11811407Lab10

Name: 黎诗龙

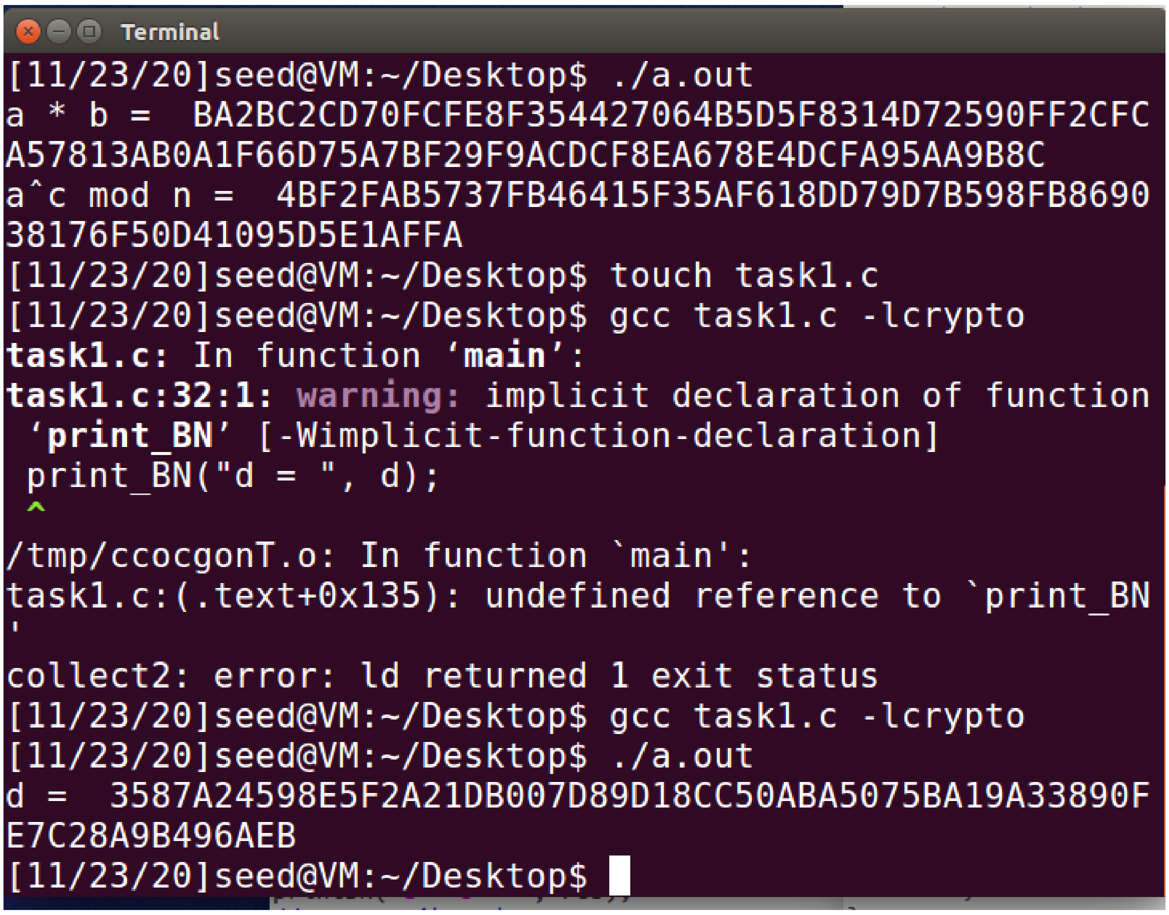
SID: 11811407

Task1

$$\phi(n) = (p-1)(q-1)$$
$$ed \equiv 1 \mod \phi(n)$$
$$\rightarrow d \equiv e^{-1} \mod \phi(n)$$

```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *p = BN_new();
BIGNUM *p_ = BN_new();//p-1
BIGNUM *q = BN_new();
BIGNUM *q_ = BN_new();// q-1
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
BIGNUM *phi = BN_new();

BN_hex2bn(&p,"F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q,"E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e,"0D88C3");
BN_hex2bn(&p_,"F7E75FDC469067FFDC4E847C51F452DE");//p-1
BN_hex2bn(&q_,"E85CED54AF57E53E092113E62F436F4E");//q-1

BN_mul(phi,p_,q_,ctx);
BN_mod_inverse(d,e,phi,ctx);
print_BN("d = ", d);
```

From the equations we can write the codes, then we can get that d (private key):



```
[11/23/20]seed@VM:~/Desktop$ ./a.out
a * b =  BA2BC2CD70FCFE8F354427064B5D5F8314D72590FF2CFC
A57813AB0A1F66D75A7BF29F9ACDCF8EA678E4DCFA95AA9B8C
a^c mod n =  4BF2FAB5737FB46415F35AF618DD79D7B598FB8690
38176F50D41095D5E1AFFA
[11/23/20]seed@VM:~/Desktop$ touch task1.c
[11/23/20]seed@VM:~/Desktop$ gcc task1.c -lcrypto
task1.c: In function ‘main’:
task1.c:32:1: warning: implicit declaration of function
 ‘print_BN’ [-Wimplicit-function-declaration]
 print_BN("d = ", d);
 ^

/tmp/ccocgonT.o: In function `main':
task1.c:(.text+0x135): undefined reference to `print_BN
'
collect2: error: ld returned 1 exit status
[11/23/20]seed@VM:~/Desktop$ gcc task1.c -lcrypto
[11/23/20]seed@VM:~/Desktop$ ./a.out
d =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890F
E7C28A9B496AEB
[11/23/20]seed@VM:~/Desktop$ 
```

