Report of RSA

## 3.1 Task 1: Deriving the Private Key

BN\_CTX \*ctx = BN\_CTX\_new();

BIGNUM \*p = BN\_new();

BIGNUM \*q = BN\_new();

BIGNUM \*n = BN\_new();

BIGNUM \*d = BN\_new();

BIGNUM \*e = BN\_new();

BIGNUM \*one = BN\_new();

BIGNUM \*zero = BN\_new();

BIGNUM \*reg1 = BN\_new();

BIGNUM \*reg2 = BN\_new();

BN\_one(one);

BIGNUM \*reg3 = BN\_new();

BN\_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");

BN\_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

BN\_hex2bn(&e, "0D88C3");

BN\_sub(reg1, p, one);

BN\_sub(reg2, q, one);

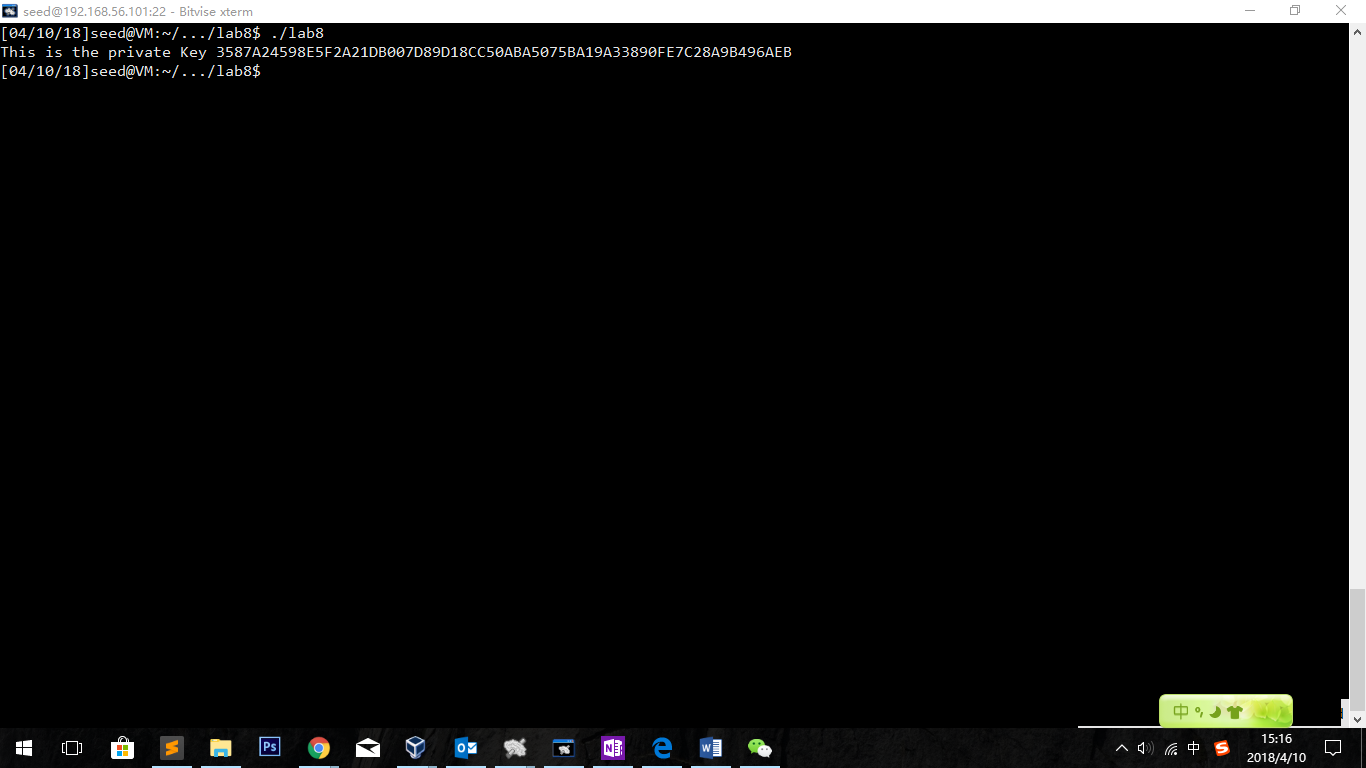
BN\_mul(reg3, reg1, reg2, ctx);

BN\_mul(n, p, q, ctx);

BN\_mod\_inverse(d, e, reg3, ctx);

printBN("This is the private Key", d);

Get the key: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB



## 3.2 Task2: Encrypting a Message

### Codes

#include <stdio.h>

#include <openssl/bn.h>

#define NBITS 256

void printBN(char \*msg, BIGNUM \* a)

{

/\* Use BN\_bn2hex(a) for hex string

\* Use BN\_bn2dec(a) for decimal string \*/

char \* number\_str = BN\_bn2hex(a);

printf("%s %s\n", msg, number\_str);

OPENSSL\_free(number\_str);

}

BIGNUM\* encrypt(BIGNUM \*ciphertext,BIGNUM \*message,BIGNUM \*publickey,BIGNUM \*theN){

BN\_CTX \*ctx = BN\_CTX\_new();

BN\_mod\_exp(ciphertext, message, publickey, theN, ctx);

return ciphertext;

}

BIGNUM\* decrypt(BIGNUM \*message,BIGNUM \*ciphertext,BIGNUM \*privatekey,BIGNUM \*theN){

BN\_CTX \*ctx = BN\_CTX\_new();

BN\_mod\_exp(message, ciphertext, privatekey, theN, ctx);

return message;

}

int main ()

{

BIGNUM \*m = BN\_new();

BN\_hex2bn(&m,"4120746F702073656372657421");

BIGNUM \*d = BN\_new();

BN\_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

BIGNUM \*e = BN\_new();

BN\_hex2bn(&e,"010001");

BIGNUM \*n = BN\_new();

BN\_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

BIGNUM \*ciphertext=BN\_new();

encrypt(ciphertext,m,e,n);

printBN("This is the ciphertext: ",ciphertext);

BIGNUM \*plaintext=BN\_new();

decrypt(plaintext,ciphertext,d,n);

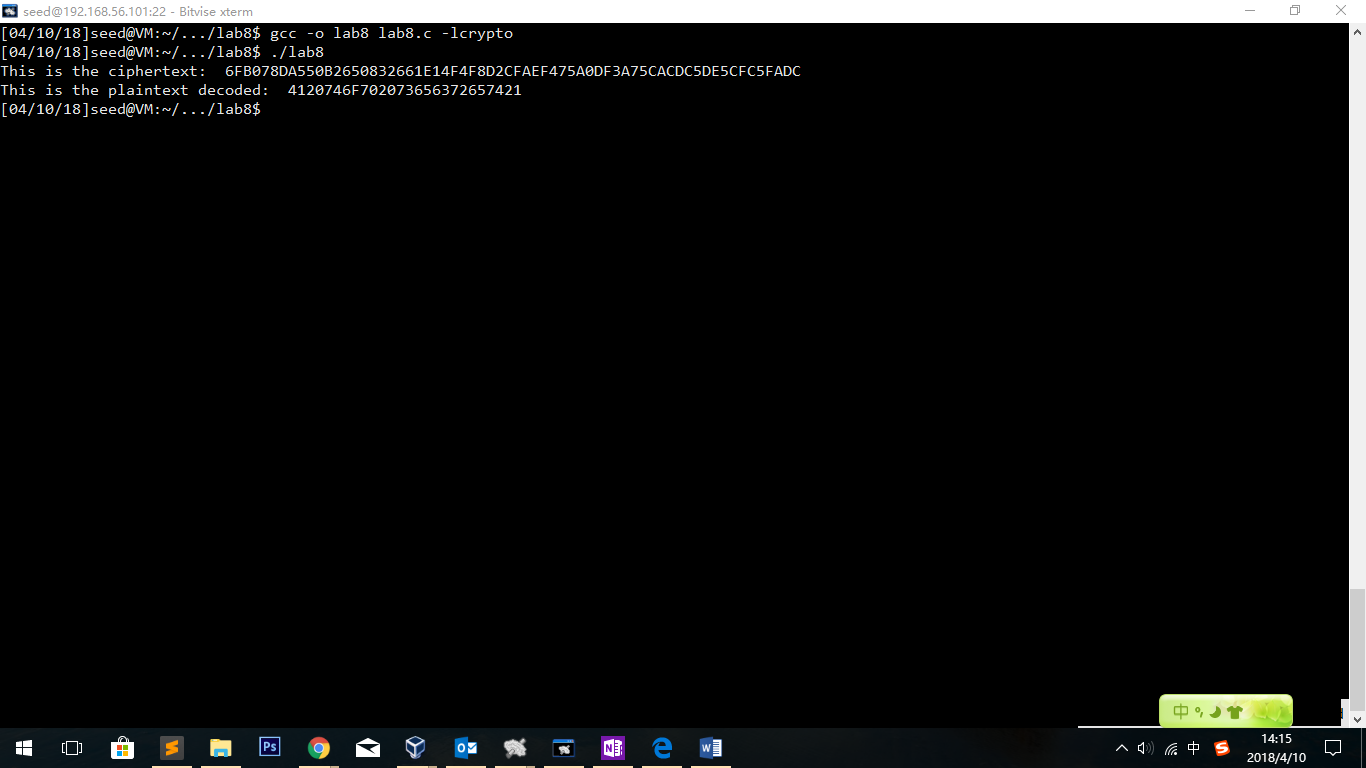
printBN("This is the plaintext decoded: ",plaintext);

return 0;

