Web Attack HW3

RSA_1

sol: 1.First get the information in the public key, obtaining N, e

```
└─ openssl rsa -pubin -inform PEM -text -noout < public-key.pem
Public-Key: (640 bit)
Modulus:
    00:ae:5b:b4:f2:66:00:32:59:cf:9a:6f:52:1c:3c:
    03:41:01:76:cf:16:df:53:95:34:76:ea:e3:b2:1e:
    de:6c:3c:7b:03:bd:ca:20:b3:1c:00:67:ff:a7:97:
    e4:e9:10:59:78:73:ee:f1:13:a6:0f:ec:cd:95:de:
    b5:b2:bf:10:06:6b:e2:22:4a:ce:29:d5:32:dc:0b:
    5a:74:d2:d0:06:f1
Exponent: 65537 (0x10001)
```

2. Use factordb to factorize the number N get p,q

Result:		
status (?)	digits	number
FF	193 (show)	3107418240...09$_{<193>}$ = 1634733645...79$_{<97>}$ · 1900871281...71$_{<97>}$

More information

ECM

3. Use formula step by step get the value of r, d and then decrypt the message.

```
In [10]: p = 16347336458092538484431338838650908598417836700330923121811108523893331001045081512121181675115 79
         q = 19008712816648221131268515739354139754718967899685154936666385390880271038021044989571912614655 71

In [11]: r = (p-1)*(q-1)

In [12]: def inv_mod(a, b):
             li = [a, b]
             p, q = a, b
             while p != 1 and q != 1:
                 r = p % q
                 li.append(r)
                 p = q
                 q = r
             li.remove(1)
             li.reverse()
             k, m = 1, -(li[1]//li[0])
             for i in range(len(li)-2):
                 m0 = m
                 m = -(li[i+2]//li[i+1])*m0 + k
                 k = m0
             return m%a

In [13]: d = inv_mod(r,e)

In [20]: n_s = '02797196da55a20737a066fae17f0f0cbcdd41acf84c91f33ac34d71ecc77614ef23fcd53c6098aae4ddb8429f4bca3fe8f79c510f9180e5

In [21]: for i in range(len(n_s)//2):
             print(chr(int(n_s[2*i:2*(i+1)], 16)), end='')

yqÚU¢7 fúá¼ÝA¬øLó:ÃMqìÇvï#üÕ<`ªäÝ¸BKÊ?è+QåQºªkFLAG_IS_WeAK_rSA
```

4. The decrypted message look like something strange and followed by “FLAG_IS_WeAK_rSA”

RSA_2

sol:

1. Followed the step exactly like the rsa_1 but with clearly value.

```
In [28]: c = 832082989951746041747735902982036393605400248712561268928896613457424033149298619391004926666056473166465764865262 1

In [29]: p = 964842302901051567659055174001042653494573763923573980064398935203985250729849139956103500916342705037010757073363 3

In [30]: q = 118748438379802970320924058486536568527609101545433809076500401907042833589092085782510630477324439922306479038875 1

In [31]: e = 65537

In [32]: r = (p-1)*(q-1)
         N = p*q
```

2. Get the decrypted value 557744663355446657776887988 8

Stego_1
sol:

According to 毛弘仁, I found the exact the same problem on the CTF. And the website also provide the source code to solve the problem. By the source code, it seems that inside the png format, there are some space could hide the information.

```
└─ python png_filter.py stego1.png
PNG Signature: ('\x89', 'P', 'N', 'G', '\r', '\n', '\x1a', '\n')
Pos : 8
Type: IHDR
Size: 13
CRC : 5412913F

Pos : 33
Type: IDAT
Size: 10980
CRC : 98F96EEB

Pos : 11025
Type: IEND
Size: 0
CRC : AE426082

Data length in PNG file :  10980
Decompressed data length:  1920800
Flag: DrgnS{WhenYouGazeIntoThePNGThePNGAlsoGazezIntoYou}
```

Strgo_2
sol:

1. Use binwalk to checkout file, it seems that a zip file is hided behind the jpg file.

```
└─ binwalk stego2.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
40804         0x9F64          Zip archive data, at least v2.0 to extract, compressed size: 32993, uncompressed size: 33783,
 name: got2.jpg
73941         0x120D5         End of Zip archive
```

2. split the zip file and then extract it, get “got2.jpg”

```
In [4]: with open('./stego2.jpg', 'rb') as f:
            l = f.read()

In [25]: p = b'\xff\xd9'

In [26]: print(p)
         b'\xff\xd9'

In [27]: l.find(p)
Out[27]: 40802

In [21]: with open('./tmp.zip', 'wb') as f:
             f.write(l[40804:])

In [ ]:
```