Task2

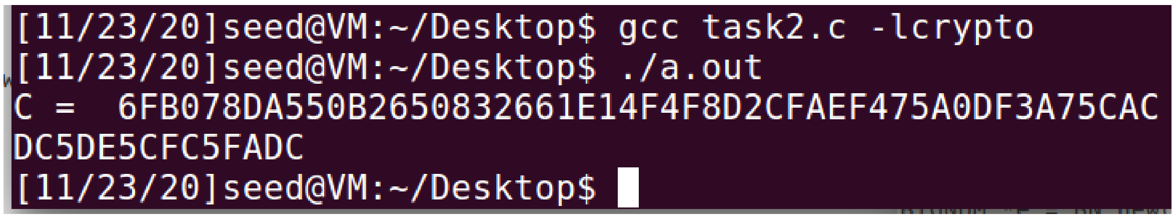
From the equation

$$C \equiv M^e \mod n$$

we can write the codes

```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *M = BN_new();
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
BIGNUM *C = BN_new();

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

//M^e mod n

BN_mod_exp(C,M,e,n,ctx);
print_BN("C = ", C);
```

Then we can get the result C:


```
[11/23/20]seed@VM:~/Desktop$ gcc task2.c -lcrypto
[11/23/20]seed@VM:~/Desktop$ ./a.out
C =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CAC
DC5DE5CFC5FADC
[11/23/20]seed@VM:~/Desktop$
```

Task3

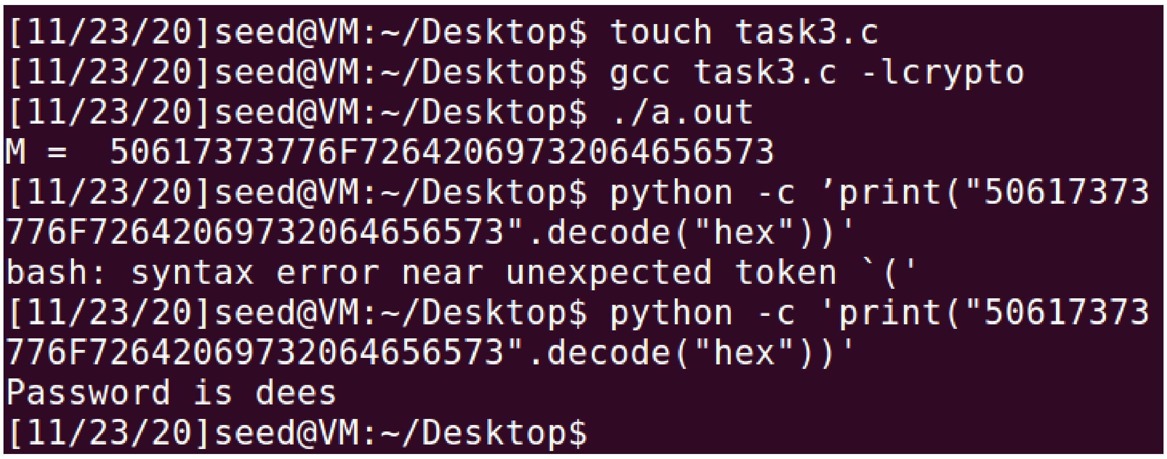
From the equation:

$$M \equiv C^d \mod n$$

We can write this codes:

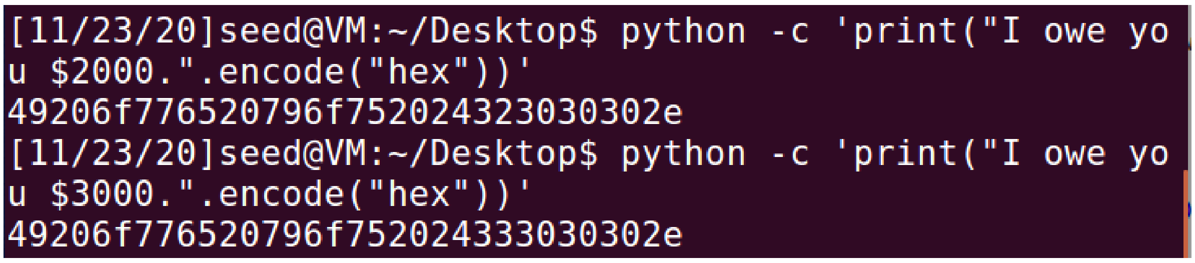
```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *M = BN_new();
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
BIGNUM *C = BN_new();

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

BN_mod_exp(M,C,d,n,ctx);
print_BN("M = ", M);
```