}

Result:

Cipher Text is 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC



Observation:

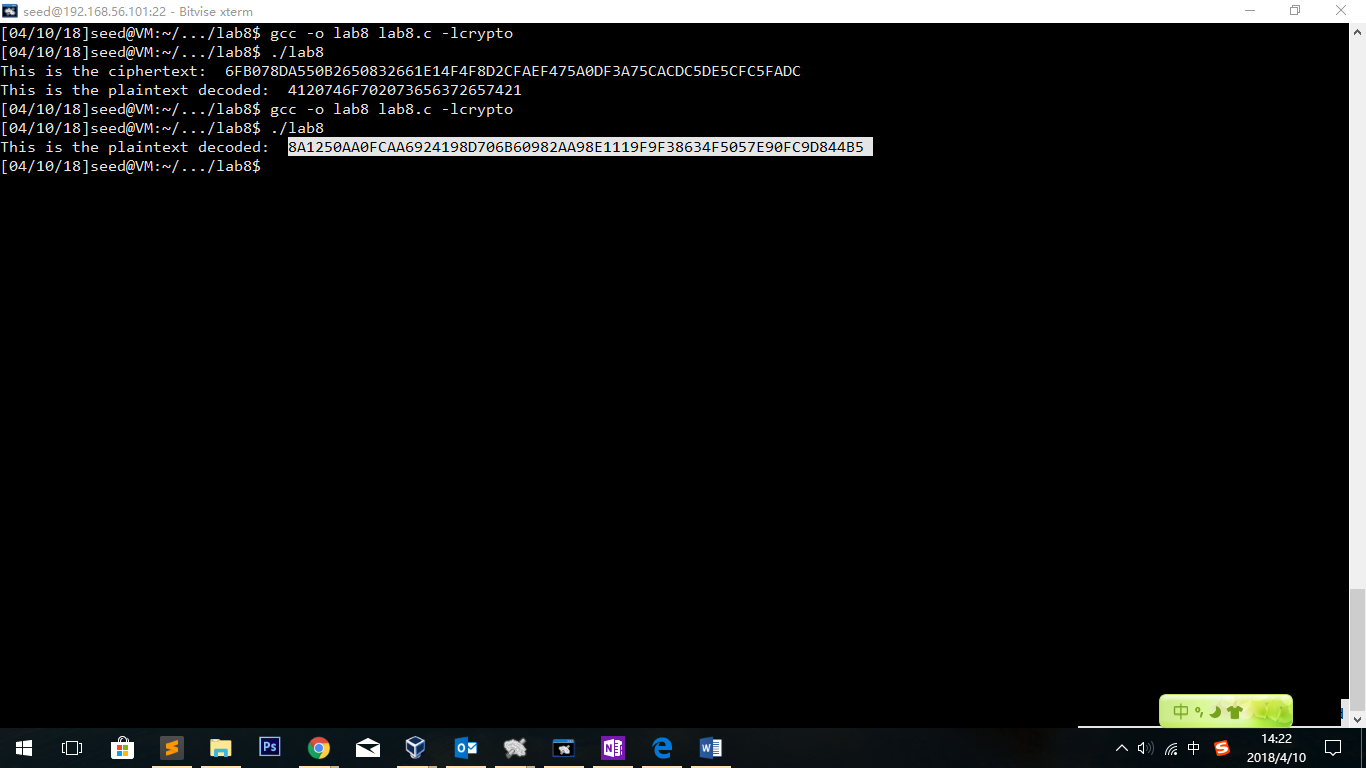
Not only we have got the ciphertext of the message, but we also encrypt it using the private key provided. After we get the plaintext decoded, we find that the text is the same as the original one. As a result, we can see that the ciphertext is correct.

## 3.3 Task3: Decrypting a Message

Codes are totally same as the previous one.

The only change is that we need to change the ciphertext from the one we generated to the given one 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493.

The plaintext should be:

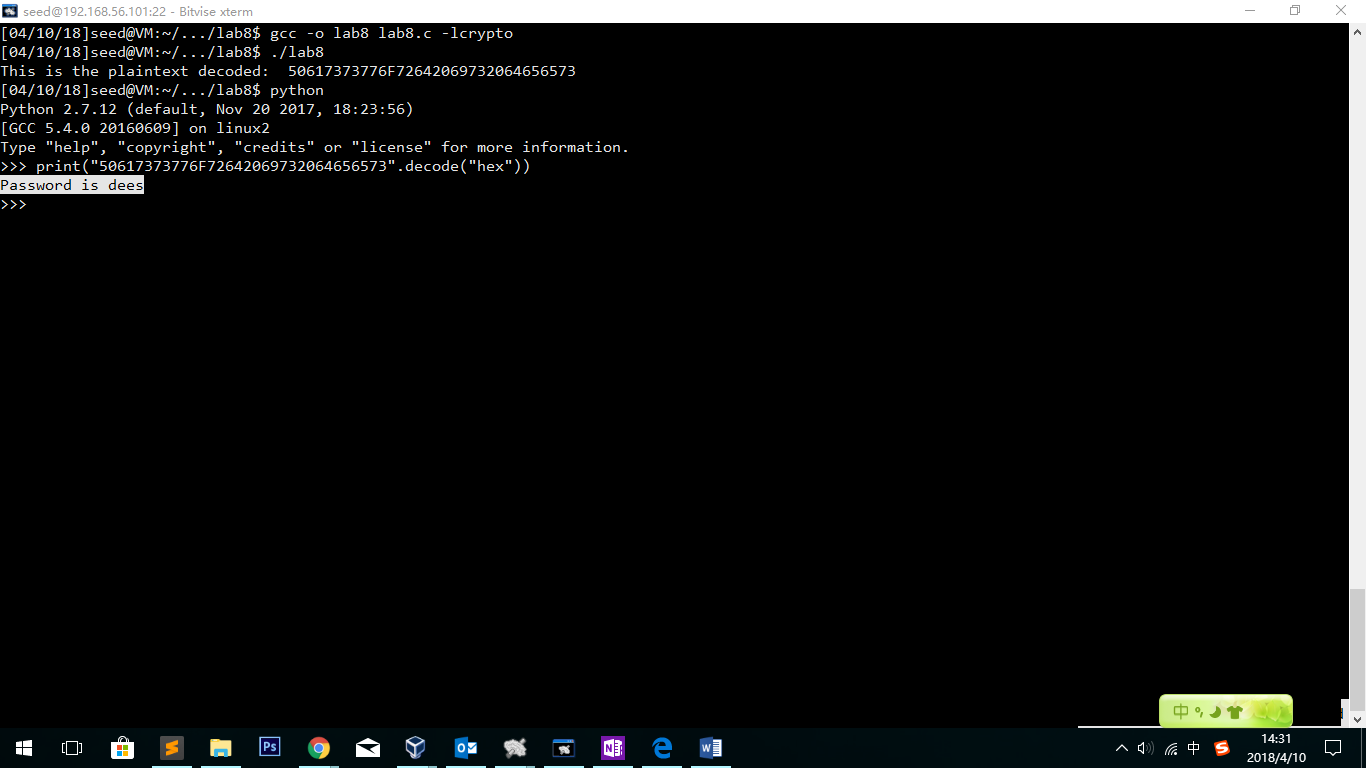
50617373776F72642069732064656573

And we need to decode it using python.

print("50617373776F72642069732064656573".decode("hex"))

Get the plain text in ASCII String:

Password is dees



## 3.4 Task4: Signing a Message

We can get the hex value of the two strings:

print("I owe you $3000.".encode("hex"))

I owe you $2000. : 49206f776520796f752024323030302e

I owe you $3000. : 49206f776520796f752024333030302e

Using the two separate plaintext to encode.

