CS315 · Computer Security

Lab 10

胡玉斌 / 11712121

November 28, 2020

Task 1

We calculate the exgcd: $e \times d + \phi(n) \times y = 1$

The Figure **??** show how to calculate by python with sympy package.

d = 0X3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB.

```
In [1]: p = 0XF7E75FDC469067FFDC4E847C51F452DF
   ...: q = 0XE85CED54AF57E53E092113E62F436F4F
   ...: e = 0X0D88C3

In [2]: d = gcdex(e, (p-1)*(q-1))[0]

In [3]: d
Out[3]: 242122252879047639391600974649432689301398289787956060225838743677206230084 91

In [4]: hex(d)
Out[4]: '0x3587a24598e5f2a21db007d89d18cc50aba5075ba19a33890fe7c28a9b496aeb'

In [5]: (e * d) % ((p - 1) * (q - 1))
Out[5]: 1
```

Figure 1: exgcd

Task 2

Encrypting the message with RSA to calculate $C = M^e$.

C = 0X6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC

To verify that, calculate $M = C^d \bmod n$.

```
[1]: ''.join([hex(ord(c)).replace('0x', '') for c in "A top secret!"])

[1]: '4120746f702073656372657421'

[2]: n = 0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
     e = 65537
     M = 0x4120746f702073656372657421
     d = 0x74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
     C = M**e % n
     print(C, '\n')
     print(str(hex(C)).upper(), '\n')
     print(pow(C, d, n) == M)

     505185253719296845563292113597219490991560578894962423769794023933889 33577436

     0X6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC

     True
```

Figure 2: Encrypt

Task 3

Calculate C^d, we have $M = 0X50617373776F72642069732064656573$

Decode by `utf-8`, we get message `Password is dees`

```
[1]: C = 0x8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493
     n = 0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
     e = 65537
     d = 0x74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
     M = pow(C, d, n)
     str(hex(M)).upper()

[1]: '0X8A1250AA0FCAA6924198D706B60982AA98E1119F9F38634F5057E90FC9D844B5'

[2]: bytes.fromhex('50617373776f72642069732064656573').decode('utf-8')

[2]: 'Password is dees'
```

Figure 3: Decrypt

Task 4

We sign the message by calculating $S = M^d$

We get $S = 0X55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB$ when $M =$ I owe you \$2000.

We get $S = 0XBCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822$ when $M =$ I owe you \$3000.

Although we change `2000` to `3000`, the signature S is changed a lot.

```
[1]: n = 0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
     e = 0x10001
     M = int(b'I owe you $2000.'.hex(), 16)
     d = 0x74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
     S = pow(M, d, n)
     str(hex(S)).upper()

[1]: '0X55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB'

[2]: n = 0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
     e = 0x10001
     M = int(b'I owe you $3000.'.hex(), 16)
     d = 0x74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
     S = pow(M, d, n)
     str(hex(S)).upper()

[2]: '0XBCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822'
```

Figure 4: Sign

Task 5

If the signature is correct, sigen message is the same as actual message.

But it the signature is wrong, the message is changed.

```
[1]: M = 'Launch a missile.'
     S = 0x643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
     e = 0x010001
     n = 0xAE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
     print('Signed Message:', pow(S, e, n))
     print('Actual Message:', int(M.encode().hex(), 16))

Signed Message: 25991004739255778713631969737248063907118
Actual Message: 25991004739255778713631969737248063907118
```

```
[2]: M = 'Launch a missile.'
     S = 0x643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F
     e = 0x010001
     n = 0xAE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
     print('Signed Message:', pow(S, e, n))
     print('Actual Message:', int(M.encode().hex(), 16))

Signed Message: 65710982802337676312353566637294969595512391135201560281913984954792299643540
Actual Message: 25991004739255778713631969737248063907118
```

Figure 5: Verify

Task 6

Step 1

Download a certificate from a real web server.

We use the `www.baidu.com` server.