And we get this result:


```
[11/23/20]seed@VM:~/Desktop$ touch task3.c
[11/23/20]seed@VM:~/Desktop$ gcc task3.c -lcrypto
[11/23/20]seed@VM:~/Desktop$ ./a.out
M =  50617373776F72642069732064656573
[11/23/20]seed@VM:~/Desktop$ python -c ’print("50617373
776F72642069732064656573".decode("hex"))'
bash: syntax error near unexpected token `('
[11/23/20]seed@VM:~/Desktop$ python -c 'print("50617373
776F72642069732064656573".decode("hex"))'
Password is dees
[11/23/20]seed@VM:~/Desktop$
```

Task4

First encoding the message, we get



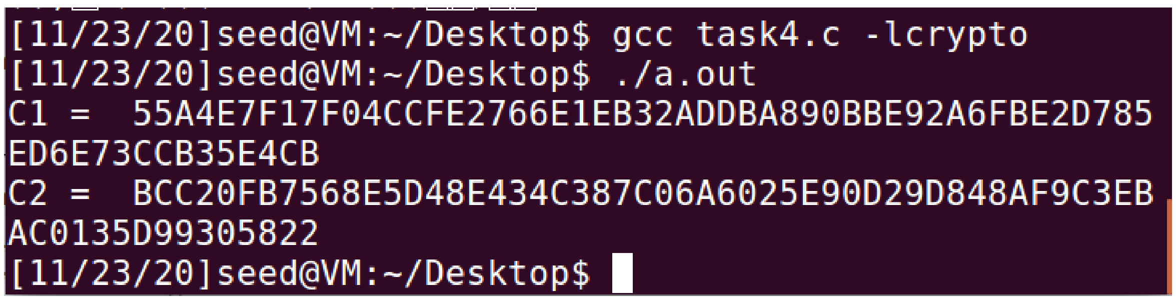
```
[11/23/20]seed@VM:~/Desktop$ python -c 'print("I owe yo
u $2000.".encode("hex"))'
49206f776520796f75202432303030302e
[11/23/20]seed@VM:~/Desktop$ python -c 'print("I owe yo
u $3000.".encode("hex"))'
49206f776520796f75202433303030302e
```

We use **private key** to sign the message

we can write the codes

```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
BIGNUM *M1 = BN_new();
BIGNUM *M2 = BN_new();
BIGNUM *C1 = BN_new();
BIGNUM *C2 = BN_new();

BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e,"010001");
BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&M1,"49206f776520796f75202432303030302e");
BN_hex2bn(&M2,"49206f776520796f75202433303030302e");

BN_mod_exp(C1,M1,d,n,ctx);
print_BN("C1 = ", C1);
BN_mod_exp(C2,M2,d,n,ctx);
print_BN("C2 = ", C2);
```

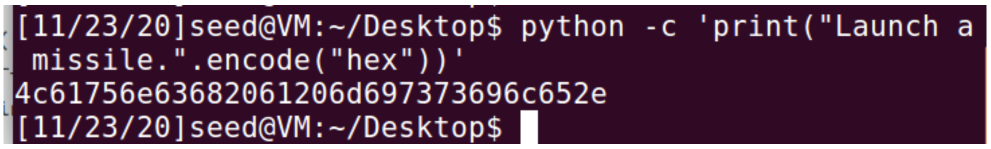
And get the result:



```
[11/23/20]seed@VM:~/Desktop$ gcc task4.c -lcrypto
[11/23/20]seed@VM:~/Desktop$ ./a.out
C1 =  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785
ED6E73CCB35E4CB
C2 =  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EB
AC0135D99305822
[11/23/20]seed@VM:~/Desktop$
```

Task5

First encoding the message:



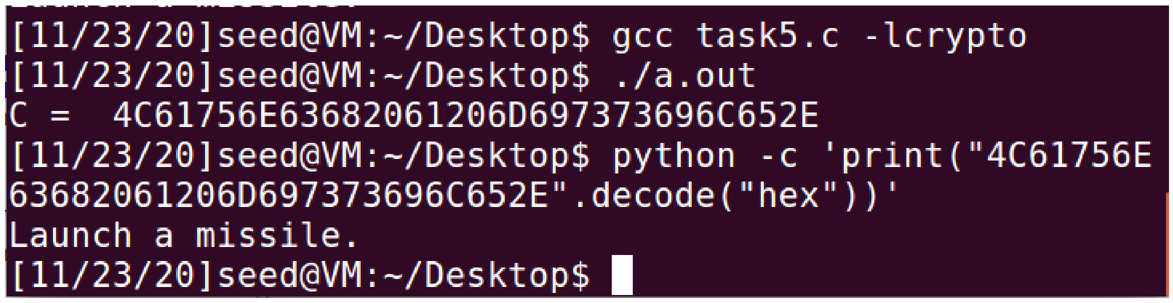
```
[11/23/20]seed@VM:~/Desktop$ python -c 'print("Launch a
 missile.".encode("hex"))'
4c61756e63682061206d697373696c652e
[11/23/20]seed@VM:~/Desktop$
```