This time we need to change to use d to encrypt and e to verify

The only change of the code should be:

encrypt(ciphertext,m,e,n);🡺 encrypt(ciphertext,m,d,n);

decrypt(plaintext,c,d,n);🡺 decrypt(plaintext,c,e,n);

BIGNUM \*m1 = BN\_new();

BN\_hex2bn(&m1,"49206f776520796f752024323030302e");

BIGNUM \*m2 = BN\_new();

BN\_hex2bn(&m2,"49206f776520796f752024333030302e");

BIGNUM \*d = BN\_new();

BN\_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

BIGNUM \*e = BN\_new();

BN\_hex2bn(&e,"010001");

BIGNUM \*n = BN\_new();

BN\_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");

BIGNUM \*ciphertext1=BN\_new();

encrypt(ciphertext1,m1,d,n);

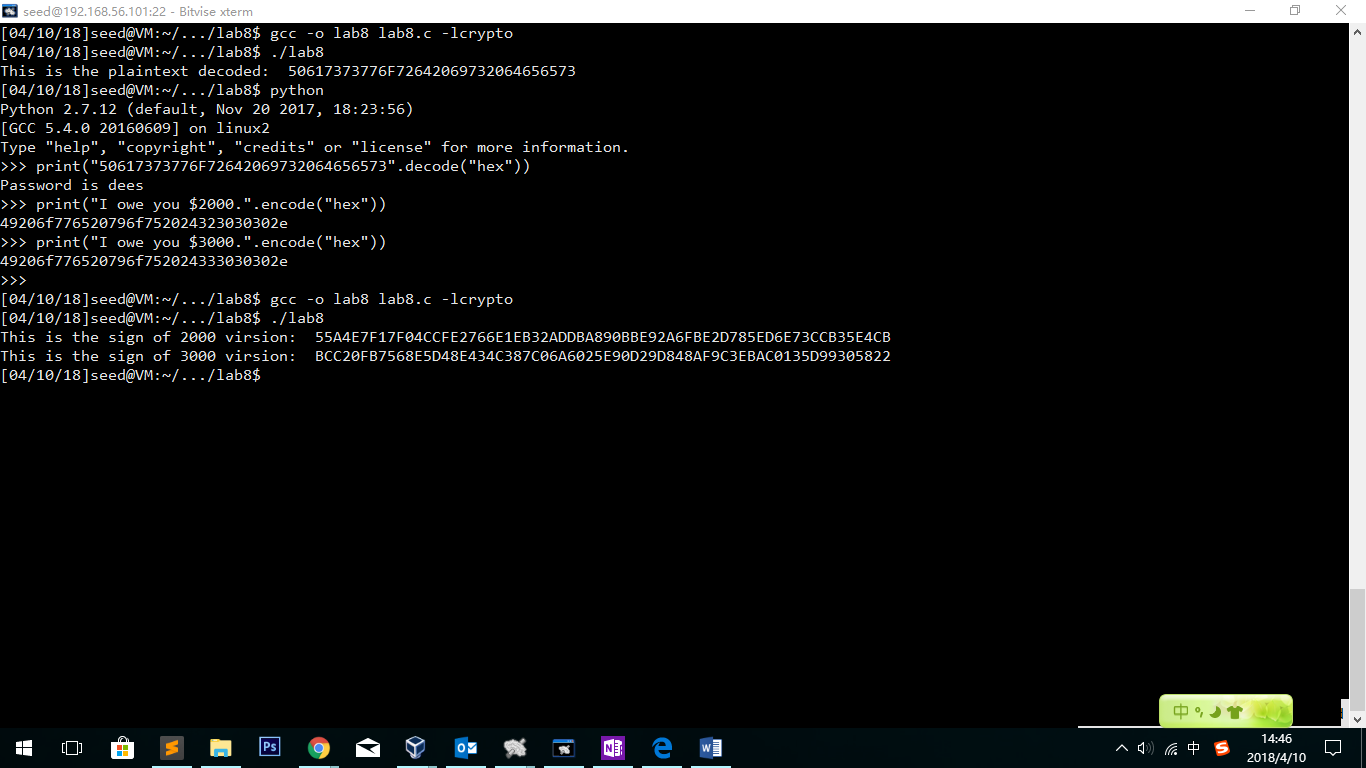
printBN("This is the sign of 2000 virsion: ",ciphertext1);

BIGNUM \*ciphertext2=BN\_new();

encrypt(ciphertext2,m2,d,n);

printBN("This is the sign of 3000 virsion: ",ciphertext2);

Result:



This is the sign of 2000 virsion: 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB

This is the sign of 3000 virsion: BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822

**Observation:**

Although in the hex text they only have one bit different, their signs are completely different.

## 3.5 Task5: Verifying a Signature

print("Launch a missile.".encode("hex"))

Launch a missle. 🡺

Codes:

BIGNUM \*m1 = BN\_new();

BN\_hex2bn(&m1,"4c61756e63682061206d697373696c652e");

BIGNUM \*s = BN\_new();

BN\_hex2bn(&s,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

BIGNUM \*e = BN\_new();

BN\_hex2bn(&e,"010001");

BIGNUM \*n = BN\_new();

BN\_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");

BIGNUM \*ciphertext=BN\_new();

encrypt(ciphertext,s,e,n);

printBN("This is the decoded message:", ciphertext);

if(BN\_cmp(ciphertext,m1)==0){

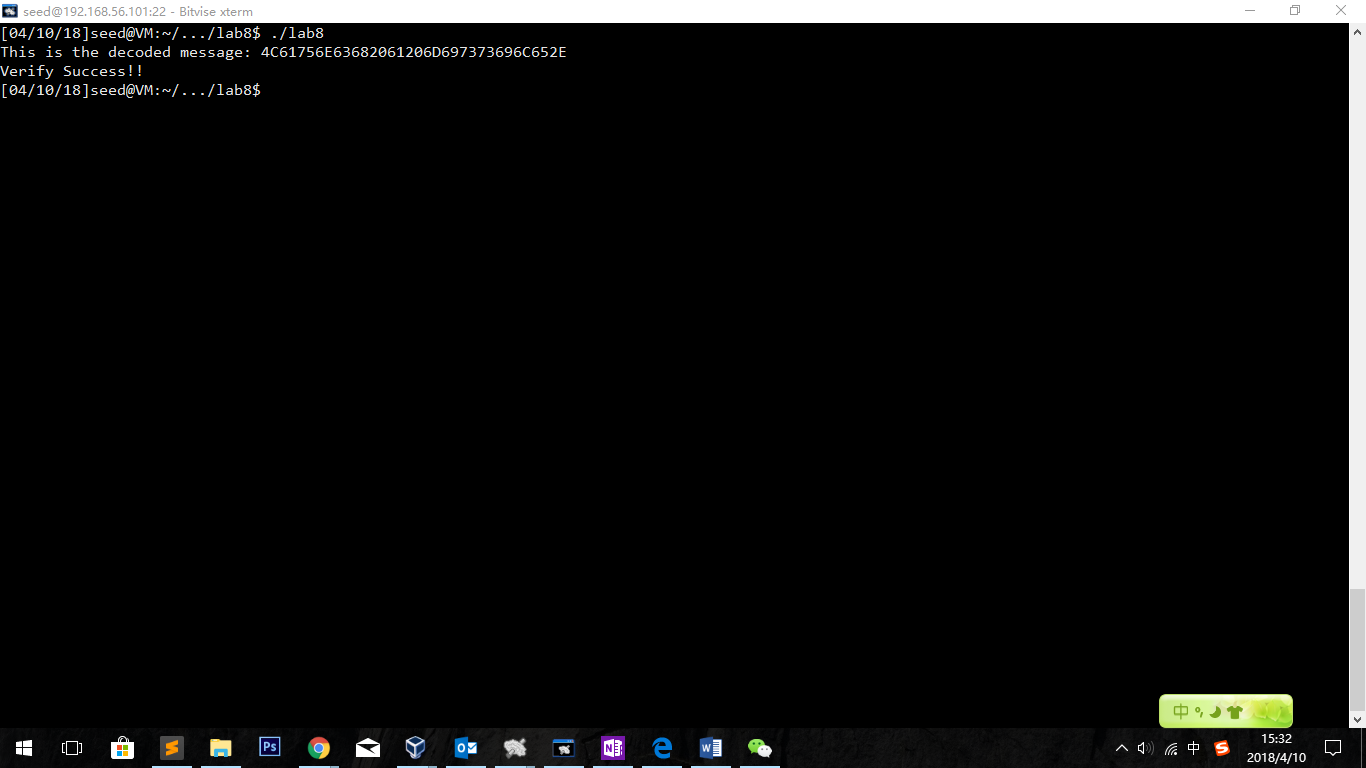
printf("Verify Success!!\n");

}

else{

printf("Verify Denied!!\n");