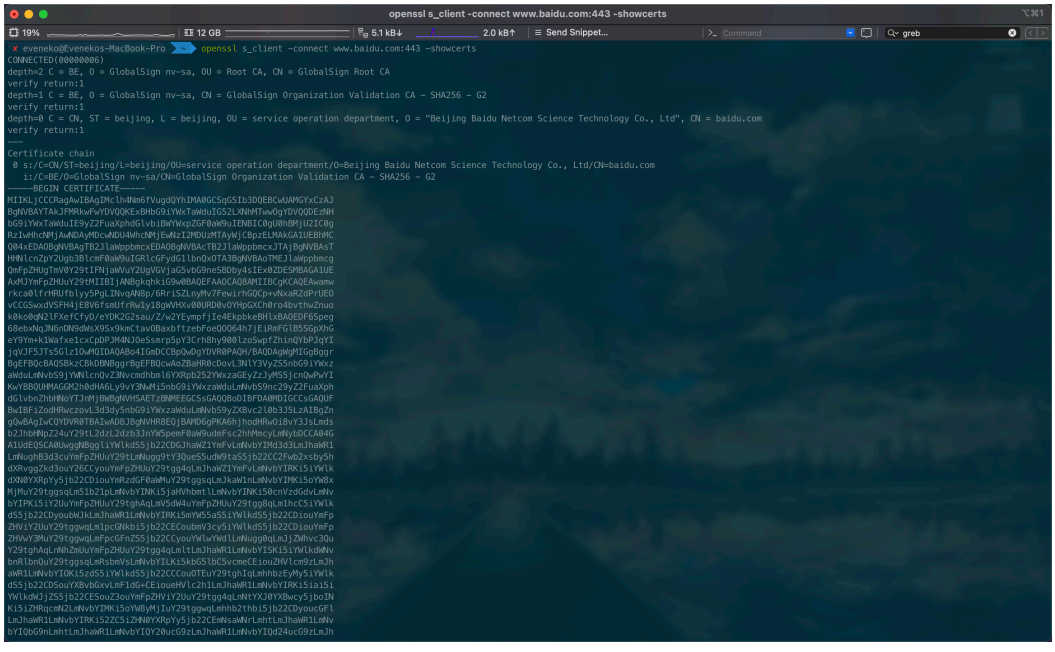

Figure 6: Step 1

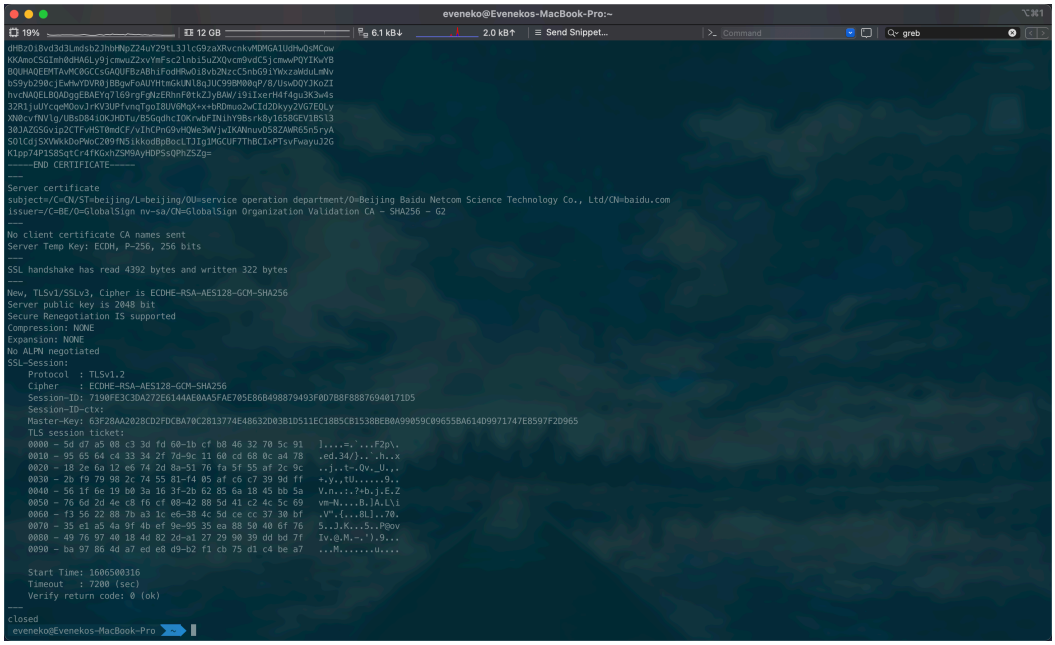

Figure 7: Step 1

Step 2

Extract the public key (e, n) from the issuer’s certificate.

We have n and $e = 0x10001$ now.

