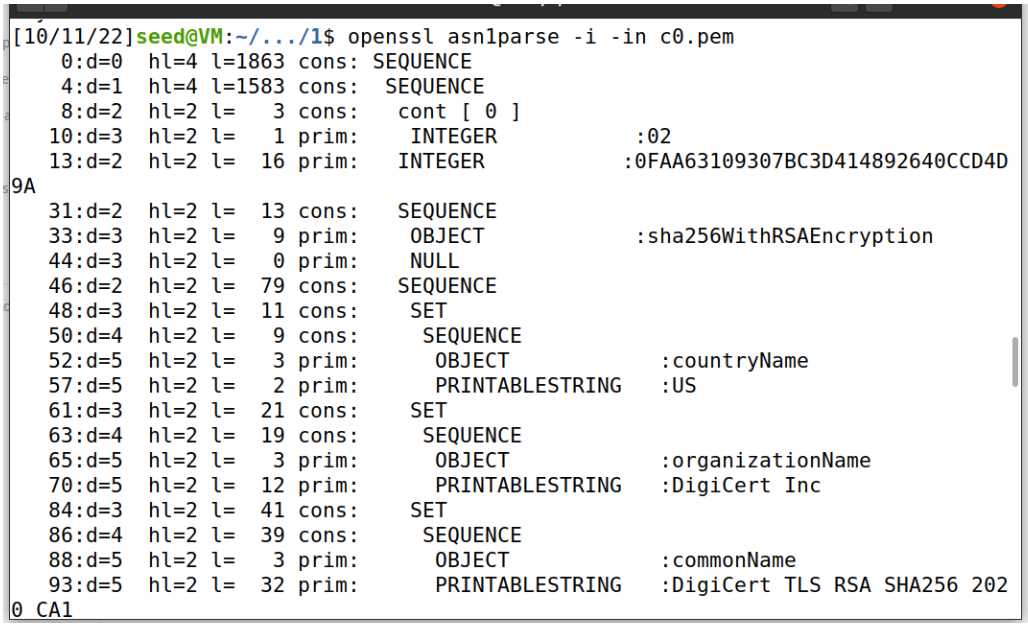
Remove the space and column



```
        0e:84:52:94
[10/11/22]seed@VM:~/.../1$ cat signature | tr -d '[:space:]:'
aa9fbe5d911bade44e4ecc8f07644435b4ad3b133fc129d8b4abf3425149463bd6cf1e4183e10b57
2f83697965076f59038c51948918103e1e5cedba3d8e4f1a1492d32bffd498cba7930ebcb71b93a4
424246d9e5b11a6b682a9b2e48a92f1d2ab0e3f820945481502eeed7e0207a7b2e67fbfad817a45b
dcca0062ef23af7a58f07a740cbd4d43f18c0287dce3ae09d2f7fa373cd24bab04e543a5d255110e
41875f38a8e57a5e4c46b8b6fa3fc34bcd4035ffe0a471740ac1208be3544784d518bd519b405ddd
423012d13aa5639aaf9008d61bd1710b067190ebaeadafba5fc7db6b1e78a2b4d10623a763f3b543
fa568c50177b1c1b4e106b220e845294[10/11/22]seed@VM:~/.../1$
```

Step 4:

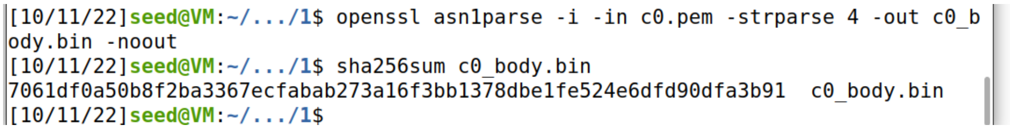
Extract the body of the server's certificate



```
[10/11/22]seed@VM:~/.../1$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1863 cons: SEQUENCE
    4:d=1  hl=4 l=1583 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           :0FAA63109307BC3D414892640CCD4D
9A
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l=  79 cons:   SEQUENCE
   48:d=3  hl=2 l=  11 cons:    SET
   50:d=4  hl=2 l=   9 cons:     SEQUENCE
   52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   61:d=3  hl=2 l=  21 cons:    SET
   63:d=4  hl=2 l=  19 cons:     SEQUENCE
   65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   70:d=5  hl=2 l=  12 prim:      PRINTABLESTRING   :DigiCert Inc
   84:d=3  hl=2 l=  41 cons:    SET
   86:d=4  hl=2 l=  39 cons:     SEQUENCE
   88:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
   93:d=5  hl=2 l=  32 prim:      PRINTABLESTRING   :DigiCert TLS RSA SHA256 202
0 CA1
```

