CS315 Lab10

Name: 王奕童

SID: 11910104

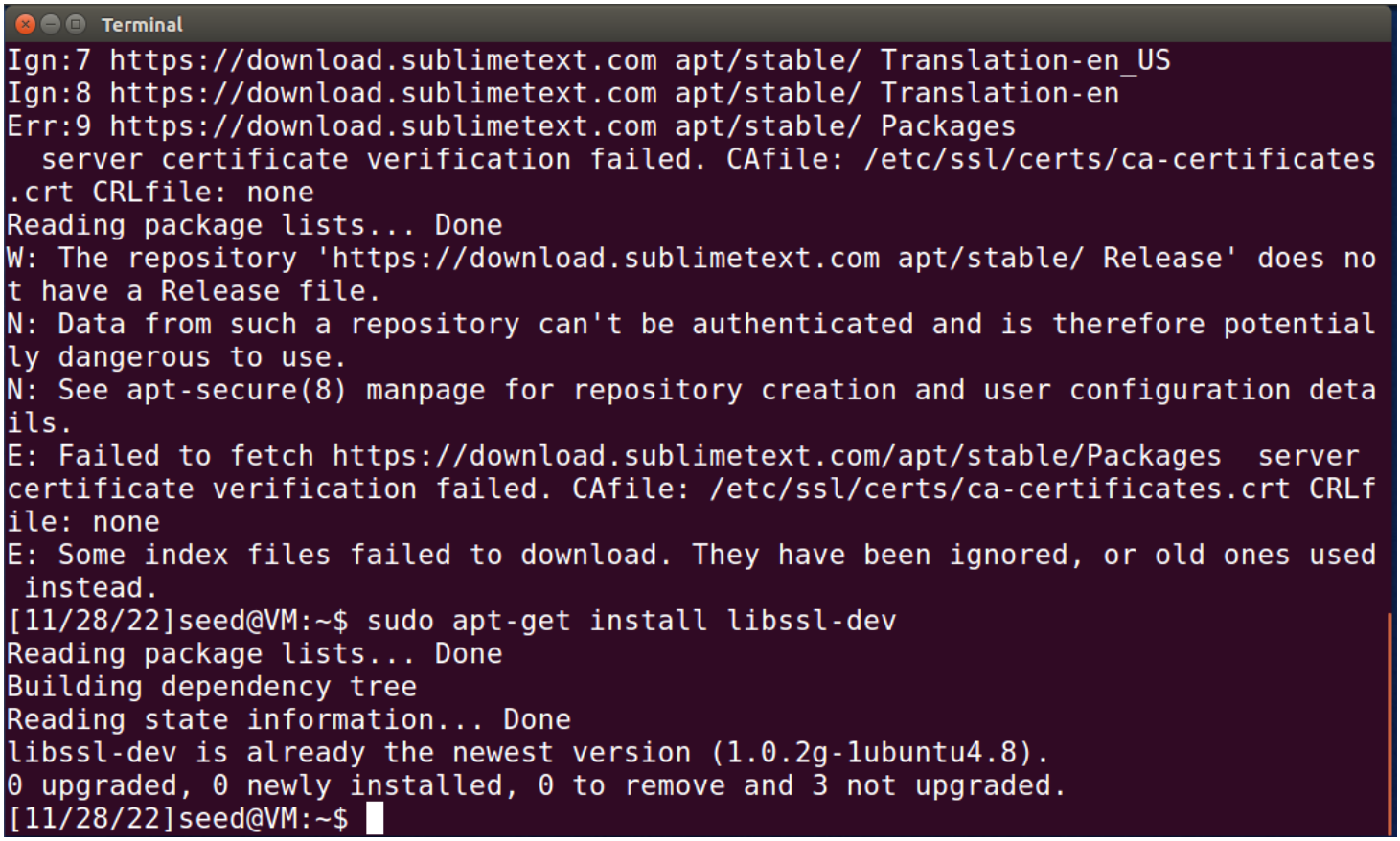
1 Overview

Lab Environment

虚拟机环境是Ubuntu 16.04。

安装openssl:

```
sudo apt-get update
sudo apt-get install libssl-dev
```

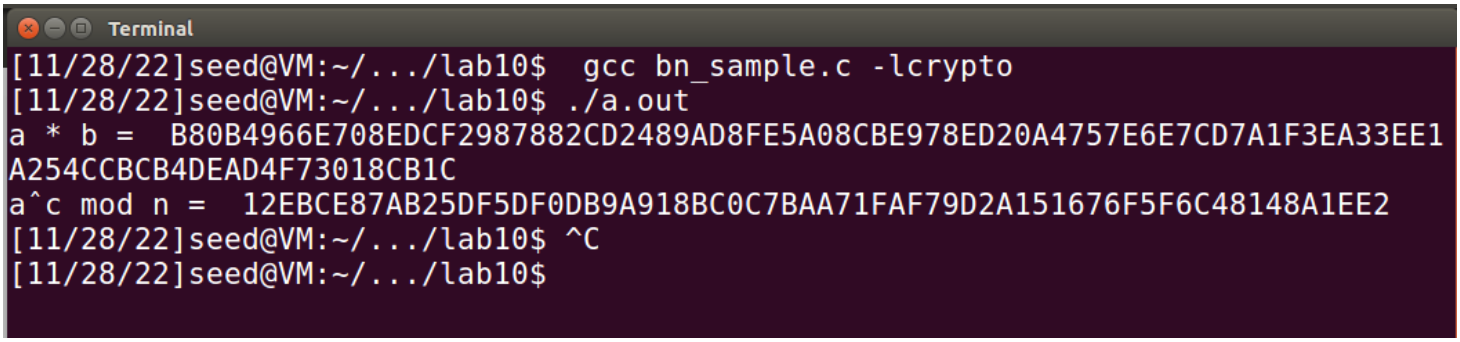


2 Background

2.2 A Complete Example

输入课件上的代码，并且编译执行，可以得到十六进制的结果：

```
gcc bn_sample.c -lcrypto
./a.out
```



3 Lab Tasks

3.1 Task 1: Deriving the Private Key

算法原理见大课第17页：

Choose two prime numbers p = 13 and q = 17

Find e:

- $n = pq = 221$
- $\varphi(n) = (p - 1)(q - 1) = 192$
- choose e = 7 (7 is relatively prime to $\phi(n)$)

Find d:

- $ed = 1 \bmod \varphi(n)$

简化成公式：

```
p_1 = p - 1
q_1 = q - 1
phi(n) = p_1 * q_1
d = e^-1 mod phi(n) // d是e模phi(n)的逆元
```

代码实现：

```
BN_CTX* ctx = BN_CTX_new();
BIGNUM* p = BN_new();
BIGNUM* p_1 = BN_new();
BIGNUM* q = BN_new();
BIGNUM* q_1 = BN_new();
BIGNUM* e = BN_new();
BIGNUM* n = BN_new();
BIGNUM* phi_n = BN_new();

BIGNUM* d = BN_new();

BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&p_1, "F7E75FDC469067FFDC4E847C51F452DE");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&q_1, "E85CED54AF57E53E092113E62F436F4E");
BN_hex2bn(&e, "0D88C3");

BN_rand(n, NBITS, 0, 0);
BN_rand(phi_n, NBITS, 0, 0);
BN_mul(phi_n, p_1,q_1,ctx);
BN_mod_inverse(d, e, phi_n, ctx);

printBN("private key: ", d);
```

获得私钥内容是：

```
3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

```
[11/28/22]seed@VM:~/.../lab10$ gcc task1.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
test:  0