Figure 8: Step 2

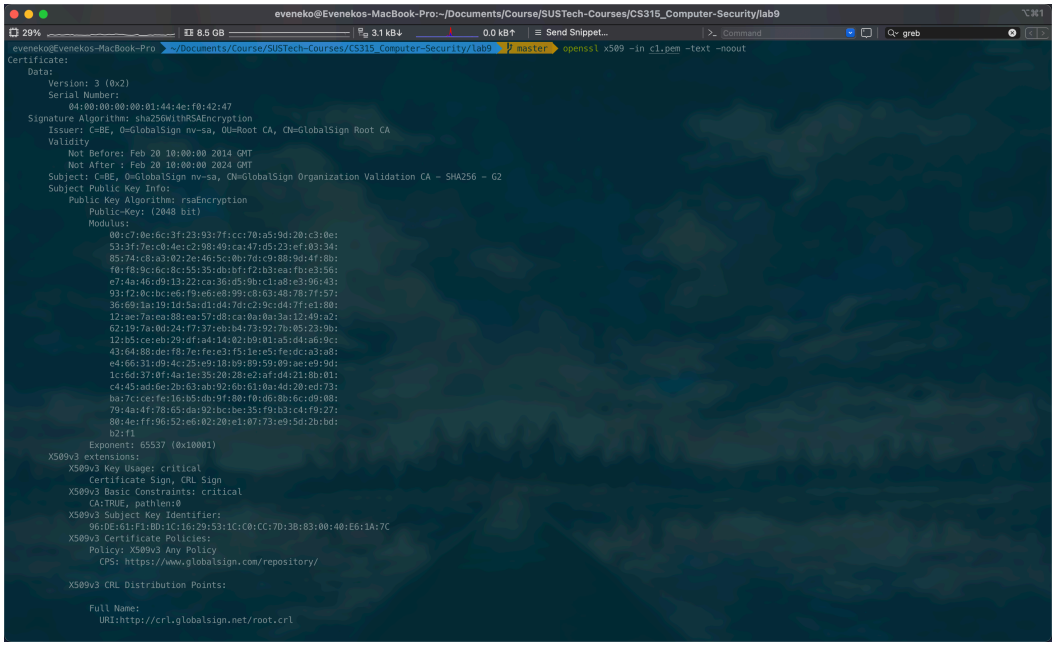

Figure 9: Step 2

Step 3

Extract the signature from the server’s certificate.

We have s now.

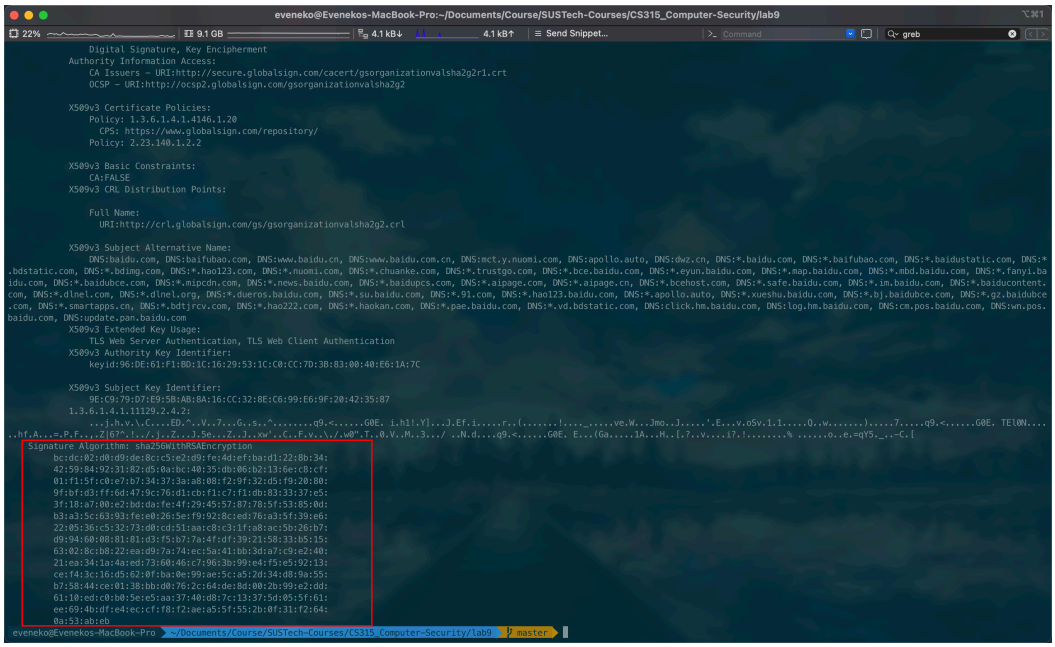

Figure 10: Step 3

Figure 11: Step 3

Step 4

Extract the body of the server’s certificate.

We have $sha256sum$.

