

Lab 10

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 = 0XD88C3

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

In [3]: d
Out[3]: 24212225287904763939160097464943268930139828978795606022583874367720623008491

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)

5051852537192968455632921135972194909915605788949624237697940239338893
3577436

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 $M = \text{I owe you \$2000}$.

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

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, sign message is the same as actual message.

But if 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: 657109828023376763123535666372949695955123911352015602
81913984954792299643540
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

```
eveneko@Evenekos-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9
eveneko@Evenekos-MacBook-Pro ~ % 8.5 GB 3.1 kB 0.0 kB Send Snippet... Command
eveneko@Evenekos-MacBook-Pro ~ % openssl x509 -in ci.pem -text -noout
Certificate:
Data:
Version: 3 (0x2)
Serial Number:
        04:00:01:00:00:01:44:4e:f0:42:47
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=DE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
Validity
    Not Before: Feb 20 10:00:00 2014 GMT
    Not After : Feb 20 10:00:00 2024 GMT
Subject: C=DE, O=GlobalSign nv-sa, OU=GlobalSign Organization Validation CA - SHA256 - G2
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
            Modulus:
                00:c7:9e:6c:3f:23:93:7f:cc:70:a5:9d:20:c3:9e:
                53:3f:7e:c0:4e:c2:98:49:ca:47:d5:23:ef:03:34:
                00:18:0d:03:0c:35:35:5f:1f:23:81:8a:08:03:
                e7:4a:46:d0:91:37:22:c4:36:d5:9b:c1:a8:e3:90:3:
                93:f2:0c:bc:ee:6f:79:6d:08:99:c8:63:48:78:77:f7:
                36:69:1a:19:1d:5a:01:d1:7d:c2:9c:d4:77:fe:18:0:
                12:ae:7a:ea:88:ea:57:08:ca:0a:0a:3a:12:49:2:
                62:35:1c:3d:2a:39:73:44:14:02:09:01:a0:d0:08:02:
                43:64:89:de:0f:78:7e:ef:3f:51:ce:5f:fed:c4:3:98:
                e4:66:33:09:4c:25:09:18:b9:89:59:09:aa:ee:9:9d:
                1c:fd:37:0f:4a:be:35:20:28:c2:af:d4:21:bb:01:
                c4:45:ad:0e:2b:b6:3a:ab:92:6b:61:0a:4d:20:ed:73:
                ba:0c:fe:ff:66:65:05:0b:9f:08:f0:dc:8u:6c:db:08:
                80:4e:4f:78:65:05:92:bc:be:35:f9:b7:c4:f9:72:
                80:4e:ff:f9:53:22:66:02:20:e1:07:73:ce:96:20:0:
                b2:f1
            Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Key Usage: critical
    CertSign, CRL Sign
X509v3 Extended Key Constraints: critical
    CA:TRUE, pathlen:0
X509v3 Subject Key Identifier:
        96:0E:61:F1:B0:1C:16:29:53:1C:0:0:C:C:70:3B:83:00:40:E6:1A:7C
X509v3 Certificate Policies:
    Policy: X509v3 Any Policy
        CPS: https://www.globalsign.com/repository/
X509v3 CRL Distribution Points:
    Full Name:
        URL:http://crl.globalsign.net/root.crl
```

Figure 9: Step 2

Step 3

Extract the signature from the server's certificate.

We have *s* now.

```
eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9$ curl -k https://www.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt | openssl x509 -noout -pubkey -out publickey.pem
eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9$ curl -k https://www.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt | openssl x509 -noout -signer publickey.pem -out signedcert.pem
eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9$ curl -k https://www.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt | openssl x509 -noout -signer publickey.pem -out signedcert.pem | openssl dgst -sha256 -out sha256sum
eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9$ curl -k https://www.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt | openssl x509 -noout -signer publickey.pem -out signedcert.pem | openssl dgst -sha256 -out sha256sum > sha256sum
```

Figure 10: Step 3

```
eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9$ curl -k https://www.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt | openssl x509 -noout -signer publickey.pem -out signedcert.pem | openssl dgst -sha256 -out sha256sum
eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9$ curl -k https://www.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt | openssl x509 -noout -signer publickey.pem -out signedcert.pem | openssl dgst -sha256 -out sha256sum > sha256sum
```

Figure 11: Step 3

Step 4

Extract the body of the server's certificate.

