第一届强网杯密码数学专项赛参赛报告

RSA 公钥密码体制的攻击

Bintou Tover. c10udlnk M0D1

—— Team · Sloth ——

2023.04.23

目录

赛题复述

加密系统描述

该系统是一个混合加密系统，

1) 数据封装（Data Encapsulation Mechanism，DEM）

$$K \xleftarrow{\$} \{0,1\}^{128}$$

$$C_M = \mathrm{Enc}_K^{\mathrm{DEM}}(M)$$

加密系统描述

该系统是一个混合加密系统，

1) 数据封装（Data Encapsulation Mechanism，DEM）

$$K \xleftarrow{\$} \{0,1\}^{128}$$
$$C_M = \mathrm{Enc}_K^{\mathrm{DEM}}(M)$$

2) 基于RSA的密钥封装（Key Encapsulation Mechanism，KEM）

$$N = PQ; \quad P, Q \leftarrow \{0,1\}^{1024}$$
$$C_K = \mathrm{Enc}_{N,e}^{\mathrm{KEM}}(K) := \mathrm{padding}(K)^e \pmod N$$

加密系统描述

该系统是一个混合加密系统，

1) 数据封装（Data Encapsulation Mechanism，DEM）

$$K \xleftarrow{\$} \{0,1\}^{128}$$
$$C_M = \mathrm{Enc}_K^{\mathrm{DEM}}(M)$$

2) 基于RSA的密钥封装（Key Encapsulation Mechanism，KEM）

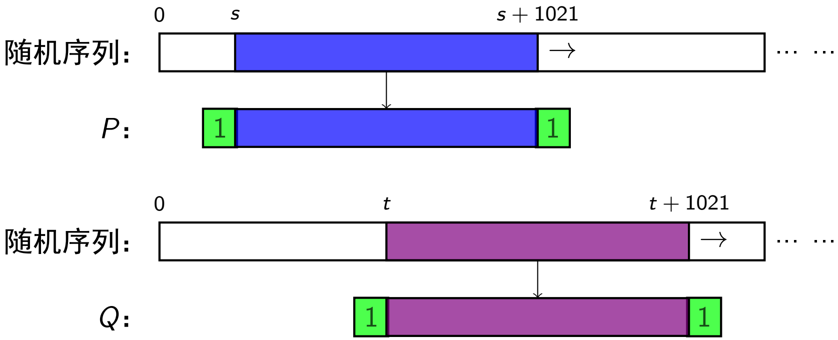
$$N = PQ; \quad P, Q \leftarrow \{0,1\}^{1024}$$
$$C_K = \mathrm{Enc}_{N,e}^{\mathrm{KEM}}(K) := \mathrm{padding}(K)^e \pmod N$$

3) 密文传输

$$\text{Alice} \xrightarrow{(C_M, C_K)} \text{Bob}$$

随机数生成

在一固定的随机比特序列中，以s为起点选择1022比特，首尾拼接上比特1产生P，若P不为素数则起点后移一位后重复该操作，**直到P为素数**。 Q以t为起点按类似方法产生。



题目数据分析

题目数据：

Id.	1	2	3	4	5	6
s	1010	15010	1011	9257	10021	54781
t	3927	65927	8746	77971	92790	69593
e	31	17	131	253	9	89

题目数据分析

题目数据：

Id.	1	2	3	4	5	6
s	1010	15010	1011	9257	10021	54781
t	3927	65927	8746	77971	92790	69593
e	31	17	131	253	9	89

对s和t进行排序：

s_1	s_3	t_1	t_3	s_4	s_5	s_2	s_6	t_2	t_6	t_4	t_5
1010	1011	3927	8746	9257	10021	15010	54781	65927	69593	77971	92790

题目数据分析

题目数据：

Id.	1	2	3	4	5	6
s	1010	15010	1011	9257	10021	54781
t	3927	65927	8746	77971	92790	69593
e	31	17	131	253	9	89

对s和t进行排序：

s_1	s_3	t_1	t_3	s_4	s_5	s_2	s_6	t_2	t_6	t_4	t_5
1010	1011	3927	8746	9257	10021	15010	54781	65927	69593	77971	92790

找出其中差值小于1024的项：

$$\begin{cases} s_3 - s_1 & = 1 \\ s_4 - t_3 & = 511 \\ s_5 - s_4 & = 764 \end{cases}$$

攻击目标（OW-CPA）

已知$(N_i, e_i, s_i, t_i, C_{K_i})$

求K_i

前置知识

素数定理

以下摘抄自[HPS14]:

Definition. For any number X, let

$$\pi(X) = (\# \text{ of primes } p \text{ satisfying } 2 \le p \le X).$$

For example, $\pi(10) = 4$, since the primes between 2 and 10 are 2, 3, 5, and 7.