[11/28/22]seed@VM:~/.../lab10$ gcc task1.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
test:  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
[11/28/22]seed@VM:~/.../lab10$ gcc task1.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
private key:  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[11/28/22]seed@VM:~/.../lab10$
```

3.2 Task 2: Encrypting a Message

算法原理见大课第18页：

RSA Exercise: Small Numbers (Contd.)

Encrypt M = 36

$$
\begin{aligned}
M^e \mod n &= 36^7 \mod 221 \\
&= (36^2 \mod 221)^3 * 36 \mod 221 \\
&= 191^3 * 36 \mod 221 \\
&= 179 \mod 221.
\end{aligned}
$$

Cipher text (C) = 179

代码实现：

```
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();

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("Task 2: Encrypting a Message--", C);
```

获得加密后字符串为：

```
6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

```
[11/28/22]seed@VM:~/.../lab10$ gcc task2.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
Task 2: Encrypting a Message-- 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CA
CDC5DE5CFC5FADC
[11/28/22]seed@VM:~/.../lab10$
```

Task 3: Decrypting a Message

算法原理在大课课件第16页：

- Decryption
  - $M = C^d \mod n$

代码实现：

```
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();

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


BN_mod_exp(M,C,d,n,ctx);

printBN("Task 3: Decrypting a Message--", M);
```

沿用上题数据：

```
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e = 010001 (this hex value equals to decimal 65537)
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
```