We have *sha256sum*.

```

eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9
eveneko@Eveneko-MacBook-Pro:~/Documents/Course/SUSTech-Courses/CS315_Computer-Security/lab9> ./lab9
[...]

```

Figure 12: Step 4

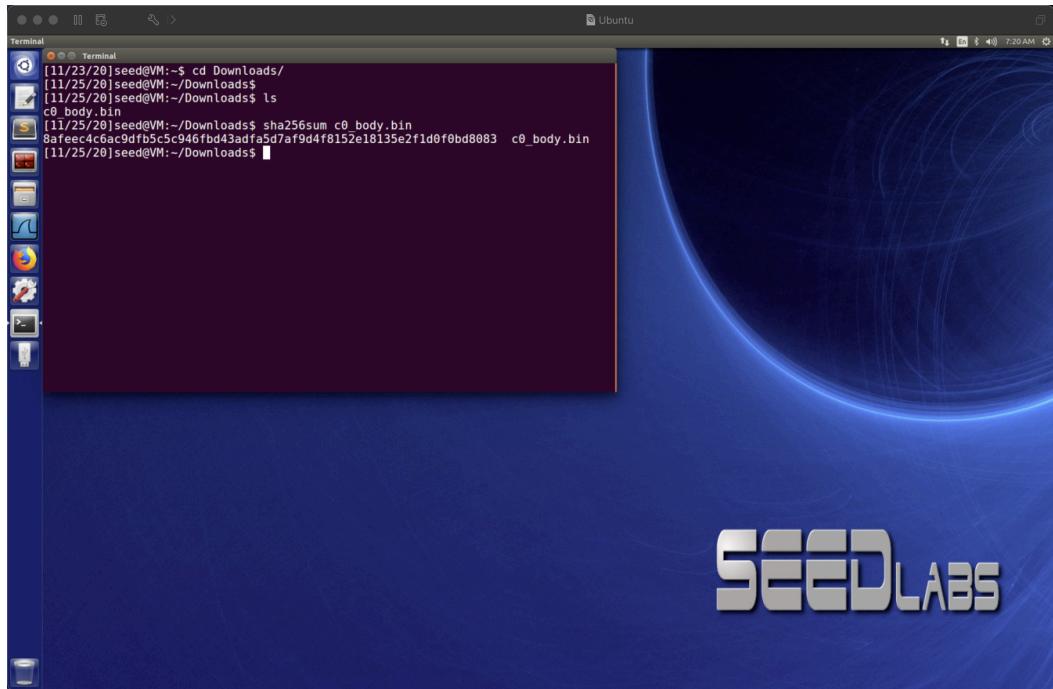


Figure 13: Step 4

Step 5

Verify the signature.

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

```
[1]: n=0xC70E6C3F23937FCC70A59D20C30E533F7EC04EC29849CA47D523EF03348574C8A30  
e=0x10001  
s=0xbcdc02d0d9de8cc5e2d9fe4defbad1228b34425984923182d50abc4035db06b2136  
sha256sum=0x8afeec4c6ac9dfb5c5c946fbd43adfa5d7af9d4f8152e18135e2f1d0f0b  
m = pow(s, e, n)  
str(hex(m)).upper()
```



```
[1]: '0X1FFFFFFFFFFFFFFF09608648016503040201050004208AFEEC4C6AC9DFB5C5C946FBD43ADFA5D7AF9D4F81  
52E18135E2F1D0F0BD8083'
```

Figure 14: Step 5