Theorem 3.21 (The Prime Number Theorem)**.**

$$\lim_{X\to\infty} \frac{\pi(X)}{X/\ln(X)} = 1.$$

How many primes p satisfy $2^{1023} < p < 2^{1024}$? The prime number theorem gives us an answer:

$$\# \text{ of 1024 bit primes} = \pi(2^{1024}) - \pi(2^{1023}) \approx \frac{2^{1024}}{\ln 2^{1024}} - \frac{2^{1023}}{\ln 2^{1023}} \approx 2^{1013.53}.$$

So there should be lots of primes in this interval.

素数密度估计

根据素数定理估算在1024比特奇数中随机选中素数的概率约为：

$$\frac{2^{1013.53}}{(2^{1024}-2^{1023})/2}=\frac{2^{1013.53}}{2^{1022}}<\frac{1}{354}$$

即素数生成枚举约354次会找到一个素数（假设随机）。

最大公因子

定义以下求最大公因子操作：

$$\mathrm{GCD}(a, b) := g$$

输入正整数a和b，输出两者的最大公因子g，
可使用欧几里德算法实现。

素因子高位泄露攻击

素因子泄露一半以上高位比特可被攻击，以下摘抄自[Gal12]的19.4.2节：

Let $N = pq$ and suppose we are given an approximation $\tilde{p}$ to p such that $p = \tilde{p} + x_0$ where $|x_0| < X$. For example, suppose p is a 2κ-bit prime and $\tilde{p}$ is an integer that has the same κ most significant bits as p (so that $|p - \tilde{p}| < 2^\kappa$). Coppersmith used his ideas to get an algorithm for finding p given N and $\tilde{p}$. Note that Coppersmith originally used a bivariate polynomial method, but we present a simpler version following work of Howgrave-Graham, Boneh, Durfee and others.

The polynomial $F(x) = (x + \tilde{p})$ has a small solution modulo p. The problem is that we don't know p, but we do know a multiple of p (namely, N). The idea is to form a lattice corresponding to polynomials that have a small root modulo p and to apply Coppersmith's method to find this root x_0. Once we have x_0 then we compute p as $\gcd(N, F(x_0))$.

Theorem 19.4.2. *Let $N = pq$ with $p < q < 2p$. Let $0 < \epsilon < 1/4$, and suppose $\tilde{p} \in \mathbb{N}$ is such that $|p - \tilde{p}| \le \frac{1}{2\sqrt{2}} N^{1/4-\epsilon}$. Then given N and $\tilde{p}$ one can factor N in time polynomial in $\log(N)$ and $1/\epsilon$.*

$P-1$（或$Q-1$）无大素因子时可通过枚举分解N，以下摘抄自[HPS14]的3.5节：

Remark 3.29. How long does it take to compute the value of $a^{n!} \bmod N$? The fast exponentiation algorithm described in Sect. 1.3.2 gives a method for computing $a^k \bmod N$ in at most $2\log_2 k$ steps, where each step is a multiplication modulo N. Stirling's formula[5] says that if n is large, then $n!$ is approximately equal to $(n/e)^n$. So we can compute $a^{n!} \bmod N$ in $2n\log_2(n)$ steps. Thus it is feasible to compute $a^{n!} \bmod N$ for reasonably large values of n.

Input. Integer N to be factored.

1. Set $a = 2$ (or some other convenient value).
2. Loop $j = 2, 3, 4, \ldots$ up to a specified bound.
   3. Set $a = a^j \bmod N$.
   4. Compute $d = \gcd(a-1, N)^\dagger$.
   5. If $1 < d < N$ then **success**, return d.
6. Increment j and loop again at Step **2**.

† For added efficiency, choose an appropriate k and compute the gcd in Step **4** only every kth iteration.