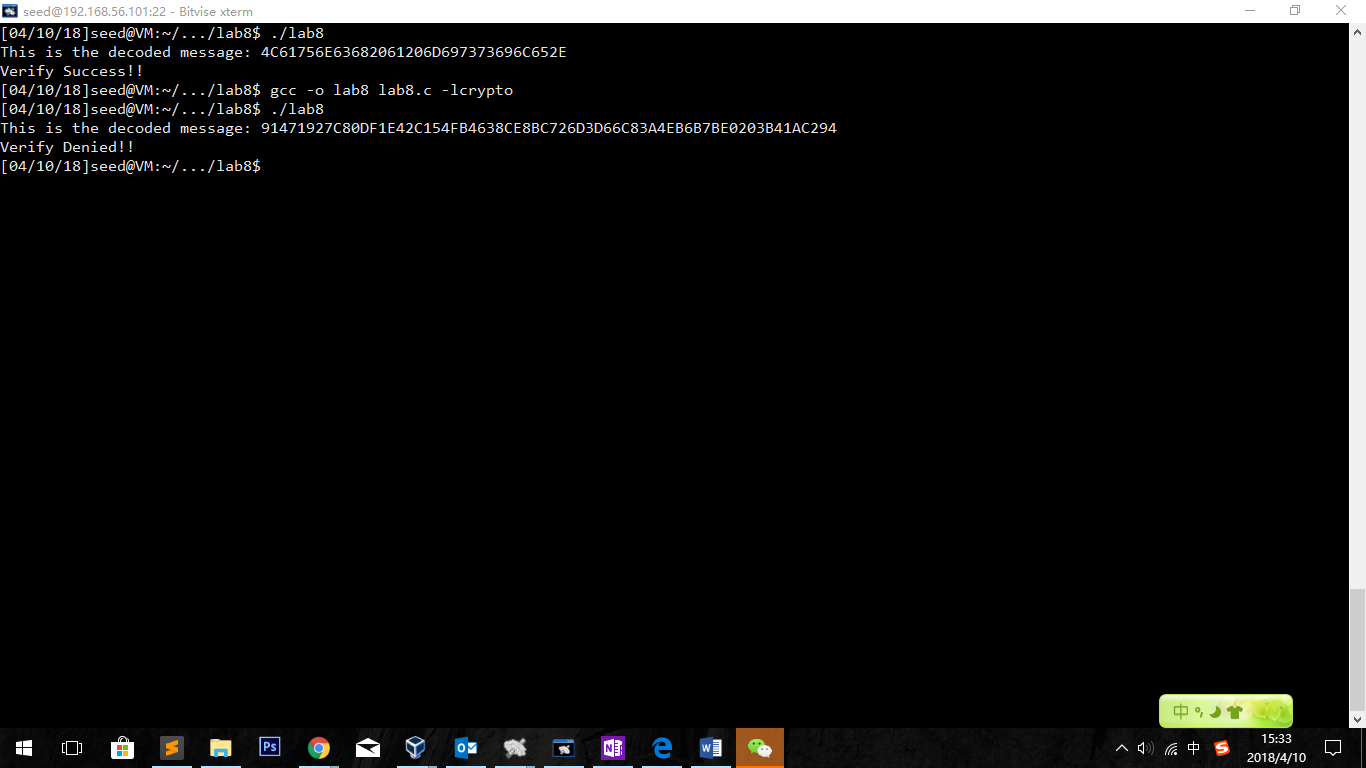
}



Observation:

The verify is successful!!

Change 2 to 3:



Observation:

The plaintexts are completely different and the final result is the verify is denied.

## 3.6 Task6: Manually Verifying an X.509 Certiﬁcate

Use baidu.com as an example

*openssl s\_client -connect www.baidu.com:443 -showcerts*

Feedbacks are as follows:

CONNECTED(00000003)

depth=2 C = US, O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "(c) 2006 VeriSign, Inc. - For authorized use only", CN = VeriSign Class 3 Public Primary Certification Authority - G5

verify return:1

depth=1 C = US, O = Symantec Corporation, OU = Symantec Trust Network, CN = Symantec Class 3 Secure Server CA - G4

verify return:1

depth=0 C = CN, ST = beijing, L = beijing, O = "BeiJing Baidu Netcom Science Technology Co., Ltd", OU = service operation department., CN = baidu.com

verify return:1

---

Certificate chain

0 s:/C=CN/ST=beijing/L=beijing/O=BeiJing Baidu Netcom Science Technology Co., Ltd/OU=service operation department./CN=baidu.com

i:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 Secure Server CA - G4

-----BEGIN CERTIFICATE-----

MIIIdDCCB1ygAwIBAgIQRgvtzGxo+wBn8JgNuE2/gjANBgkqhkiG9w0BAQsFADB+

MQswCQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAd

BgNVBAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxLzAtBgNVBAMTJlN5bWFudGVj

IENsYXNzIDMgU2VjdXJlIFNlcnZlciBDQSAtIEc0MB4XDTE3MDYyOTAwMDAwMFoX

DTE4MDgxNzIzNTk1OVowgagxCzAJBgNVBAYTAkNOMRAwDgYDVQQIDAdiZWlqaW5n

MRAwDgYDVQQHDAdiZWlqaW5nMTkwNwYDVQQKDDBCZWlKaW5nIEJhaWR1IE5ldGNv

bSBTY2llbmNlIFRlY2hub2xvZ3kgQ28uLCBMdGQxJjAkBgNVBAsMHXNlcnZpY2Ug

b3BlcmF0aW9uIGRlcGFydG1lbnQuMRIwEAYDVQQDDAliYWlkdS5jb20wggEiMA0G

CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDImQ8LQt6/ovSyE1jc5M7knA7yDHOb

