

《数据安全与隐私》

课 程 报 告



班级: _____

姓名: _____

学号: _____

日期: _____

课程报告

一、报告内容

1. 课程作业：RSA 明文（编程）
2. 课程作业：RSA 基本原理（编程）
3. 课程作业：取证
4. 课程作业：综合性试题（有时间可挑战）

https://faculty.xidian.edu.cn/JT/zh_CN/article/391792/content/2361.htm#article

二、理论分析

1. openssl 提取出 pubkey.pem 中的参数 n 和 e。

```
openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem
```



```
smilin9@kali: ~/桌面
文件 动作 编辑 查看 帮助

(smilin9@kali)-[~/桌面]
$ openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem
Public-Key: (256 bit)
Modulus:
  00:c2:63:6a:e5:c3:d8:e4:3f:fb:97:ab:09:02:8f:
  1a:ac:6c:0b:f6:cd:3d:70:eb:ca:28:1b:ff:e9:7f:
  be:30:dd
Exponent: 65537 (0x10001)
Modulus=C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgMBAAE=
-----END PUBLIC KEY-----

(smilin9@kali)-[~/桌面]
$
```

通过网站在线将 n 分解为 p, q, n 可以分解为 275127860351348928173285174381581152299 和 319576316814478949870590164193048041239 的乘积。然后用 rsatool 生成私钥文件 private.pem。

```
python rsatool.py -o private.pem -e 65537 -p XXX -q XXX
```

```
(smilin9@kali)-[~/桌面]
$ python rsatool.py -o private.pem -e 65537 -p 2751278603513489281732851743
81581152299 -q 319576316814478949870590164193048041239
Using (p, q) to calculate RSA paramaters

n =
c2636ae5c3d8e43ffb97ab09028f1aac6c0bf6cd3d70ebca281bffe97fbe30dd

e = 65537 (0x10001)

d =
1806799bd44ce649122b78b43060c786f8b77fb1593e0842da063ba0d8728bf1

p = 275127860351348928173285174381581152299 (0xcefb2cf7e18a98ebcdc36e3e7c3b0
2b)

q = 319576316814478949870590164193048041239 (0xf06c28e91c8922b9c236e23560c097
17)

Saving PEM as private.pem

(smilin9@kali)-[~/桌面]
```

用 private.pem 解密 flag.enc。

```
openssl rsautl -decrypt -in flag.enc -inkey private.pem
```

```
(smilin9@kali)-[~/桌面]
$ openssl pkeyutl -decrypt -in flag.enc -inkey private.pem
PCTF{256b_i5_m3dium}
```

2. RSA 求解，已知 p, q, e ，求解私钥 d ，即 e 模 $\phi(n)$ 的逆，其中 $\phi(n) = (p-1) * (q-1)$ 。求逆可以调用 gmpy2 库中的 invert 函数。
3. 从图片里可以看到背后的文件名为 volatility2.5。



4. 首先解 m_1 ，普通 RSA 求解， n 的长度小于 1024 位，通过网站在线分解为 p_0 和 q_0 ，然后正常求解 RSA 的私钥，最后解密即可。其中 n_0 可以分解为 275127860351348928173285174381581152299 和 3487583947589437589237958723892346254777 的乘积。

```
def dem1(n0, p0, q0, c0, e0):
    d0 = libnum.invmod(e0, (p0 - 1) * (q0 - 1))
    m1 = pow(c0, d0, n0) # m1 的十进制形式
    m1 = long_to_bytes(m1) # m1 明文
    print("m1=%s" % m1) # 结果为 b' m1 ' 的形式
```