先解密为十六进制，结果为：

```
50617373776F72642069732064656573
```

```
[11/28/22]seed@VM:~/.../lab10$ gcc task3.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
Task 3: Decrypting a Message-- 50617373776F72642069732064656573
```

再通过python进行16进制转换，结果为：

```
Password is dees
```

```
[11/28/22]seed@VM:~/.../lab10$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "50617373776F726420697320646565573".decode("hex")
'Password is dees'
>>>
```

3.4 Task 4: Signing a Message

首先将两个字符串都转为16进制的编码。二者非常相似，只有1位有所不同。

```
I owe you $2000.
49206f776520796f752024323030302e

I owe you $3000.
49206f776520796f752024333030302e
```

```
>>> "I owe you $2000.".encode("hex")
'49206f776520796f752024323030302e'
>>> "I owe you $3000.".encode("hex")
'49206f776520796f752024333030302e'
>>>
```

编写获取签名的代码：

```
BN_CTX* ctx = BN_CTX_new();
BIGNUM* n = BN_new();
BIGNUM* e = BN_new();
BIGNUM* M2000 = BN_new();
BIGNUM* M3000 = BN_new();
BIGNUM* d = BN_new();
BIGNUM* C2000 = BN_new();
BIGNUM* C3000 = BN_new();

BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&M2000, "49206f776520796f752024323030302e");
BN_hex2bn(&M3000, "49206f776520796f752024333030302e");



BN_mod_exp(C2000,M2000,d,n,ctx);
BN_mod_exp(C3000,M3000,d,n,ctx);

printBN("Task 4: Signing a Message 2000->", C2000);
printBN("Task 4: Signing a Message 3000->", C3000);
```

此时查看获取的签名，可以发现它们有比较大的差异：

```
I owe you $2000.
55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB

I owe you $3000.
BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
```

```
[11/28/22]seed@VM:~/.../lab10$ gcc task4.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
Task 4: Signing a Message 2000-> 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D7
85ED6E73CCB35E4CB
Task 4: Signing a Message 3000-> BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3
EBAC0135D99305822
```

3.5 Task 5: Verifying a Signature

条件数据：

```
M = Launch a missile.
S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
e = 010001 (this hex value equals to decimal 65537)
n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
```

首先对M进行16进制转换，结果为：

```
4c61756e63682061206d697373696c652e
```

```
[11/28/22]seed@VM:~/.../lab10$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "Launch a missile.".encode("hex")
'4c61756e63682061206d697373696c652e'
>>>
```

其次是签名的验证代码实现：

```
BN_CTX* ctx = BN_CTX_new();

BIGNUM* n = BN_new();
BIGNUM* e = BN_new();
BIGNUM* d = BN_new();
BIGNUM* S = BN_new();
BIGNUM* M = BN_new();
BIGNUM* C = BN_new();

BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&M, "49206f776520796f752024323030302e");
BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");


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

printBN("Task 5: Verifying a Signature, C is", C);
```

运行后输出C，结果为：

```
4C61756E63682061206D697373696C652E
```

```
[11/28/22]seed@VM:~/.../lab10$ gcc task5.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
Task 5: Verifying a Signature, C is 4C61756E63682061206D697373696C652E
```

将该字符串放入python进行16进制解码，可以得到Alice的消息，因此可以验证该消息是来自于Alice：

```
[11/28/22]seed@VM:~/.../lab10$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "4C61756E63682061206D697373696C652E".decode("hex")
'Launch a missile.'
>>>
```

再考虑签名被污染的情况，修改S

为 `643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F` 。

再次输出C，结果是：

```
91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
```

```
[11/28/22]seed@VM:~/.../lab10$ gcc task5.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
Task 5: Verifying a Signature, C is 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A
4EB6B7BE0203B41AC294
[11/28/22]seed@VM:~/.../lab10$
```

再次进行python十六进制解码，可以看到字符已不可读：

```
[11/28/22]seed@VM:~/.../lab10$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294".decode("h
ex")
"\x91G\x19'\xc8\r\xf1\xe4,\x15O\xb4c\x8c\xe8\xbcrm=f\xc8:N\xb6\xb7\xbe\x02\x03\x
b4\x1a\xc2\x94"
>>>
```

Task 6: Manually Verifying an X.509 Certificate

Step 1: Download a certificate from a real web server

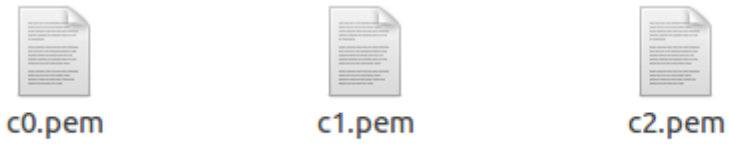
我下载 `www.bilibili.com` 的证书：

```
openssl s_client -connect www.bilibili.com:443 -showcerts
```