We use public key ***e*** to verify the signature, so we write the codes:

```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
```

```c
    BIGNUM *S = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();

    BN_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_hex2bn(&e,"010001");
    BN_hex2bn(&S,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    BN_hex2bn(&M,"4120746f702073656372657421");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    BN_mod_exp(C,S,e,n,ctx);
    print_BN("C = ", C);
```

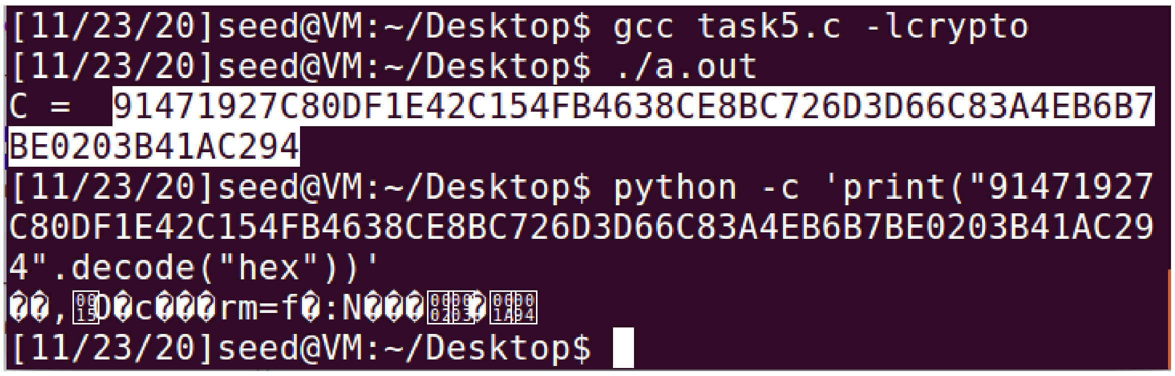
And we get such result:



```
[11/23/20]seed@VM:~/Desktop$ gcc task5.c -lcrypto
[11/23/20]seed@VM:~/Desktop$ ./a.out
C =  4C61756E63682061206D697373696C652E
[11/23/20]seed@VM:~/Desktop$ python -c 'print("4C61756E
63682061206D697373696C652E".decode("hex"))'
Launch a missile.
[11/23/20]seed@VM:~/Desktop$ 
```

This verifies that it is from Alice.

And if the `2F` becomes `3F`, then it becomes



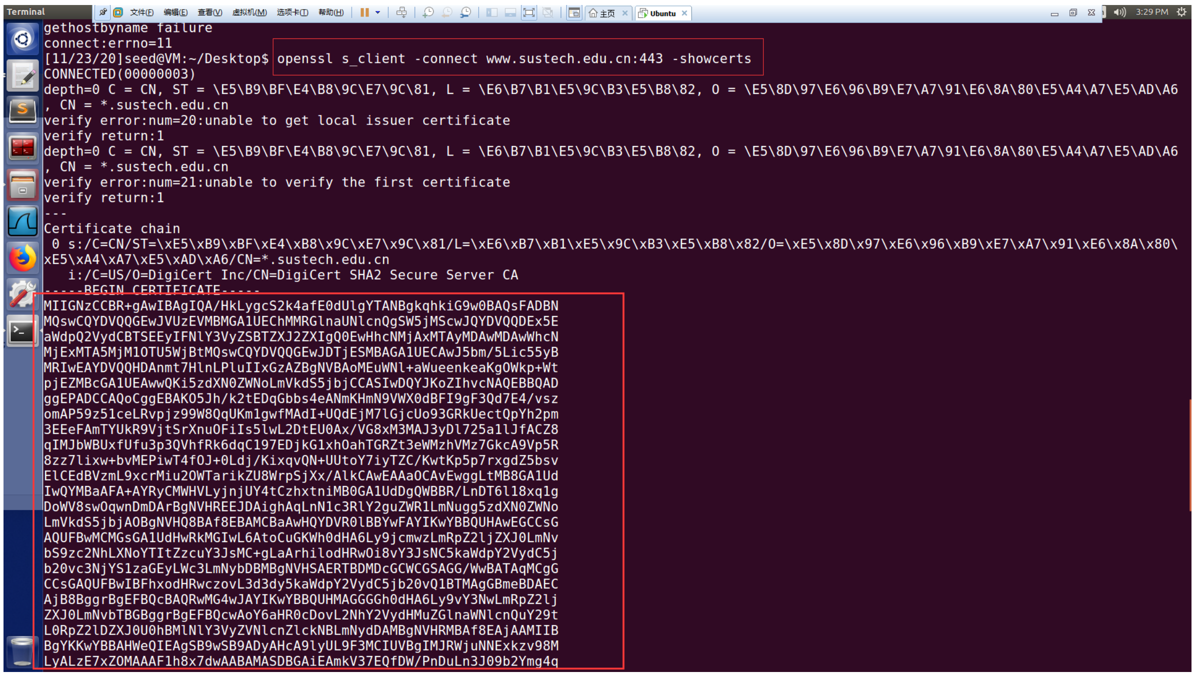
```
[11/23/20]seed@VM:~/Desktop$ gcc task5.c -lcrypto
[11/23/20]seed@VM:~/Desktop$ ./a.out
C =  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7
BE0203B41AC294
[11/23/20]seed@VM:~/Desktop$ python -c 'print("91471927
C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC29
4".decode("hex"))'
��,▯D�c���rm=f�:N���▯▯�▯▯
[11/23/20]seed@VM:~/Desktop$ 
```

It corrupts.