对 m_2 的加密使用 n 生成了两个公钥 e_1, e_2 ，可采用共模攻击。

```
def dem2(n, c1, c2, e1, e2):

    def egcd(a, b):
```

```
if b == 0:
    return a, 0
else:
    x, y = egcd(b, a % b)
    return y, x - (a // b) * y

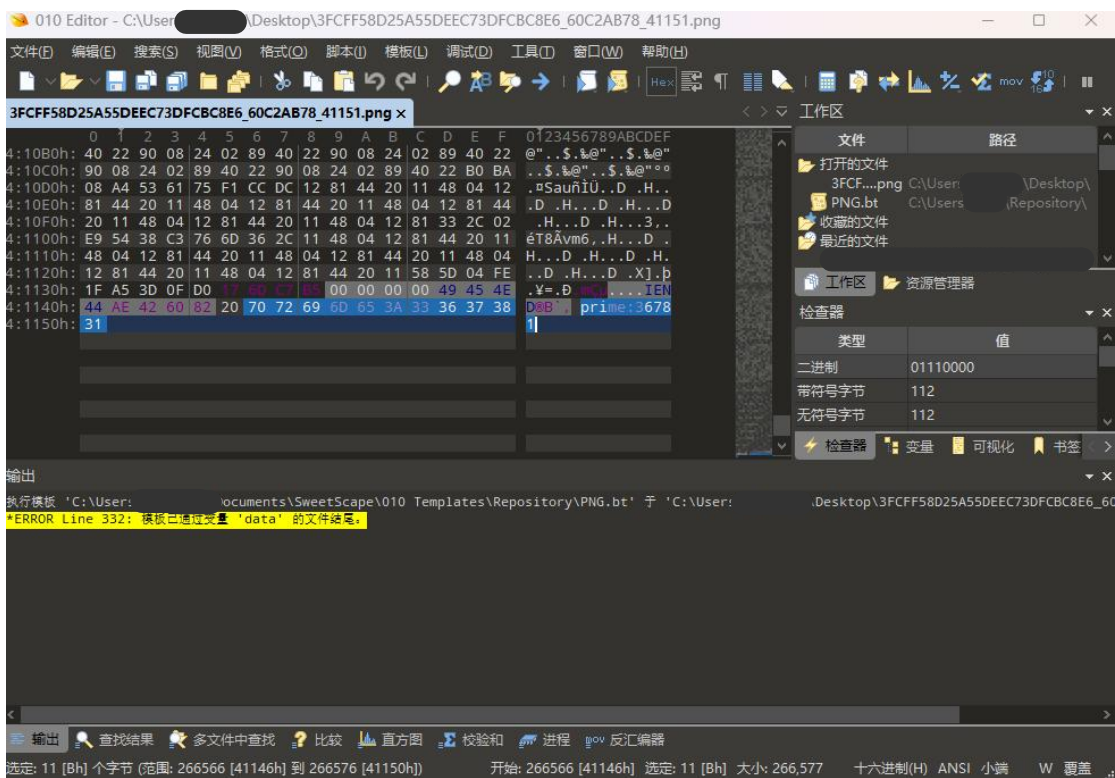
s = egcd(e1, e2)
s1 = s[0]
s2 = s[1]
# 求模反元素
if s1 < 0:
    s1 = -s1
    c1 = invert(c1, n)
elif s2 < 0:
    s2 = -s2
    c2 = invert(c2, n)
m2 = pow(c1, s1, n) * pow(c2, s2, n) % n
m2 = long_to_bytes(m2) # m2 明文
print("m2=%s" % m2) # 结果为 b' m2 ' 的形式
```

运行结果如下：

```
PS C:\Users\> cd "c:\Users\
PS C:\Users> python -u "c:\Users\
m1=b'prime:173'
m2=b'prime:1181'
PS C:\Users>
```

得到 m1=173, m2=1181。

开始没找到 m3 和 m4, 最后猜测是不是因为网页原因图片没展示完整, 把图片保存到本地之后还是没看到, 暴躁之下把图片拖进 010 Editor, 发现图片结尾被塞进了一个数据 “prime:36781”。



得到 $m_3=36781$ ，然后实在找不到 m_4 了，抱着试一试的心态把三个数据乘了一下发现已经是 10 位数了，那么 m_4 一定是一个小素数，从 2 开始尝试发现 2 刚好符合，因此 $m_4=2$ 。

三、编程实现

1. 无编程
2. 源码如下

```
import gmpy2

p = 3487583947589437589237958723892346254777
q = 8767867843568934765983476584376578389
e = 65537
phi = (p - 1) * (q - 1)
print(gmpy2.invert(e, phi))
```