攻击思路

素因子共用攻击

- 由$s_3 - s_1 = 1011 - 1010 = 1$，结合素数定理得$P_1$和$P_3$大概率相同；

素因子共用攻击

- 由$s_3 - s_1 = 1011 - 1010 = 1$，结合素数定理得$P_1$和$P_3$大概率相同；
- 假设$P_1 = P_3$（$Q_1 \neq Q_3$），则可以通过以下方式分解N_1和N_3：

$$\begin{cases} P_1 = P_3 = \mathrm{GCD}(N_1, N_3) = \mathrm{GCD}(P_1 Q_1, P_3 Q_3) \\ Q_1 = N_1 / P_1 \\ Q_3 = N_3 / P_3 \end{cases}$$

- 由$s_3 - s_1 = 1011 - 1010 = 1$，结合素数定理得$P_1$和$P_3$大概率相同；
- 假设$P_1 = P_3$（$Q_1 \neq Q_3$），则可以通过以下方式分解N_1和N_3：

$$\begin{cases} P_1 = P_3 = \mathrm{GCD}(N_1, N_3) = \mathrm{GCD}(P_1 Q_1, P_3 Q_3) \\ Q_1 = N_1 / P_1 \\ Q_3 = N_3 / P_3 \end{cases}$$

- 实测$P_1 = P_3$成立，通过RSA解密可得 $\mathrm{padding}(K_1)$和$\mathrm{padding}(K_3)$。

填充去除

- 观察padding(K_1)和padding(K_3)，得到填充方式为15个K相接：

$$\text{padding}(K) := 0^{128} \| \overbrace{K \| K \| \ldots \| K}^{15\ K\text{s fused}}$$

填充去除

- 观察padding(K_1)和padding(K_3)，得到填充方式为15个K相接：

$$\text{padding}(K) := 0^{128} \| \overbrace{K \| K \| \ldots \| K}^{15\ K\text{s fused}}$$

- 可使用数学方式表达为：

$$\text{padding}(K) := K \cdot \sum_{i=0}^{15-1} 2^{128 \cdot i}$$

填充去除

- 观察padding(K_1)和padding(K_3)，得到填充方式为15个K相接：

$$\text{padding}(K) := 0^{128} \| \overbrace{K \| K \| \ldots \| K}^{15\ K\text{s fused}}$$

- 可使用数学方式表达为：

$$\text{padding}(K) := K \cdot \sum_{i=0}^{15-1} 2^{128 \cdot i}$$

- 可使用以下方式去除密文的padding：

$$C_K \equiv \text{padding}(K)^e \pmod N$$

$$\text{unpad}_{N,e}(C_K) := C_K \cdot \left(\sum_{i=0}^{15-1} 2^{128 \cdot i}\right)^{-e}$$

$$\equiv K^e \cdot \left(\sum_{i=0}^{15-1} 2^{128 \cdot i}\right)^e \cdot \left(\sum_{i=0}^{15-1} 2^{128 \cdot i}\right)^{-e} \equiv K^e \pmod N$$

小加密指数攻击

- 假设$K_i^{e_i} < N_i$，则$K_i^{e_i} \pmod{N_i} = K_i^{e_i}$，取余操作可忽略；

小加密指数攻击

- 假设$K_i^{e_i} < N_i$，则$K_i^{e_i} \pmod{N_i} = K_i^{e_i}$，取余操作可忽略；
- 此时可通过以下方式恢复K_i：

$$
\begin{aligned}
K_i &= \sqrt[e_i]{\mathrm{unpad}_{N_i,e_i}(C_{K_i})} \\
&= \sqrt[e_i]{K_i^{e_i} \pmod{N_i}} = \sqrt[e_i]{K_i^{e_i}}
\end{aligned}
$$

小加密指数攻击

- 假设$K_i^{e_i} < N_i$，则$K_i^{e_i} \pmod{N_i} = K_i^{e_i}$，取余操作可忽略；
- 此时可通过以下方式恢复K_i：

$$\begin{aligned} K_i &= \sqrt[e_i]{\mathrm{unpad}_{N_i,e_i}(C_{K_i})} \\ &= \sqrt[e_i]{K_i^{e_i} \pmod{N_i}} = \sqrt[e_i]{K_i^{e_i}} \end{aligned}$$