Task6

Step1

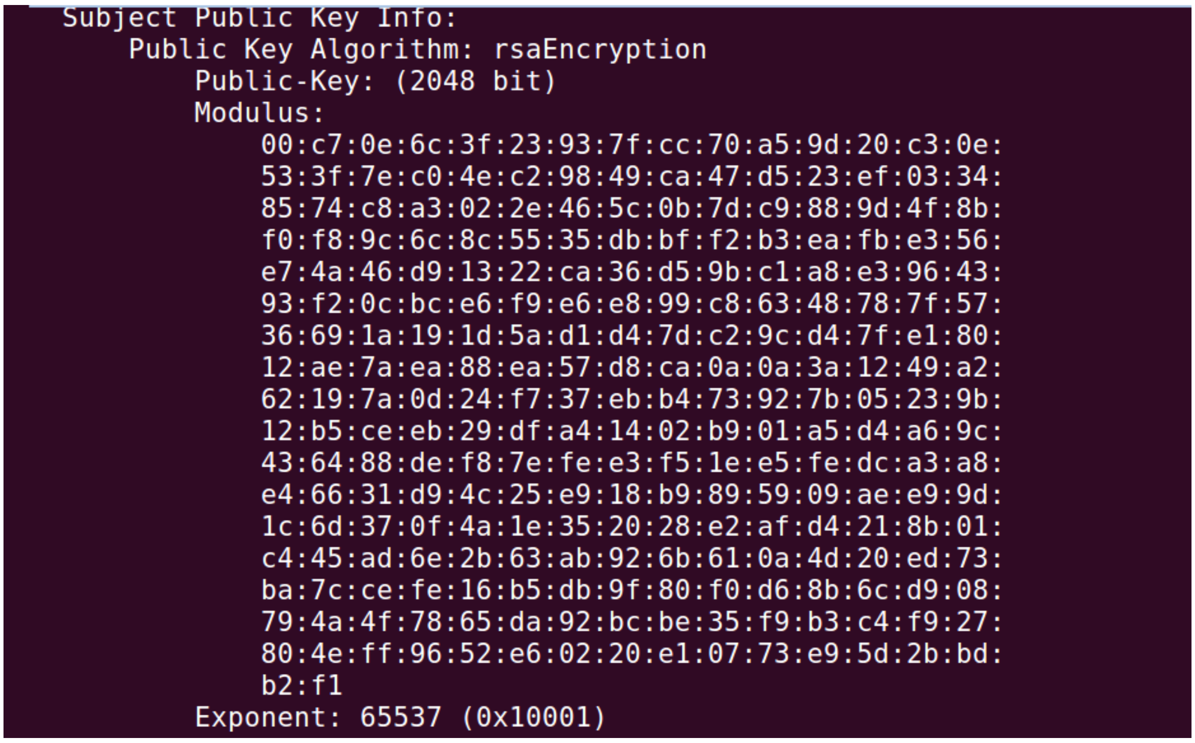
In this task I use command `openssl s_client -connect www.baidu.com:443 -showcerts` to get the certs from **www.baidu.com**



```
gethostbyname failure
connect:errno=11
[11/23/20]seed@VM:~/Desktop$ openssl s_client -connect www.sustech.edu.cn:443 -showcerts
CONNECTED(00000003)
depth=0 C = CN, ST = \E5\B9\BF\E4\B8\9C\E7\9C\81, L = \E6\B7\B1\E5\9C\B3\E5\B8\82, O = \E5\8D\97\E6\96\B9\E7\A7\91\E6\8A\80\E5\A4\A7\E5\AD\A6
, CN = *.sustech.edu.cn
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = CN, ST = \E5\B9\BF\E4\B8\9C\E7\9C\81, L = \E6\B7\B1\E5\9C\B3\E5\B8\82, O = \E5\8D\97\E6\96\B9\E7\A7\91\E6\8A\80\E5\A4\A7\E5\AD\A6
, CN = *.sustech.edu.cn
verify error:num=21:unable to verify the first certificate
verify return:1
---
Certificate chain
 0 s:/C=CN/ST=\xE5\xB9\xBF\xE4\xB8\x9C\xE7\x9C\x81/L=\xE6\xB7\xB1\xE5\x9C\xB3\xE5\xB8\x82/O=\xE5\x8D\x97\xE6\x96\xB9\xE7\xA7\x91\xE6\x8A\x80\
xE5\xA4\xA7\xE5\xAD\xA6/CN=*.sustech.edu.cn
   i:/C=US/O=DigiCert Inc/CN=DigiCert SHA2 Secure Server CA
-----BEGIN CERTIFICATE-----
MIIGNzCCBR+gAwIBAgIQA/HkLygcS2k4afE0dUlgYTANBgkqhkiG9w0BAQsFADBN
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMScwJQYDVQQDEx5E
aWdpQ2VydCBTSEEyIFNlY3VyZSBTZXJ2ZXIgQ0EwHhcNMjAxMTAyMDAwMDAwWhcN
MjExMTA5MjM1OTU5WjBtMQswCQYDVQQGEwJDTjESMBAGA1UECAwJ5bm/5Lic55yB