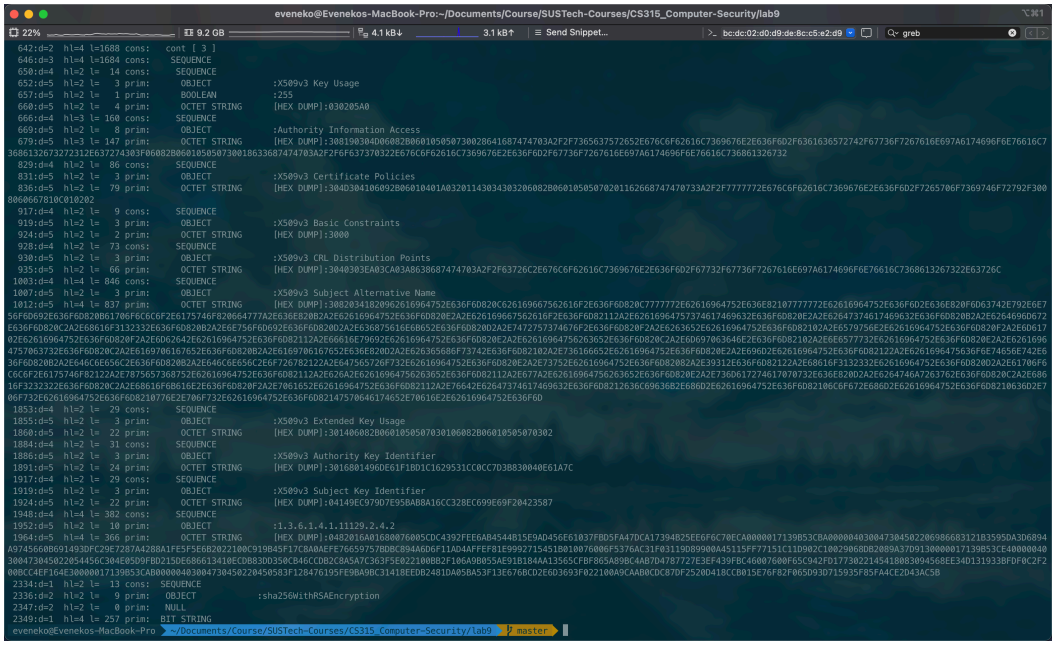

Figure 12: Step 4

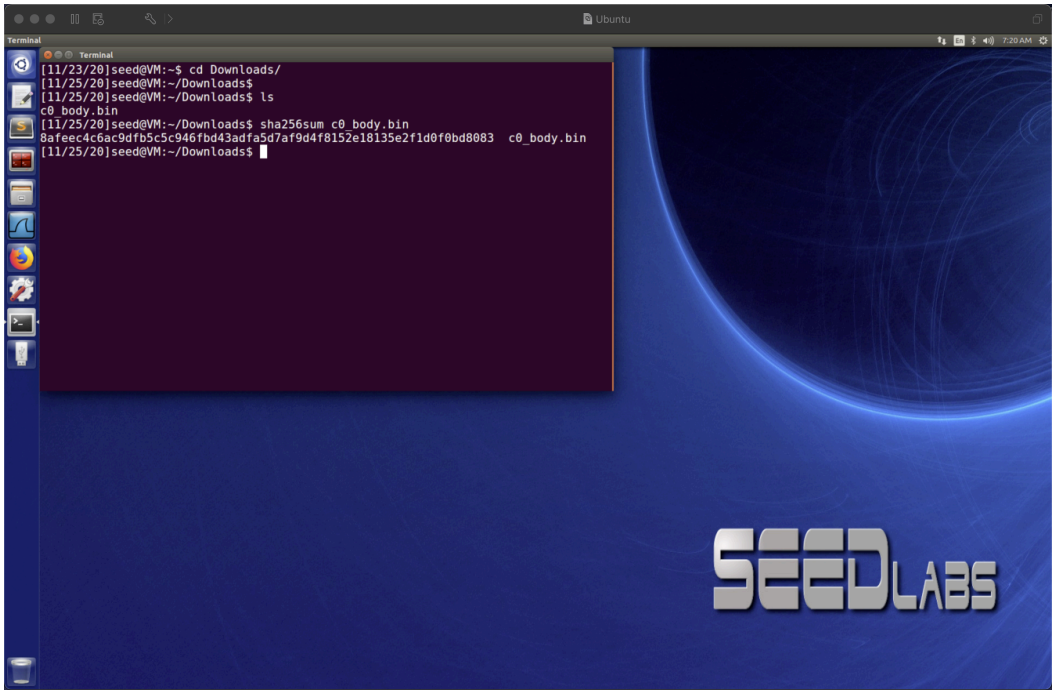

Figure 13: Step 4

Step 5

Verify the signature.

We find that the signature s is the same as the result.

```
[1]: n=0xC70E6C3F23937FCC70A59D20C30E533F7EC04EC29849CA47D523EF03348574C8A3
     e=0x10001
     s=0xbcdc02d0d9de8cc5e2d9fe4defbad1228b34425984923182d50abc4035db06b213
     sha256sum=0x8afeec4c6ac9dfb5c5c946fbd43adfa5d7af9d4f8152e18135e2f1d0f0
     m = pow(s, e, n)
     str(hex(m)).upper()

[1]: '0X1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D06
     09608648016503040201050004208AFEEC4C6AC9DFB5C5C946FBD43ADFA5D7AF9D4F81
     52E18135E2F1D0F0BD8083'
```

Figure 14: Step 5