ZUA8OgG8ciml3aPKDbuVQ0JLcuFkRBb74nVxe9EAwgN7NMrSL0dSTl2pYonMfklj

7ymfr+LKUigoO8So1XKUsntq6KU7/qXRI4Tpd9zoWoTXUSMgM3BvjN9tyows9GSm

v2foMURsnO00az4YITnM2bVuklui6j4TfkRuA/uypcLPRrZM6XnxRvHrznsMCZ5N

wqpOQeDQuQCGaFqkKPFlbi8Rb+LPFAICUmmqFplit6ac7gvQeLBN3BeJx1Pu4Jmg

oSTPKU0OtRtQtuXaY6+oqpGGP9gqvE5eWYy2YPA2AeojmiNsSPrzf8bDAgMBAAGj

ggTBMIIEvTCCAm8GA1UdEQSCAmYwggJiggsqLmJhaWR1LmNvbYIOKi5iYWlmdWJh

by5jb22CEiouYmFpZHVjb250ZW50LmNvbYIRKi5iYWlkdXN0YXRpYy5jb22CDiou

YmRzdGF0aWMuY29tggsqLmJkaW1nLmNvbYIMKi5iYWlmYWUuY29tggwqLmhhbzEy

My5jb22CCyoubnVvbWkuY29tgg0qLmNodWFua2UuY29tgg0qLnRydXN0Z28uY29t

gg4qLmJhaWR1YmNlLmNvbYIMKi5taXBjZG4uY29tgg4qLmJhaWR1cGNzLmNvbYIN

Ki5iY2Vob3N0LmNvbYIMKi5haXBhZ2UuY29tggsqLmFpcGFnZS5jboILKi5kbG5l

bC5jb22CDyouYmNlLmJhaWR1LmNvbYIQKi5leXVuLmJhaWR1LmNvbYIPKi5tYXAu

YmFpZHUuY29tgg8qLm1iZC5iYWlkdS5jb22CECoubmV3cy5iYWlkdS5jb22CESou

ZmFueWkuYmFpZHUuY29tgg4qLmltLmJhaWR1LmNvbYIQKi5zYWZlLmJhaWR1LmNv

bYIRKi5zc2wyLmR1YXBwcy5jb22CCWJhaWR1LmNvbYIMYmFpZnViYW8uY29tggpi

YWlmYWUuY29tggx3d3cuYmFpZHUuY26CEHd3dy5iYWlkdS5jb20uY26CEmNsaWNr

LmhtLmJhaWR1LmNvbYIQbG9nLmhtLmJhaWR1LmNvbYIQY20ucG9zLmJhaWR1LmNv

bYIQd24ucG9zLmJhaWR1LmNvbYIUdXBkYXRlLnBhbi5iYWlkdS5jb22CD21jdC55

Lm51b21pLmNvbTAJBgNVHRMEAjAAMA4GA1UdDwEB/wQEAwIFoDAdBgNVHSUEFjAU

BggrBgEFBQcDAQYIKwYBBQUHAwIwYQYDVR0gBFowWDBWBgZngQwBAgIwTDAjBggr

BgEFBQcCARYXaHR0cHM6Ly9kLnN5bWNiLmNvbS9jcHMwJQYIKwYBBQUHAgIwGQwX

aHR0cHM6Ly9kLnN5bWNiLmNvbS9ycGEwHwYDVR0jBBgwFoAUX2DPYZBV34RDFIpg

KrL1evRDGO8wKwYDVR0fBCQwIjAgoB6gHIYaaHR0cDovL3NzLnN5bWNiLmNvbS9z

cy5jcmwwVwYIKwYBBQUHAQEESzBJMB8GCCsGAQUFBzABhhNodHRwOi8vc3Muc3lt

Y2QuY29tMCYGCCsGAQUFBzAChhpodHRwOi8vc3Muc3ltY2IuY29tL3NzLmNydDCC

AQIGCisGAQQB1nkCBAIEgfMEgfAA7gB1AN3rHSt6DU+mIIuBrYFocH4ujp0B1VyI

jT0RxM227L7MAAABXPLl8Q4AAAQDAEYwRAIgYmI+Xnom5nr+k+p8RW/BdqobpmTN

ubZW8eM1DTueRfcCIAXdDURAie+w+eZMDfgxHlo/r9wZ8gpQVoTib5w3b/6+AHUA

pLkJkLQYWBSHuxOizGdwCjw1mAT5G9+443fNDsgN3BAAAAFc8uXxRAAABAMARjBE

AiA+PkSvZOvIBmxaBLXLceHH1NqW+Mcrz+xtXNmJOl2ElQIgfVeG4xeuWrb7bpoU

smR+LStIG1TPCwimn3JRE3we/fYwDQYJKoZIhvcNAQELBQADggEBADjrCz8a7cax

h7vpyuUFZ/fiKBHE7VLqfppgf3XYNBoqh21qM6gTGzdiSeZj+vx+KOUn38f080Rg

N2aEkag3n03cufIXR8Yn8haXcusz5PONSlMQnN5rZBwpZ8obItiO8KGOh5lgHQ+s

SloX/j8nDDCQgrNkcG2A78nUT+VxGGENxnPmqajP/O2h/kg02qjcnPoj6Elmm/At

5dWWANX374yS7c0fgLZZ1mfZoIqooaRxsSJl5RzyRNU3Bzv5CZCJCGYFqC3RS28Q

vTCjde7TMsAQiWkZ97IKlUMXdbHManm7K85aWcG4Wg8isr9d2GPUZYgcUSc8KfWY

aP5MzoeU6ug=

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

1 s:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 Secure Server CA - G4

i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5

-----BEGIN CERTIFICATE-----

MIIFODCCBCCgAwIBAgIQUT+5dDhwtzRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB

yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL

ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp

U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW

ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0

aG9yaXR5IC0gRzUwHhcNMTMxMDMxMDAwMDAwWhcNMjMxMDMwMjM1OTU5WjB+MQsw

CQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAdBgNV

BAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxLzAtBgNVBAMTJlN5bWFudGVjIENs

YXNzIDMgU2VjdXJlIFNlcnZlciBDQSAtIEc0MIIBIjANBgkqhkiG9w0BAQEFAAOC

AQ8AMIIBCgKCAQEAstgFyhx0LbUXVjnFSlIJluhL2AzxaJ+aQihiw6UwU35VEYJb

A3oNL+F5BMm0lncZgQGUWfm893qZJ4Itt4PdWid/sgN6nFMl6UgfRk/InSn4vnlW

9vf92Tpo2otLgjNBEsPIPMzWlnqEIRoiBAMnF4scaGGTDw5RgDMdtLXO637QYqzu

s3sBdO9pNevK1T2p7peYyo2qRA4lmUoVlqTObQJUHypqJuIGOmNIrLRM0XWTUP8T

L9ba4cYY9Z/JJV3zADreJk20KQnNDz0jbxZKgRb78oMQw7jW2FUyPfG9D72MUpVK

Fpd6UiFjdS8W+cRmvvW1Cdj/JwDNRHxvSz+w9wIDAQABo4IBYzCCAV8wEgYDVR0T

AQH/BAgwBgEB/wIBADAwBgNVHR8EKTAnMCWgI6Ahhh9odHRwOi8vczEuc3ltY2Iu

Y29tL3BjYTMtZzUuY3JsMA4GA1UdDwEB/wQEAwIBBjAvBggrBgEFBQcBAQQjMCEw

HwYIKwYBBQUHMAGGE2h0dHA6Ly9zMi5zeW1jYi5jb20wawYDVR0gBGQwYjBgBgpg

hkgBhvhFAQc2MFIwJgYIKwYBBQUHAgEWGmh0dHA6Ly93d3cuc3ltYXV0aC5jb20v

Y3BzMCgGCCsGAQUFBwICMBwaGmh0dHA6Ly93d3cuc3ltYXV0aC5jb20vcnBhMCkG

A1UdEQQiMCCkHjAcMRowGAYDVQQDExFTeW1hbnRlY1BLSS0xLTUzNDAdBgNVHQ4E

FgQUX2DPYZBV34RDFIpgKrL1evRDGO8wHwYDVR0jBBgwFoAUf9Nlp8Ld7LvwMAnz

Qzn6Aq8zMTMwDQYJKoZIhvcNAQELBQADggEBAF6UVkndji1l9cE2UbYD49qecxny

H1mrWH5sJgUs+oHXXCMXIiw3k/eG7IXmsKP9H+IyqEVv4dn7ua/ScKAyQmW/hP4W

Ko8/xabWo5N9Q+l0IZE1KPRj6S7t9/Vcf0uatSDpCr3gRRAMFJSaXaXjS5HoJJtG

QGX0InLNmfiIEfXzf+YzguaoxX7+0AjiJVgIcWjmzaLmFN5OUiQt/eV5E1PnXi8t

TRttQBVSK/eHiXgSgW7ZTaoteNTCLD0IX4eRnh8OsN4wUmSGiaqdZpwOdgyA8nTY

Kvi4Os7X1g8RvmurFPW9QaAiY4nxug9vKWNmLT+sjHLF+8fk1A/yO0+MKcc=

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

2 s:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5

i:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certification Authority

-----BEGIN CERTIFICATE-----

MIIE0DCCBDmgAwIBAgIQJQzo4DBhLp8rifcFTXz4/TANBgkqhkiG9w0BAQUFADBf

MQswCQYDVQQGEwJVUzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xNzA1BgNVBAsT

LkNsYXNzIDMgUHVibGljIFByaW1hcnkgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkw

HhcNMDYxMTA4MDAwMDAwWhcNMjExMTA3MjM1OTU5WjCByjELMAkGA1UEBhMCVVMx

FzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZWZXJpU2lnbiBUcnVz

dCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2lnbiwgSW5jLiAtIEZv

ciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJpU2lnbiBDbGFzcyAz

IFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9yaXR5IC0gRzUwggEi

MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1nmAMqudLO07cfLw8

RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbext0uz/o9+B1fs70Pb

ZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIzSdhDY2pSS9KP6HBR

TdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQGBO+QueQA5N06tRn/

Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+rCpSx4/VBEnkjWNH

iDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/NIeWiu5T6CUVAgMB

AAGjggGbMIIBlzAPBgNVHRMBAf8EBTADAQH/MDEGA1UdHwQqMCgwJqAkoCKGIGh0

dHA6Ly9jcmwudmVyaXNpZ24uY29tL3BjYTMuY3JsMA4GA1UdDwEB/wQEAwIBBjA9

BgNVHSAENjA0MDIGBFUdIAAwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cudmVy

aXNpZ24uY29tL2NwczAdBgNVHQ4EFgQUf9Nlp8Ld7LvwMAnzQzn6Aq8zMTMwbQYI

KwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAHBgUrDgMCGgQU

j+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVyaXNpZ24uY29t

L3ZzbG9nby5naWYwNAYIKwYBBQUHAQEEKDAmMCQGCCsGAQUFBzABhhhodHRwOi8v

b2NzcC52ZXJpc2lnbi5jb20wPgYDVR0lBDcwNQYIKwYBBQUHAwEGCCsGAQUFBwMC

BggrBgEFBQcDAwYJYIZIAYb4QgQBBgpghkgBhvhFAQgBMA0GCSqGSIb3DQEBBQUA

A4GBABMC3fjohgDyWvj4IAxZiGIHzs73Tvm7WaGY5eE43U68ZhjTresY8g3JbT5K

lCDDPLq9ZVTGr0SzEK0saz6r1we2uIFjxfleLuUqZ87NMwwq14lWAyMfs77oOghZ

tOxFNfeKW/9mz1Cvxm1XjRl4t7mi0VfqH5pLr7rJjhJ+xr3/

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

---

Server certificate

subject=/C=CN/ST=beijing/L=beijing/O=BeiJing Baidu Netcom Science Technology Co., Ltd/OU=service operation department./CN=baidu.com

issuer=/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 Secure Server CA - G4

---

No client certificate CA names sent

Peer signing digest: SHA256

Server Temp Key: ECDH, P-256, 256 bits

---

SSL handshake has read 5396 bytes and written 431 bytes

---

New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256

Server public key is 2048 bit

Secure Renegotiation IS supported

Compression: NONE

Expansion: NONE

No ALPN negotiated

SSL-Session:

Protocol : TLSv1.2

Cipher : ECDHE-RSA-AES128-GCM-SHA256

Session-ID: FCAAFF8BE98DEE61FF69F0A2854229FFB3DBDDDD4B79E3A70DF7187995AF6DAB

Session-ID-ctx:

Master-Key: 1721C4419A551AB8A9CDEC33798C726DED9A5E743711C8B80DEA6819D8412CD62507212AB9AFD2E88BB0DD999F7D5C31

Key-Arg : None

PSK identity: None

PSK identity hint: None

SRP username: None

TLS session ticket:

0000 - a5 0b a4 32 27 c9 ee 26-94 52 06 0d 91 bd d5 41 ...2'..&.R.....A

0010 - cd a0 f4 77 f0 df 4b b0-0b b7 36 80 a0 42 3e e3 ...w..K...6..B>.