MRIwEAYDVQQHDAnmt7HlnLPluIIxGzAZBgNVBAoMEuWNl+aWueenkeaKgOWkp+Wt
pjEZMBcGA1UEAwwQKi5zdXN0ZWNoLmVkdS5jbjCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAKO5Jh/k2tEDqGbbs4eANmKHmN9VWX0dBFI9gF3Qd7E4/vsz
omAP59z51ceLRvpjz99W8QqUKm1gwfMAdI+UQdEjM7lGjcUo93GRkUectQpYh2pm
3EEeFAmTYUkR9VjtSrXnuOFiIs5lwL2DtEU0Ax/VG8xM3MAJ3yDl725a1lJfACZ8
qIMJbWBUxfUfu3p3QVhfRk6dqC197EDjkG1xhOahTGRZt3eWMzhVMz7GkcA9Vp5R
8zz7lixw+bvMEPiwT4fOJ+0Ldj/KixqvQN+UUtoY7iyTZC/KwtKp5p7rxgdZ5bsv
ElCEdBVzmL9xcrMiu2OWTarikZU8WrpSjXx/AlkCAwEAAaOCAvEwggLtMB8GA1Ud
IwQYMBaAFA+AYRyCMWHVLyjnjUY4tCzhxtniMB0GA1UdDgQWBBR/LnDT6l18xq1g
DoWV8swOqwnDmDArBgNVHREEJDAighAqLnN1c3RlY2guZWR1LmNugg5zdXN0ZWNo
LmVkdS5jbjAOBgNVHQ8BAf8EBAMCBaAwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsG
AQUFBwMCMGsGA1UdHwRkMGIwL6AtoCuGKWh0dHA6Ly9jcmwzLmRpZ2ljZXJ0LmNv
bS9zc2NhLXNoYTItZzcuY3JsMC+gLaArhilodHRwOi8vY3JsNC5kaWdpY2VydC5j
b20vc3NjYS1zaGEyLWc3LmNybDBMBgNVHSAERTBDMDcGCWCGSAGG/WwBATAqMCgG
CCsGAQUFBwIBFhxodHRwczovL3d3dy5kaWdpY2VydC5jb20vQ1BTMAgGBmeBDAEC
AjB8BggrBgEFBQcBAQRwMG4wJAYIKwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRpZ2lj
ZXJ0LmNvbTBGBggrBgEFBQcwAoY6aHR0cDovL2NhY2VydHMuZGlnaWNlcnQuY29t
L0RpZ2lDZXJ0U0hBMlNlY3VyZVNlcnZlckNBLmNydDAMBgNVHRMBAf8EAjAAMIIB
BgYKKwYBBAHWeQIEAgSB9wSB9ADyAHcA9lyUL9F3MCIUVBgIMJRWjuNNExkzv98M
LyALzE7xZOMAAAF1h8x7dwAABAMASDBGAiEAmkV37EQfDW/PnDuLn3J09b2Ymg4q
```