3. 无编程
4. 源码如下

```
import libnum
from Crypto.Util.number import long_to_bytes
from gmpy2 import invert

n0 = 0x8496396a0973fea57cda4bbe85f9d134244a21e7132735908ba604ae75227f613
p0 = 275127860351348928173285174381581152299
q0 = 3487583947589437589237958723892346254777
```



```
c0 = 0x28a34a6a3695d6aff1524aed642f5e60c103952ad23c6746c36e19a4606a9cac7
e0 = 65537

n =
0x00b0bee5e3e9e5a7e8d00b493355c618fc8c7d7d03b82e409951c182f398dee3104580e7ba
70d383ae5311475656e8a964d380cb157f48c951adfa65db0b122ca40e42fa709189b719a4f0
d746e2f6069baf11cebd650f14b93c977352fd13b1eea6d6e1da775502abff89d3a8b3615fd0
db49b88a976bc20568489284e181f6f11e270891c8ef80017bad238e363039a458470f174910
1bc29949d3a4f4038d463938851579c7525a69984f15b5667f34209b70eb261136947fa123e5
49dfff00601883afd936fe411e006e4e93d1a00b0fea541bbfc8c5186cb6220503a94b241311
0d640c77ea54ba3220fc8f4cc6ce77151e29b3e06578c478bd1bebe04589ef9a197f6f806db8
b3ecd826cad24f5324ccdec6e8fead2c2150068602c8dcdc59402ccac9424b790048ccdd9327
068095efa010b7f196c74ba8c37b128f9e1411751633f78b7b9e56f71f77a1b4daad3fc54b5e
7ef935d9a72fb176759765522b4bbc02e314d5c06b64d5054b7b096c601236e6ccf45b5e611c
805d335dbab0c35d226cc208d8ce4736ba39a0354426fae006c7fe52d5267dcfb9c3884f51fd
dfdf4a9794bcfe0e1557113749e6c8ef421dba263aff68739ce00ed80fd0022ef92d3488f76d
eb62bdef7bea6026f22a1d25aa2a92d124414a8021fe0c174b9803e6bb5fad75e186a946a172
80770f1243f4387446ccceb222a965cc30b3929

c1 =
0xe2656d98064060e2d223768e65ddaf83b031c12b1764c02a200ea358d621a1df50dce5aa40
58aaffc9471bb8cafd609f4093adc71167775ecaf53e77b08fee395855e833f6da9060d10dd7
534937e7136323588bc76905a284f1bbef30fc8723ef804eeafaf40d42227b0aaf213dcbd1a8
4aafb8aa0f96a132c648cb2eace527c2510e3c17e071543229f69ede7ec97e511d15b90b6e90
8440b07c19bd5f8db4fa225cc1e790331

c2 =
0xafd609e652925da44893025767bb57f11fd49ee7f1d6b63d21a9aab22e70ea91e404386cf5
7fc79f00629512592cf827c21147297f773f0c12912274d747ed81b4e8dfe01f6f71259dd5c
9c2b9ffbf7e7450fa2eaf9a3e932c0c877f374471d8651d75f5bb5769fcbfbb837636385dadd7
0db774fafecb6f2812c1fc2e06cd73404ae9859e5d30fe2dcf906067325ef6407b01de7093c3
d11685b31b8b924cac7ee5e0444deeff2190875edcc4e4f9d4e991a083dd946b5ad53ecc53f9
63b5211c57e9a78aa717763c45a4ba1c1bfac7687bc0e6e5c1b452faf9232f30356214746fbb
625d20a693d32bf665232daeb82f8460d5b38355ca2a0226e89f12f0d12bf09bd581274c5ed4
7ab1d1caf1e25a37f974c1f0808ae0984108ff69047abbca7f45430b260036d93a318625dfab
0e8c3172db0a58496c0f47ac8af3cc425686f4233d085e88f81680a78a9213405be118f83017
ad6c8a5908d1cb584ec8334c125305d32e17fd2cdf07cb0f06c8b16940b6e171ddfdffd55cee
5d72002b66d4102243dab3dcfa21b5d12b384820ac1a1ba4f83ce8b6c8f3e8cedd8af42ed085
bac111763d416a61b22eed6c5aa52af01851607de198ea1297e4aa9343afefa475f63b5d8502
b9952d8008227f24fa2331254851dd5e9415ca7ad5a255fc5d74ead231f9e360040ff5ee5968
d696a264bf6fdf09e7ad28584690f5cc311494

e1 = 17
e2 = 65537
```

```

def dem1(n0, p0, q0, c0, e0):
    d0 = libnum.invmod(e0, (p0 - 1) * (q0 - 1))
    m1 = pow(c0, d0, n0) # m1 的十进制形式
    m1 = long_to_bytes(m1) # m1 明文
    print("m1=%s" % m1) # 结果为 b' m1 ' 的形式

def dem2(n, c1, c2, e1, e2):

    def egcd(a, b):
        if b == 0:
            return a, 0
        else:
            x, y = egcd(b, a % b)
            return y, x - (a // b) * y

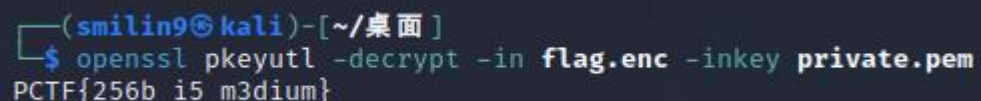
    s = egcd(e1, e2)
    s1 = s[0]
    s2 = s[1]
    # 求模反元素
    if s1 < 0:
        s1 = -s1
        c1 = invert(c1, n)
    elif s2 < 0:
        s2 = -s2
        c2 = invert(c2, n)
    m2 = pow(c1, s1, n) * pow(c2, s2, n) % n
    m2 = long_to_bytes(m2) # m2 明文
    print("m2=%s" % m2) # 结果为 b' m2 ' 的形式

dem1(n0, p0, q0, c0, e0)
dem2(n, c1, c2, e1, e2)

```

四、实验结果

1. 结果如下：



```

(smilin9@kali)-[~/桌面]
$ openssl pkeyutl -decrypt -in flag.enc -inkey private.pem
PCTF{256b_i5_m3dium}

```

PCTF{256b_i5_m3dium}

2. 运行结果如下：

```
PS C:\Users\ [redacted] > cd "c:\Users\ [redacted]\"
PS C:\Users\ [redacted] > python -u "c:\Users\ [redacted]\"rsa.py"
19178568796155560423675975774142829153827883709027717723363077606260717434369
```

19178568796155560423675975774142829153827883709027717723363077606260717434369

3. volatility

4. 运行结果如下:

```
PS C:\Users\ [redacted] > cd "c:\Users\ [redacted]\"
PS C:\Users\ [redacted] > python -u "c:\Users\ [redacted]\"phone_number.py"
m1=b'prime:173'
m2=b'prime:1181'
PS C:\Users\ [redacted] >
```

m3=36781, m4=2, 则手机号为 [redacted]