0020 - 70 d0 38 e4 e4 b8 41 f2-eb 43 2b 6a 56 92 4c cd p.8...A..C+jV.L.

0030 - 18 0f 46 b3 9a 54 42 0e-f2 d2 fc 75 0d 4a 64 b9 ..F..TB....u.Jd.

0040 - 6c 17 aa 0f 6d 6d 7b d0-20 26 b3 66 3d d6 5b e4 l...mm{. &.f=.[.

0050 - 51 c6 ad 70 e3 73 16 d0-e5 80 32 f4 9c bb ae 8f Q..p.s....2.....

0060 - 3f c1 1e 64 b1 d4 90 ff-58 6a 3c 88 8e b3 e8 b7 ?..d....Xj<.....

0070 - 69 17 e5 0b a5 17 d0 d3-20 3a ae 6a ba 25 02 af i....... :.j.%..

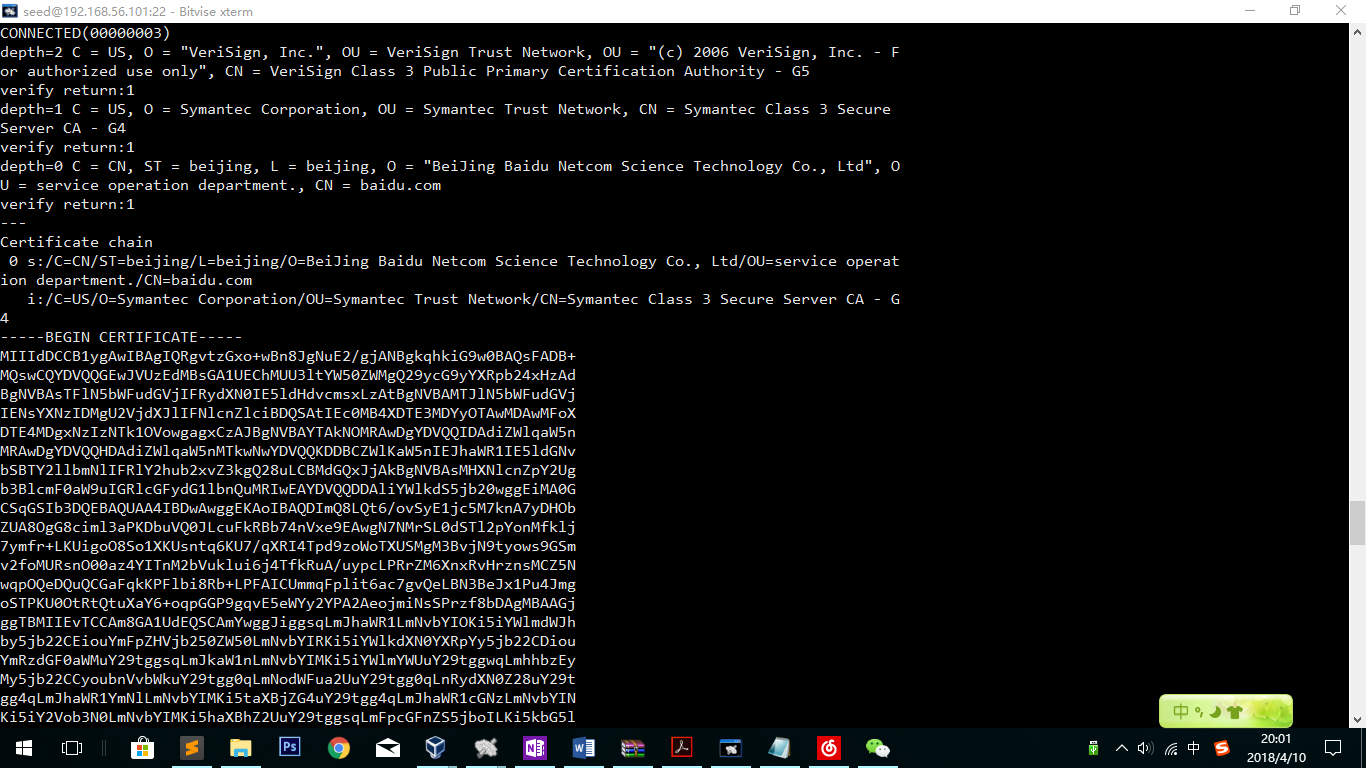
0080 - 8e 3f 03 5e 18 9a a6 1c-80 c8 89 d0 03 fa cc 47 .?.^...........G

0090 - 3d a7 45 67 8f 7f b3 dd-93 8a 79 26 ef 4c 42 87 =.Eg......y&.LB.

Start Time: 1523404767

Timeout : 300 (sec)

Verify return code: 0 (ok)