Step2

`openssl x509 -in c1.pem -text -noout` we got this



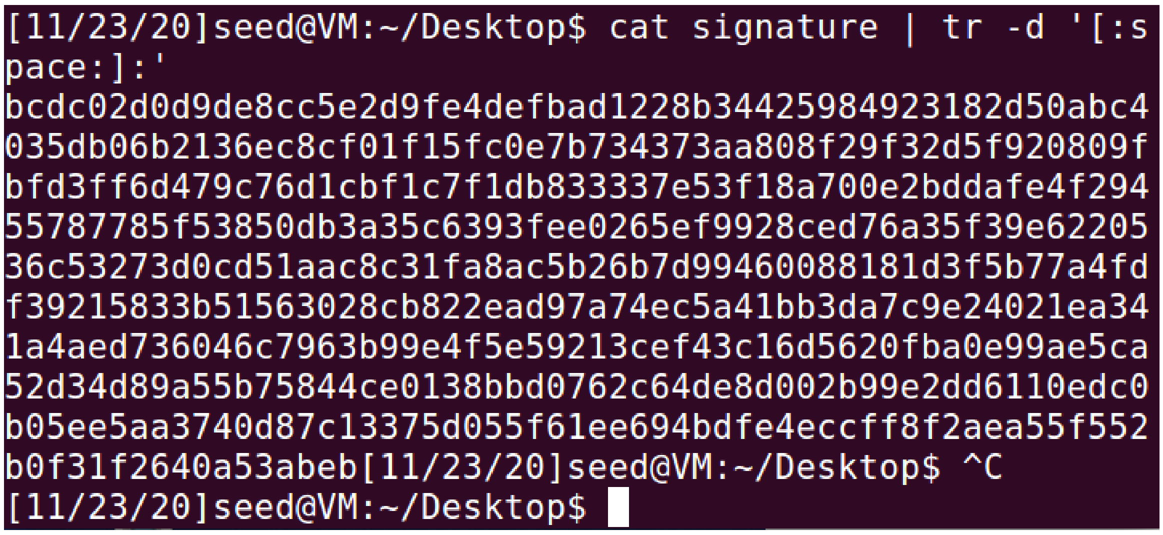
```
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:c7:0e:6c:3f:23:93:7f:cc:70:a5:9d:20:c3:0e:
                53:3f:7e:c0:4e:c2:98:49:ca:47:d5:23:ef:03:34:
                85:74:c8:a3:02:2e:46:5c:0b:7d:c9:88:9d:4f:8b:
                f0:f8:9c:6c:8c:55:35:db:bf:f2:b3:ea:fb:e3:56:
                e7:4a:46:d9:13:22:ca:36:d5:9b:c1:a8:e3:96:43:
                93:f2:0c:bc:e6:f9:e6:e8:99:c8:63:48:78:7f:57:
                36:69:1a:19:1d:5a:d1:d4:7d:c2:9c:d4:7f:e1:80:
                12:ae:7a:ea:88:ea:57:d8:ca:0a:0a:3a:12:49:a2:
                62:19:7a:0d:24:f7:37:eb:b4:73:92:7b:05:23:9b:
                12:b5:ce:eb:29:df:a4:14:02:b9:01:a5:d4:a6:9c:
                43:64:88:de:f8:7e:fe:e3:f5:1e:e5:fe:dc:a3:a8:
                e4:66:31:d9:4c:25:e9:18:b9:89:59:09:ae:e9:9d:
                1c:6d:37:0f:4a:1e:35:20:28:e2:af:d4:21:8b:01:
                c4:45:ad:6e:2b:63:ab:92:6b:61:0a:4d:20:ed:73:
                ba:7c:ce:fe:16:b5:db:9f:80:f0:d6:8b:6c:d9:08:
                79:4a:4f:78:65:da:92:bc:be:35:f9:b3:c4:f9:27:
                80:4e:ff:96:52:e6:02:20:e1:07:73:e9:5d:2b:bd:
                b2:f1
            Exponent: 65537 (0x10001)
```

Step3

`openssl x509 -in c0.pem -text -noout`

`cat signature | tr -d '[:space:]:'`

I got the signature like this

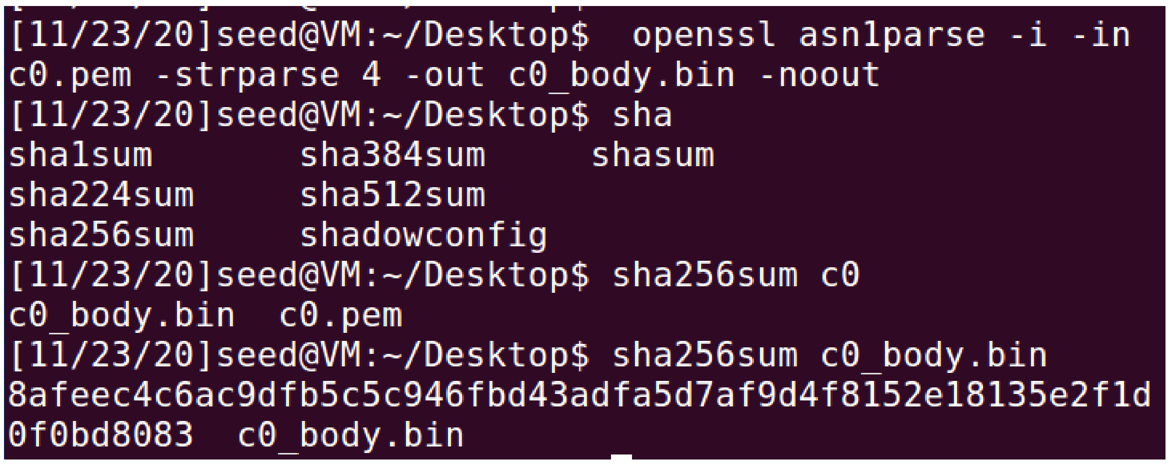

```
[11/23/20]seed@VM:~/Desktop$ cat signature | tr -d '[:s
pace:]:'
bcdc02d0d9de8cc5e2d9fe4defbad1228b34425984923182d50abc4
035db06b2136ec8cf01f15fc0e7b734373aa808f29f32d5f920809f
bfd3ff6d479c76d1cbf1c7f1db833337e53f18a700e2bddafe4f294
55787785f53850db3a35c6393fee0265ef9928ced76a35f39e62205
36c53273d0cd51aac8c31fa8ac5b26b7d99460088181d3f5b77a4fd
f39215833b51563028cb822ead97a74ec5a41bb3da7c9e24021ea34
1a4aed736046c7963b99e4f5e59213cef43c16d5620fba0e99ae5ca
52d34d89a55b75844ce0138bbd0762c64de8d002b99e2dd6110edc0
b05ee5aa3740d87c13375d055f61ee694bdfe4eccff8f2aea55f552
b0f31f2640a53abeb[11/23/20]seed@VM:~/Desktop$ ^C
[11/23/20]seed@VM:~/Desktop$
```