The offset is 4, so:

Calculate the hash



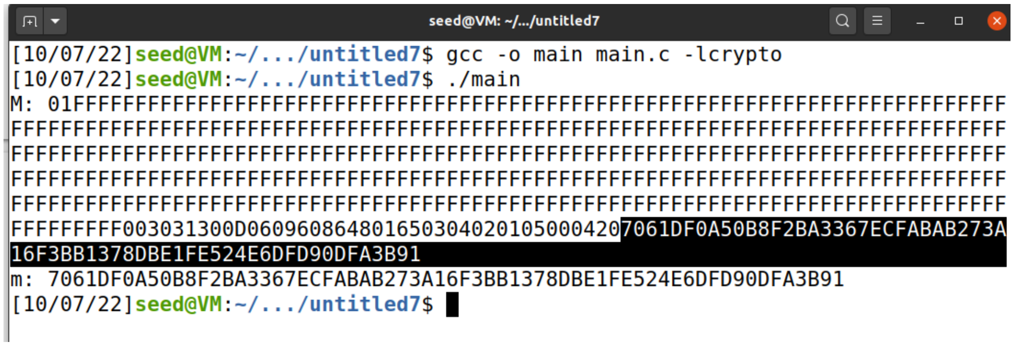
```
[10/11/22]seed@VM:~/.../1$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_b
ody.bin -noout
[10/11/22]seed@VM:~/.../1$ sha256sum c0_body.bin
7061df0a50b8f2ba3367ecfabab273a16f3bb1378dbe1fe524e6dfd90dfa3b91  c0_body.bin
[10/11/22]seed@VM:~/.../1$
```

Verify the signature

```
#include <stdio.h>

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

```
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 *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *m = BN_new();
    BIGNUM *s = BN_new();
    BIGNUM *M1 = BN_new();

    BN_hex2bn(&n,
"C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A66461F2C0AEC6407E52EDCDCB90A20EDDF
E3C4D09E9AA97A1D8288E51156DB1E9F58C251E72C340D2ED292E156CBF1795FB3BB87CA25037B9A52
416610604F571349F0E8376783DFE7D34B674C2251A6DF0E9910ED57517426E27DC7CA622E131B7F23
8825536FC13458008B84FFF8BEA75849227B96ADA2889B15BCA07CDFE951A8D5B0ED37E236B4824B62
B5499AECC767D6E33EF5E3D6125E44F1BF71427D58840380B18101FAF9CA32BBB48E278727C52B74D4
A8D697DEC364F9CACE53A256BC78178E490329AEFB494FA415B9CEF25C19576D6B79A72BA2272013B5
D03D40D321300793EA99F5"); // modulus n
    BN_hex2bn(&e, "010001"); // exponent e
    BN_hex2bn(&m,
"7061df0a50b8f2ba3367ecfabab273a16f3bb1378dbe1fe524e6dfd90dfa3b91");//certification
BN_hex2bn(&s,"aa9fbe5d911bade44e4ecc8f07644435b4ad3b133fc129d8b4abf3425149463bd6cf1e418
3e10b572f83697965076f59038c51948918103e1e5cedba3d8e4f1a1492d32bffd498cba7930ebcb71b93a
4424246d9e5b11a6b682a9b2e48a92f1d2ab0e3f820945481502eeed7e0207a7b2e67fbfad817a45bdcca0
062ef23af7a58f07a740cbd4d43f18c0287dce3ae09d2f7fa373cd24bab04e543a5d255110e41875f38a8e5
7a5e4c46b8b6fa3fc34bcd4035ffe0a471740ac1208be3544784d518bd519b405ddd423012d13aa5639aaf
9008d61bd1710b067190ebaeadafba5fc7db6b1e78a2b4d10623a763f3b543fa568c50177b1c1b4e106b2
20e845294");//signature
```

```
    BN_mod_exp(M1, s, e, n, ctx); //verify siganature

    printBN("M:", M1);

    printBN("m:", m);

    return 0;

}
```



```
[10/07/22]seed@VM:~/.../untitled7$ gcc -o main main.c -lcrypto
[10/07/22]seed@VM:~/.../untitled7$ ./main
M: 01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFF003031300D0609608648016503040201050004207061DF0A50B8F2BA3367ECFABAB273A
16F3BB1378DBE1FE524E6DFD90DFA3B91
m: 7061DF0A50B8F2BA3367ECFABAB273A16F3BB1378DBE1FE524E6DFD90DFA3B91
[10/07/22]seed@VM:~/.../untitled7$ 
```

The last bytes are same as the certification, which means the signature is verified.