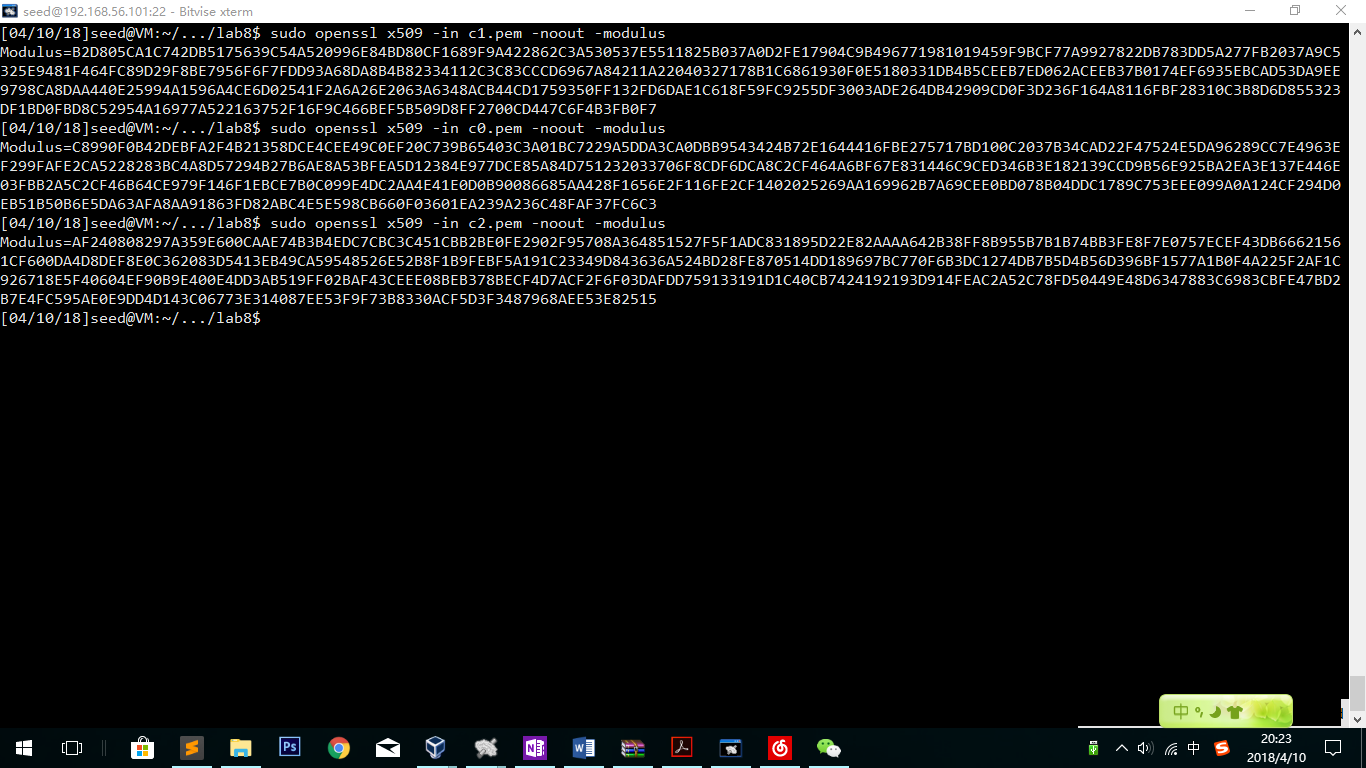


Create these files as c0.pem c1.pem c2.pem

Find n and e

openssl x509 -in c1.pem -noout -modulus

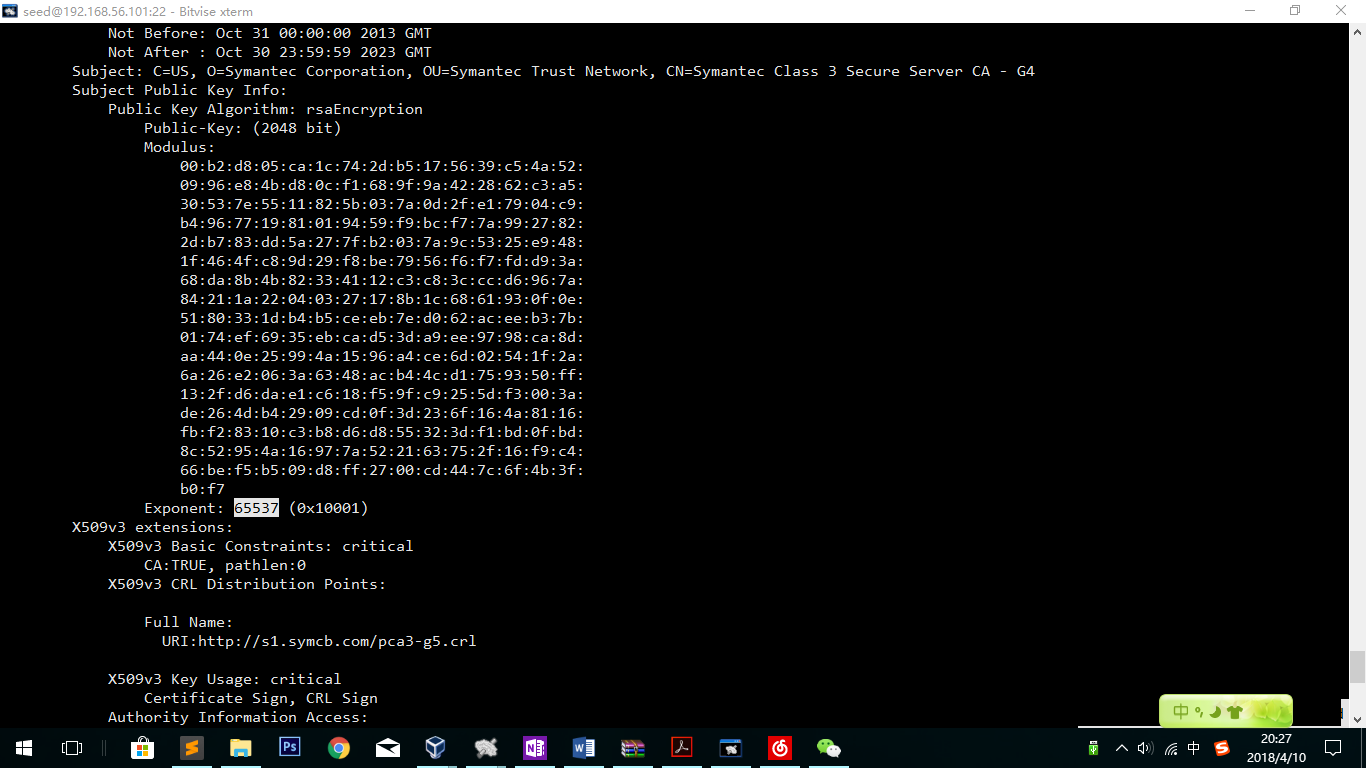
openssl x509 -in c1.pem -text -noout



In this task, we will use c1 to verify. Thus, c2.pem become useless in the rest of the task.

C1’s n value:





E:65537 for all

openssl x509 -in c0.pem -text -noout

Save the file as Signature:

38:eb:0b:3f:1a:ed:c6:b1:87:bb:e9:ca:e5:05:67:f7:e2:28:

11:c4:ed:52:ea:7e:9a:60:7f:75:d8:34:1a:2a:87:6d:6a:33:

a8:13:1b:37:62:49:e6:63:fa:fc:7e:28:e5:27:df:c7:f4:f3:

44:60:37:66:84:91:a8:37:9f:4d:dc:b9:f2:17:47:c6:27:f2:

16:97:72:eb:33:e4:f3:8d:4a:53:10:9c:de:6b:64:1c:29:67:

ca:1b:22:d8:8e:f0:a1:8e:87:99:60:1d:0f:ac:4a:5a:17:fe:

3f:27:0c:30:90:82:b3:64:70:6d:80:ef:c9:d4:4f:e5:71:18:

61:0d:c6:73:e6:a9:a8:cf:fc:ed:a1:fe:48:34:da:a8:dc:9c:

fa:23:e8:49:66:9b:f0:2d:e5:d5:96:00:d5:f7:ef:8c:92:ed:

cd:1f:80:b6:59:d6:67:d9:a0:8a:a8:a1:a4:71:b1:22:65:e5:

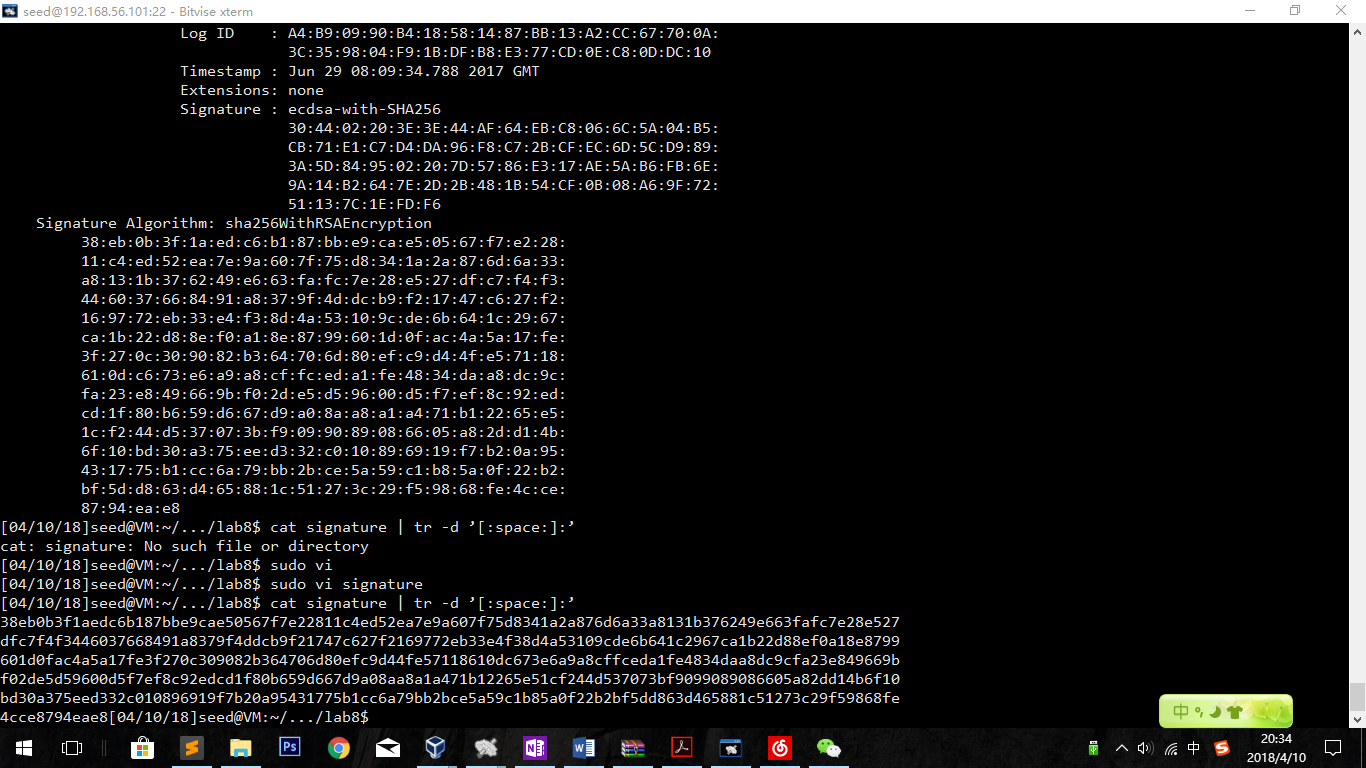
1c:f2:44:d5:37:07:3b:f9:09:90:89:08:66:05:a8:2d:d1:4b:

6f:10:bd:30:a3:75:ee:d3:32:c0:10:89:69:19:f7:b2:0a:95:

43:17:75:b1:cc:6a:79:bb:2b:ce:5a:59:c1:b8:5a:0f:22:b2:

bf:5d:d8:63:d4:65:88:1c:51:27:3c:29:f5:98:68:fe:4c:ce:

87:94:ea:e8



cat signature | tr -d ’[:space:]:’