Step4

`openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout`

`sha256sum c0_body.bin`

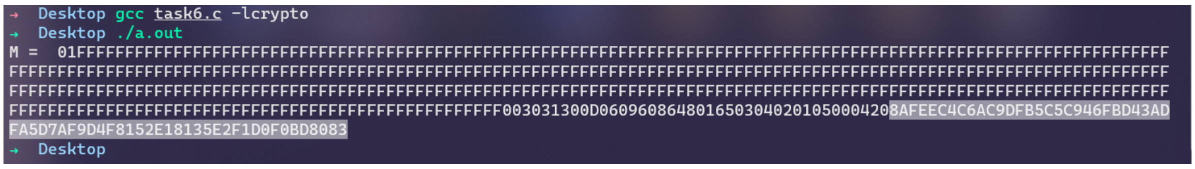
I got this hash:


```
[11/23/20]seed@VM:~/Desktop$  openssl asn1parse -i -in
c0.pem -strparse 4 -out c0_body.bin -noout
[11/23/20]seed@VM:~/Desktop$ sha
sha1sum        sha384sum      shasum
sha224sum      sha512sum
sha256sum      shadowconfig
[11/23/20]seed@VM:~/Desktop$ sha256sum c0
c0_body.bin  c0.pem
[11/23/20]seed@VM:~/Desktop$ sha256sum c0_body.bin
8afeec4c6ac9dfb5c5c946fbd43adfa5d7af9d4f8152e18135e2f1d
0f0bd8083  c0_body.bin
```

Then we write codes like this, the basic idea is to use the public key of in the `c1.pem`.

```c
BN_CTX *ctx = BN_CTX_new();
BIGNUM *M = BN_new();
BIGNUM *e = BN_new();
BIGNUM *n = BN_new();
BIGNUM *S = BN_new();

BN_hex2bn(&n,
"00c70e6c3f23937fcc70a59d20c30e533f7ec04ec29849ca47d523ef03348574c8a3022e465c0b7dc9889d4f8bf0f89c6c8c5535dbbff2b3eafbe356e74a46d91322ca36d59bc1a8e3964393f20cbce6f9e6e899c86348787f5736691a191d5ad1d47dc29cd47fe18012ae7aea88ea57d8ca0a0a3a1249a262197a0d24f737ebb473927b05239b12b5ceeb29dfa41402b901a5d4a69c436488def87efee3f51ee5fedca3a8e46631d94c25e918b9895909aee99d1c6d370f4a1e352028e2afd4218b01c445ad6e2b63ab926b610a4d20ed73ba7ccefe16b5db9f80f0d68b6cd908794a4f7865da92bcbe35f9b3c4f927804eff9652e60220e10773e95d2bbdb2f1");
BN_hex2bn(&e, "010001");
BN_hex2bn(&S,
"bcdc02d0d9de8cc5e2d9fe4defbad1228b34425984923182d50abc4035db06b2136ec8cf01f15fc0e7b734373aa808f29f32d5f920809fbfd3ff6d479c76d1cbf1c7f1db833337e53f18a700e2bddafe4f29455787785f53850db3a35c6393fee0265ef9928ced76a35f39e6220536c53273d0cd51aac8c31fa8ac5b26b7d99460088181d3f5b77a4fdf39215833b51563028cb822ead97a74ec5a41bb3da7c9e24021ea341a4aed736046c7963b99e4f5e59213cef43c16d5620fba0e99ae5ca52d34d89a55b75844ce0138bbd0762c64de8d002b99e2dd6110edc0b05ee5aa3740d87c13375d055f61ee694bdfe4eccff8f2aea55f552b0f31f2640a53abeb");
BN_mod_exp(M, S, e, n, ctx);
print_BN("M = ", M);
```

Then we got this:



```
→  Desktop gcc task6.c -lcrypto
→  Desktop ./a.out
M =  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D0609608648016503040201050004208AFEEC4C6AC9DFB5C5C946FBD43AD
FA5D7AF9D4F8152E18135E2F1D0F0BD8083
→  Desktop
```

The last 64 bytes is just the same as the hash value, which verifies that it is valid.

Summary

RSA public-key encryption and signature is widely used in our life, and this asymmetric encryption secures the information.

It usually uses 256 bits big number to generate public key and private key. From our program we can see that it is hardly to calculate and get the results by human, and it is hard to get the prime factors of the big number.

In the previous course Discrete Math we know the basic concept and operations of the RSA, and in this lab, I put the RSA into real life, and do the practice using baidu.com certs. It is helpful for me to understand more.