- 在数据5中，$e_5 = 9$，即

$$K_5^{e_5} < 2^{9\cdot 128} < 2^{2047} < N_5$$

所以可用上述方式恢复K_5。

小加密指数攻击——扩展

- 若$K_i^{e_i} \gtrapprox N_i$，但K_i可被分解为$K_i = K_{i1} \cdot K_{i2}$，其中$K_{i1}$为可枚举的小因子，且$K_{i2}^{e_i} < N_i$；

小加密指数攻击——扩展

- 若$K_i^{e_i} \gtrapprox N_i$，但K_i可被分解为$K_i = K_{i1} \cdot K_{i2}$，其中$K_{i1}$为可枚举的小因子，且$K_{i2}^{e_i} < N_i$；
- 此时可通过以下方式枚举K_{i1}以恢复K_i：

$$
\begin{aligned}
K_{i2} &= \sqrt[e_i]{\mathrm{unpad}_{N_i,e_i}(C_{K_i}) \cdot K_{i1}^{-e_i}} \\
&= \sqrt[e_i]{(K_i \cdot K_{i1}^{-1})^{e_i} \pmod{N_i}} = \sqrt[e_i]{(K_{i2})^{e_i}}
\end{aligned}
$$

小加密指数攻击——扩展

- 若$K_i^{e_i} \gtrapprox N_i$，但K_i可被分解为$K_i = K_{i1} \cdot K_{i2}$，其中$K_{i1}$为可枚举的小因子，且$K_{i2}^{e_i} < N_i$；
- 此时可通过以下方式枚举K_{i1}以恢复K_i：

$$
\begin{aligned}
K_{i2} &= \sqrt[e_i]{\mathrm{unpad}_{N_i, e_i}(C_{K_i}) \cdot K_{i1}^{-e_i}} \\
&= \sqrt[e_i]{(K_i \cdot K_{i1}^{-1})^{e_i} \pmod{N_i}} = \sqrt[e_i]{(K_{i2})^{e_i}}
\end{aligned}
$$

- 由于赛题的K随机选取，所以大概率含有小因子；

小加密指数攻击——扩展

- 若$K_i^{e_i} \gtrapprox N_i$，但K_i可被分解为$K_i = K_{i1} \cdot K_{i2}$，其中$K_{i1}$为可枚举的小因子，且$K_{i2}^{e_i} < N_i$；
- 此时可通过以下方式枚举K_{i1}以恢复K_i：

$$
\begin{aligned}
K_{i2} &= \sqrt[e_i]{\mathrm{unpad}_{N_i,e_i}(C_{K_i}) \cdot K_{i1}^{-e_i}} \\
&= \sqrt[e_i]{(K_i \cdot K_{i1}^{-1})^{e_i} \pmod{N_i}} = \sqrt[e_i]{(K_{i2})^{e_i}}
\end{aligned}
$$

- 由于赛题的K随机选取，所以大概率含有小因子；
- 经测试，数据2中的N_2存在符合上述条件的小因子$K_{21} = 477$，所以可用上述方法恢复K_2。

素因子高位泄露攻击

- 由于$s_4 - t_3 = 511 < \frac{1024}{2}$，所以存在$Q_3$泄露$P_4$一半以上高位比特的可能性；

素因子高位泄露攻击

- 由于$s_4 - t_3 = 511 < \frac{1024}{2}$，所以存在$Q_3$泄露$P_4$一半以上高位比特的可能性；
- Q_3已从“素因子共用攻击”中解得，假设Q_3和P_4共用一半以上比特，则可以使用Q_3低位比特，结合[Gal12]的攻击解得P_4；

- 由于$s_4 - t_3 = 511 < \frac{1024}{2}$，所以存在$Q_3$泄露$P_4$一半以上高位比特的可能性；
- Q_3已从“素因子共用攻击”中解得，假设Q_3和P_4共用一半以上比特，则可以使用Q_3低位比特，结合[Gal12]的攻击解得P_4；
- 经过枚举，发现Q_3的低600比特为P_4的高600比特，所以可利用上述方法解出P_4，并通过$Q_4 = \frac{N_4}{P_4}$得Q_4；