一共有3个证书：

```
---
Certificate chain
 0 s:/C=CN/ST=\xE4\xB8\x8A\xE6\xB5\xB7/L=\xE4\xB8\x8A\xE6\xB5\xB7/O=\xE4\xB8\x8A
\xE6\xB5\xB7\xE5\xB9\xBB\xE7\x94\xB5\xE4\xBF\xA1\xE6\x81\xAF\xE7\xA7\x91\xE6\x8A
\x80\xE6\x9C\x89\xE9\x99\x90\xE5\x85\xAC\xE5\x8F\xB8/CN=*.bilibili.com
   i:/C=BE/O=GlobalSign nv-sa/CN=GlobalSign RSA OV SSL CA 2018
-----BEGIN CERTIFICATE-----
MIIGYTCCBUmgAwIBAgIMWEu0W0OlVSp/zY8wMA0GCSqGSIb3DQEBCwUAMFAxCzAJ
BgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMSYwJAYDVQQDEx1H
bG9iYWxTaWduIFJTQSBPViBTU0wgQ0EgMjAxODAeFw0yMjA5MjkwOTU2MDJaFw0y
V6uwM6E=
-----END CERTIFICATE-----
 1 s:/C=BE/O=GlobalSign nv-sa/CN=GlobalSign RSA OV SSL CA 2018
   i:/OU=GlobalSign Root CA - R3/O=GlobalSign/CN=GlobalSign
-----BEGIN CERTIFICATE-----
MIIETjCCAzagAwIBAgINAe5fIh38YjvUMzqFVzANBgkqhkiG9w0BAQsFADBMMSAw
HgYDVQQLExdHbG9iYWxTaWduIFJvb3QgQ0EgLSBSMzETMBEGA1UEChMKR2xvYmFs
U2lnbjETMBEGA1UEAxMKR2xvYmFsU2lnbjAeFw0xODExMjEwMDAwMDBaFw0yODEx
SPT=
-----END CERTIFICATE-----
 2 s:/OU=GlobalSign Root CA - R3/O=GlobalSign/CN=GlobalSign
   i:/C=BE/O=GlobalSign nv-sa/OU=Root CA/CN=GlobalSign Root CA
-----BEGIN CERTIFICATE-----
MIIETjCCAzagAwIBAgINAe5fFp3/lzUrZGXWajANBgkqhkiG9w0BAQsFADBXMQsw
CQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UECxMH
Um9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTE4MDkxOTAw
MDAwMFoXDTI4MDEyODEyMDAwMFowTDEgMB4GA1UECxMXR2xvYmFsU2lnbiBSb290
IENBIC0gUjMxEzARBgNVBAoTCkdsb2JhbFNpZ24xEzARBgNVBAMTCkdsb2JhbFNp
```

保存至文件：



Step 2: Extract the public key (e, n) from the issuer's certificate.

命令：

```
openssl x509 -in c1.pem -text -noout
```

```
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:a7:5a:c9:d5:0c:18:21:00:23:d5:97:0f:eb:ae:
            dd:5c:68:6b:6b:8f:50:60:13:7a:81:cb:97:ee:8e:
            8a:61:94:4b:26:79:f6:04:a7:2a:fb:a4:da:56:bb:
            ee:a0:a4:f0:7b:8a:7f:55:1f:47:93:61:0d:6e:71:
            51:3a:25:24:08:2f:8c:e1:f7:89:d6:92:cf:af:b3:
            a7:3f:30:ed:b5:df:21:ae:fe:f5:44:17:fd:d8:63:
            d9:2f:d3:81:5a:6b:5f:d3:47:b0:ac:f2:ab:3b:24:
            79:4f:1f:c7:2e:ea:b9:15:3a:7c:18:4c:69:b3:b5:
            20:59:09:5e:29:c3:63:e6:2e:46:5b:aa:94:90:49:
            0e:b9:f0:f5:4a:a1:09:2f:7c:34:4d:d0:bc:00:c5:
            06:55:79:06:ce:a2:d0:10:f1:48:43:e8:b9:5a:b5:
            95:55:bd:31:d2:1b:3d:86:be:a1:ec:0d:12:db:2c:
            99:24:ad:47:c2:6f:03:e6:7a:70:b5:70:cc:cd:27:
            2c:a5:8c:8e:c2:18:3c:92:c9:2e:73:6f:06:10:56:
            93:40:aa:a3:c5:52:fb:e5:c5:05:d6:69:68:5c:06:
            b9:ee:51:89:e1:8a:0e:41:4d:9b:92:90:0a:89:e9:
            16:6b:ef:ef:75:be:7a:46:b8:e3:47:8a:1d:1c:2e:
            a7:4f
        Exponent: 65537 (0x10001)
```

Step 3: Extract the signature from the server's certificate.