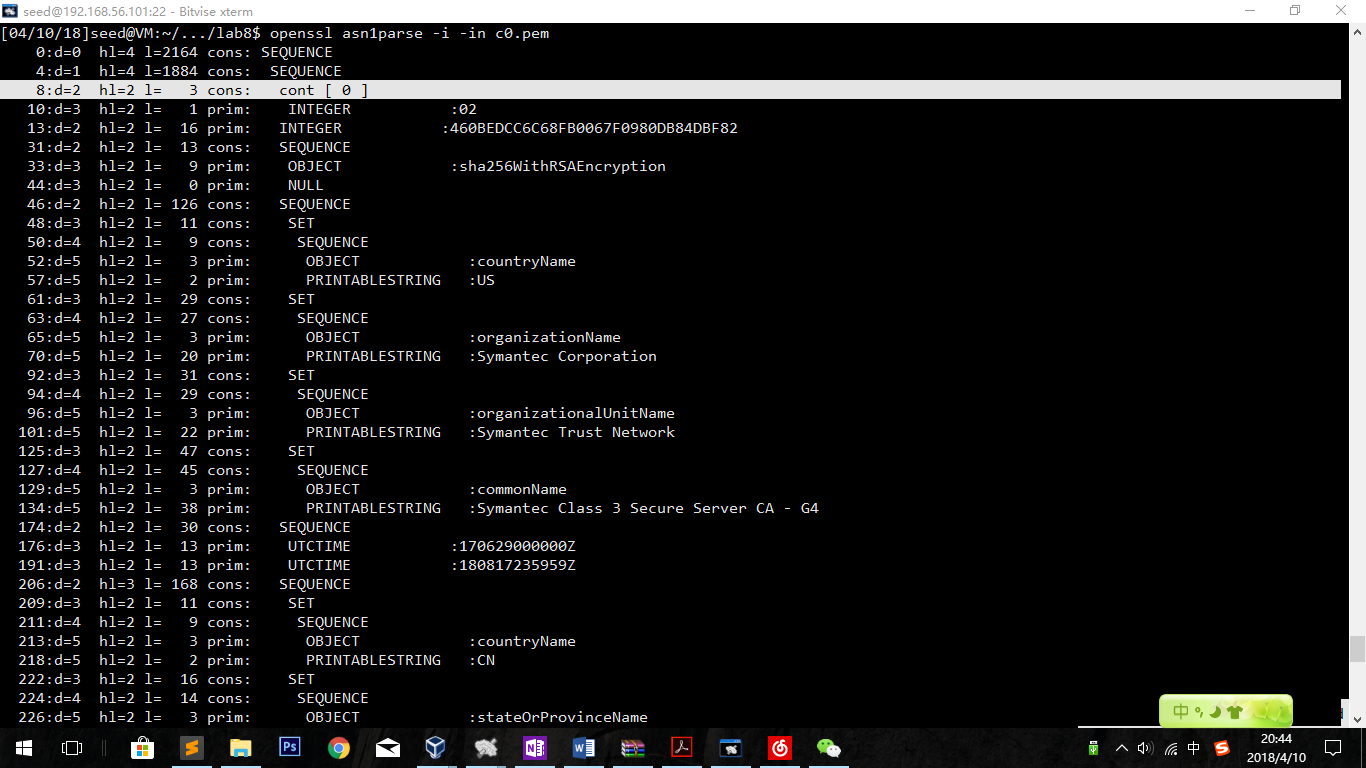
Get the hex file



By typing in the command

openssl asn1parse -i -in c0.pem

we can find that the offset is 4.



openssl asn1parse -i -in c0.pem -strparse 4 -out c0\_body.bin -noout

to get the body part in file c0\_body.bin

Checksum is

ad16fa3193e7cbf66e744ddaec324ceb5faf24bf5c95aae7ea6e78c64dd29c7a

Until now, we get all parameters in our lab:

Signature:



E:65537

N:



M:

ad16fa3193e7cbf66e744ddaec324ceb5faf24bf5c95aae7ea6e78c64dd29c7a

Using the codes provided, we can write codes as follows:

*(The same codes In previous task)*

*#include <stdio.h>*

*#include <openssl/bn.h>*

*#define NBITS 256*

*void printBN(char \*msg, BIGNUM \* a)*

*{*

*/\* Use BN\_bn2hex(a) for hex string*

*\* Use BN\_bn2dec(a) for decimal string \*/*

*char \* number\_str = BN\_bn2hex(a);*

*printf("%s %s\n", msg, number\_str);*

*OPENSSL\_free(number\_str);*

*}*

*BIGNUM\* encrypt(BIGNUM \*ciphertext,BIGNUM \*message,BIGNUM \*publickey,BIGNUM \*theN){*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*BN\_mod\_exp(ciphertext, message, publickey, theN, ctx);*

*return ciphertext;*

*}*

*BIGNUM\* decrypt(BIGNUM \*message,BIGNUM \*ciphertext,BIGNUM \*privatekey,BIGNUM \*theN){*

*BN\_CTX \*ctx = BN\_CTX\_new();*

*BN\_mod\_exp(message, ciphertext, privatekey, theN, ctx);*

*return message;*

*}*

*int main ()*

*{ BN\_CTX \*ctx = BN\_CTX\_new();*

*BIGNUM \*m1 = BN\_new();*

*BN\_hex2bn(&m1,"ad16fa3193e7cbf66e744ddaec324ceb5faf24bf5c95aae7ea6e78c64dd29c7a");*

*BIGNUM \*s = BN\_new();*

*BN\_hex2bn(&s,"");*

*BIGNUM \*e = BN\_new();*

*BN\_hex2bn(&e,"010001");*

*BIGNUM \*n = BN\_new();*

*BN\_hex2bn(&n

*BIGNUM \*ciphertext=BN\_new();*

*encrypt(ciphertext,s,e,n);*

*printBN("This is the decoded message:", ciphertext);*

*BIGNUM \*reg= BN\_new();*

*BN\_hex2bn(&reg,"10000000000000000000000000000000000000000000000000000000000000000");*

*BN\_mod(reg, ciphertext, reg, ctx);*

*if(BN\_cmp(reg,m1)==0){*

*printf("Verify Success!!\n");*

*}*

*else{*

*printf("Verify Denied!!\n");*

*}*

*return 0;*

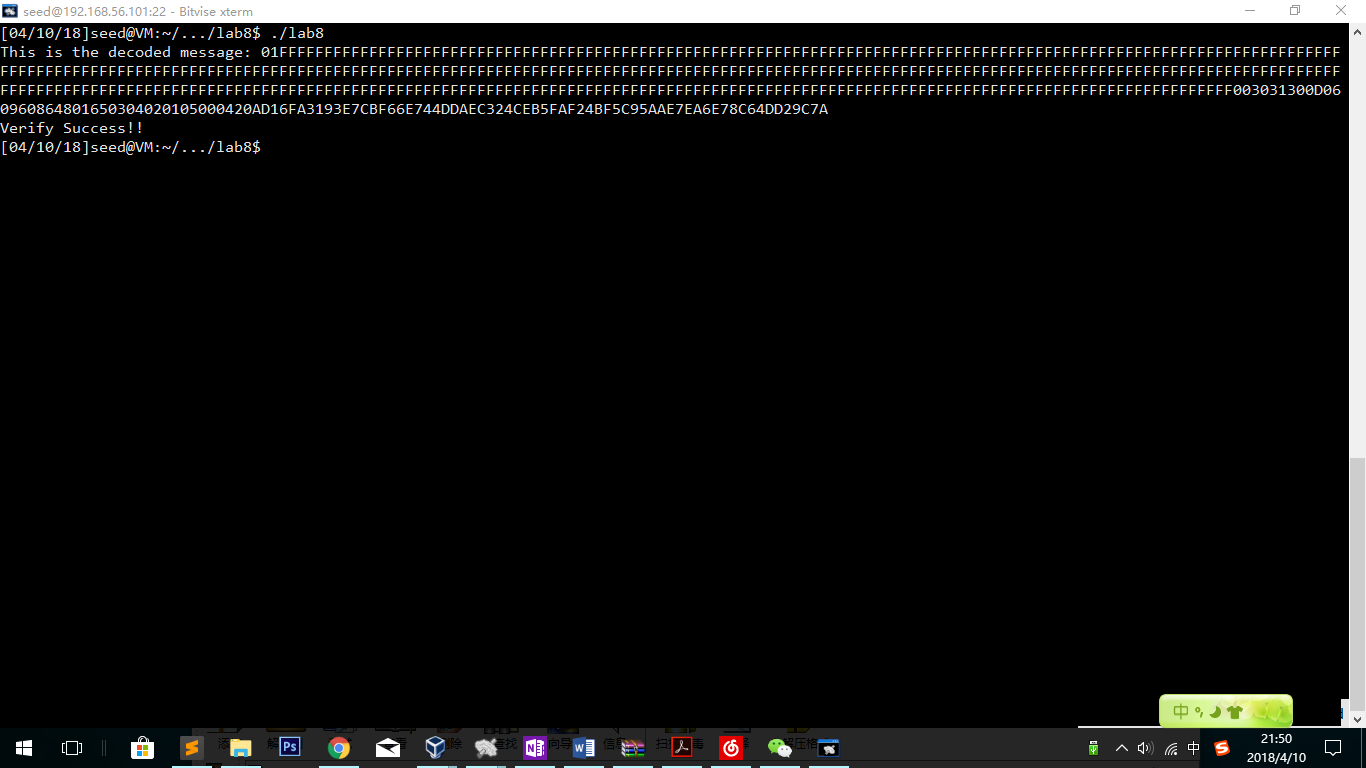
*}*

Codes explanation:

The signature part is the same as previous tasks except the new parameters. The main change is about the judge condition. Hash values will not have long bits to fit the signature algorithm, which means paddings needed. As a result, if we need to judge if the two results have the same ends. I created a BIGNUM which is one bit longer than the hash value. By building the BIGNUM of 1000…,

We can easily get the end of the result.

Result：



Observation：

Verify Success!! Which means that the certification file pass the verifying test!