素因子高位泄露攻击

- 由于$s_4 - t_3 = 511 < \frac{1024}{2}$，所以存在$Q_3$泄露$P_4$一半以上高位比特的可能性；
- Q_3已从“素因子共用攻击”中解得，假设Q_3和P_4共用一半以上比特，则可以使用Q_3低位比特，结合[Gal12]的攻击解得P_4；
- 经过枚举，发现Q_3的低600比特为P_4的高600比特，所以可利用上述方法解出P_4，并通过$Q_4 = \frac{N_4}{P_4}$得Q_4；
- 最后通过RSA解密得K_4。

大整数分解攻击

- 由于以上方法均不能攻破数据6，所以尝试使用大整数分解算法进行攻击。一般分解方法不能应用于2048比特的N_6，故应尝试特殊分解方法；

大整数分解攻击

- 由于以上方法均不能攻破数据6，所以尝试使用大整数分解算法进行攻击。一般分解方法不能应用于2048比特的N_6，故应尝试特殊分解方法；
- 经实验，使用Pollard's P-1算法可成功分解N_6，耗时约半天；

大整数分解攻击

- 由于以上方法均不能攻破数据6，所以尝试使用大整数分解算法进行攻击。一般分解方法不能应用于2048比特的N_6，故应尝试特殊分解方法；
- 经实验，使用Pollard's P-1算法可成功分解N_6，耗时约半天；
- 得$P_6 - 1$（或$Q_6 - 1$）的最大因子为$1010489929 < 2^{30}$。

```
sage: factor(p-1)
2^2 * 30303659 * 136255523 * 145240507 * 150119441 * 153521413 * 163209461 * 170487
689 * 190371767 * 200187697 * 212947699 * 222541379 * 223949587 * 231905629 * 24785
7163 * 334531961 * 335633647 * 342171407 * 365502869 * 373995463 * 374694233 * 3751
48649 * 383447677 * 524632651 * 587912909 * 647410651 * 649098137 * 688088449 * 713
523623 * 731223781 * 767113531 * 789418043 * 801556729 * 891507751 * 897527753 * 93
7873007 * 1010489929
```

大整数分解攻击

- 由于以上方法均不能攻破数据6，所以尝试使用大整数分解算法进行攻击。一般分解方法不能应用于2048比特的N_6，故应尝试特殊分解方法；
- 经实验，使用Pollard's P-1算法可成功分解N_6，耗时约半天；
- 得$P_6 - 1$（或$Q_6 - 1$）的最大因子为$1010489929 < 2^{30}$。

```
sage: factor(p-1)
2^2 * 30303659 * 136255523 * 145240507 * 150119441 * 153521413 * 163209461 * 170487
689 * 190371767 * 200187697 * 212947699 * 222541379 * 223949587 * 231905629 * 24785
7163 * 334531961 * 335633647 * 342171407 * 365502869 * 373995463 * 374694233 * 3751
48649 * 383447677 * 524632651 * 587912909 * 647410651 * 649098137 * 688088449 * 713
523623 * 731223781 * 767113531 * 789418043 * 801556729 * 891507751 * 897527753 * 93
7873007 * 1010489929
```

- 最后通过RSA解密得K_6。

大整数分解攻击——加速

1) 多次枚举后才检测GCD；
2) 使用新线程检测GCD和记录log数据；
3) 使用C语言或基于Cython的gmpy2库；
4) 加速模幂运算（GMP自带优化）。

Input. Integer N to be factored.

1. Set $a = 2$ (or some other convenient value).

2. Loop $j = 2, 3, 4, \ldots$ up to a specified bound.

  3. Set $a = a^j \bmod N$.

  4. Compute $d = \gcd(a - 1, N)^{\dagger}$.

  5. If $1 < d < N$ then **success**, return d.

6. Increment j and loop again at Step **2**.

† For added efficiency, choose an appropriate k and compute the gcd in Step **4** only every kth iteration.