命令：

```
openssl x509 -in c0.pem -text -noout
cat signature | tr -d '[:space:]:'
```

提取出的签名为：

```
[11/28/22]seed@VM:~/.../lab10$ touch signature
[11/28/22]seed@VM:~/.../lab10$ cat signature | tr -d '[:space:]:'
03eb45dde8eaf5b18654d3f5709aa85781eb25c005d700ef193a6d866361635eff8129b9a2378fca
0c756ac9150c59e87725233d68b15dee0c75b35147e3ecc57d88a7ecff425ee006bcb2ed40243ff5
aae7237e959bfe437529fef89a8a409adbf93cfb2a89532af95a66c96919cd937394117bc53fea08
a3b6924782b30daa157e6f462a16b3a1731163d1bac1ef8dc67c0d6362fd9341feaba9ffd1dc6495
390f53cbef2168f4b4433eb8ae474a029c378f1293fef2e1041e062c15b38f062218aaf4073c252c
802ecc6780e81cb0b9196dcdd59b2e7cdd69c3b01f22d96cb720533ea916bec47d6dd13235fb046d
1bf84dcf76a7bb8099f85d57abb033a1[11/28/22]seed@VM:~/.../lab10$ ^C
[11/28/22]seed@VM:~/.../lab10$ ^C
[11/28/22]seed@VM:~/.../lab10$
```

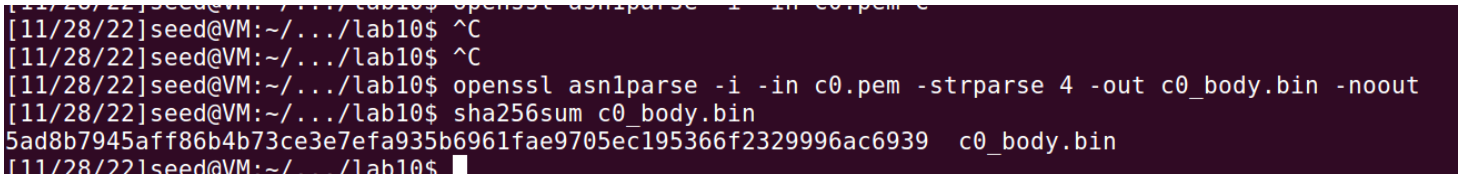
Step 4: Extract the body of the server's certificate.

命令：

```
openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
sha256sum c0_body.bin
```

计算出的hash为：

```
5ad8b7945aff86b4b73ce3e7efa935b6961fae9705ec195366f2329996ac6939
```



Step 5: Verify the signature.

使用收集到的数据进行签名验证：

```
n = 00a75a...2ea74f(Step 2中的Modulus)
e = 010001
S = 03eb45...b033a1
```

编写验证的代码：

```
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, "00a75ac9d50c18210023d5970febaedd5c686b6b8f5060137a81cb97ee8e8a61944b2679f604a72af
BN_hex2bn(&e, "010001");
BN_hex2bn(&S, "03eb45dde8eaf5b18654d3f5709aa85781eb25c005d700ef193a6d866361635eff8129b9a2378fca0

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

printBN("3.6 Task 6: Manually Verifying an X.509 Certificate, M --> ", M);
```

```
[11/28/22]seed@VM:~/.../lab10$ gcc task6.c -lcrypto
[11/28/22]seed@VM:~/.../lab10$ ./a.out
3.6 Task 6: Manually Verifying an X.509 Certificate, M -->  01FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00303130
0D0609608648016503040201050004205AD8B7945AFF86B4B73CE3E7EFA935B6961FAE9705EC195
366F2329996AC6939
```

最后64位和Step4中得到的签名相同，因此我们认定该签名是合法的。

Acknowledgement

这次报告是和term project的几位成员（张睿豪，李家奥，谢岳臻，刘晟淇）一起完成了RSA的数学公式推导。上次学RSA还是在两年前王琦老师的CS201离散数学上，很多知识早就还给王琦老师了，因此重新学习RSA的相关原理花费了一些时间。