实现代码

素因子共用攻击

```python
# 'SageMath version 9.4, Release Date: 2021-08-22'
import libnum

# 素因子共用攻击
p1 = p3 = gcd(n1, n3)
q1 = n1 // p1
q3 = n3 // p3
assert p1 != 1 and p1*q1 == n1 and p3*q3 == n3

# RSA解密
d1 = e1.inverse_mod((p1-1)*(q1-1))
d3 = e3.inverse_mod((p3-1)*(q3-1))
m1pad = libnum.n2s(int(pow(c1, d1, n1)))
m3pad = libnum.n2s(int(pow(c3, d3, n3)))
m1 = m1pad[-(128//8):]
m3 = m3pad[-(128//8):]
```

小加密指数攻击

```
# 'SageMath version 9.4, Release Date: 2021-08-22'
import libnum

# 填充去除
padding = sum([2^(128*i) for i in range(15)])
c5np = int(c5 * pow(padding.inverse_mod(n5), e5, n5) % n5)

# 直接开方
m5 = c5np^(1/e5)
```

小加密指数攻击——扩展

```python
# 'SageMath version 9.4, Release Date: 2021-08-22'
import libnum

# 填充去除
padding = sum([2^(128*i) for i in range(15)])
c2np = c2 * pow(padding.inverse_mod(n2), e2, n2) % n2

# 令K2 = a * b，枚举b（大于8比特）
for b in range(2^8, 2^10):
    b = Integer(b)
    # 按前面描述的方法恢复K2
    c2a = int(c2np * pow(b.inverse_mod(n2), e2, n2) % n2)
    m2a = c2a^(1/e2)
    m2 = m2a*b
    # 检测K2是否正确
    if pow(m2, e2) % n2 == c2np:
      print('m2 = %s' % hex(m2))
      break
```

素因子高位泄露攻击

```
# 'SageMath version 9.4, Release Date: 2021-08-22'
# Gal12的算法
def solve(n, ph, pl=1, pbits=1024):
  hbits = ph.nbits()
  lbits = pl.nbits()
  PR.<x> = PolynomialRing(Zmod(n))
  f = ph * 2^(pbits-hbits) + x * 2^lbits + pl
  f = f.monic()
  # beta < 1024 / 2048
  roots = f.small_roots(X=2^(pbits-hbits-lbits), m=10, d=10, beta=0.49)
  if roots:
    pm = Integer(roots[0])
    p = ph * 2^(pbits-hbits) + pm * 2^lbits + pl
    if n % p == 0:
      q = n // p
      return p, q
  return None
```

素因子高位泄露攻击

```python
# 'SageMath version 9.4, Release Date: 2021-08-22'

q3r = bin(q3)[3:-1] # 去除Q3首位的1比特

# 枚举Q3低位比特
for i in range(550, 1024):
  ph = Integer(int('1'+q3r[-i:], 2))
  res = solve(n4, ph) # 调用Gal12的算法
  if res == None: # 枚举错误
    pass
  else: # 枚举成功
    p4, q4 = res
    assert p4 * q4 == n4
    break
```

素因子高位泄露攻击

```python
# 'SageMath version 9.4, Release Date: 2021-08-22'

import libnum

# RSA解密
phi4 = (p4-1) * (q4-1)
d4 = e4.inverse_mod(phi4)
m4pad = libnum.n2s(int(pow(c4, d4, n4)))
m4 = m4pad[-(128//8):]
```

Pollard's P-1 攻击

```python
# Python 3.10.7

DONE = None
# Pollard's P-1算法
def pollard(n, a=2, j=2, B=inf, lfn=None, BIAS=100000):
  global DONE
  counter = j % BIAS - 1 # assert j > 1
  while j < B:
    counter += 1
    if counter == BIAS:
      if DONE != None:
        break
      counter = 0
      _thread.start_new_thread(check, (n, a, j, lfn, ) )
    a = powmod(a, j, n) # 模幂运算
    j += 1
  return DONE
```

```
# Python 3.10.7
def check(n, a, j, lfn): # GCD检测是否分解成功，以及打log
  global DONE
  d = gcd(a-1, n) # 计算GCD
  if d == n: # BIAS过大
    DONE = False
    return
  if d > 1 and d < n: # 检测GCD
    DONE = d
    log = '[Done] d = %d' % d
  else:
    log = '[Debug - %s] j = %d\ta = %d\t d = %d' %
        (time.asctime(time.localtime(time.time())), j, a, d)
  print(log)
  if lfn != None: # log输出到文件
    with open(lfn, 'a') as lf:
      lf.write(log + '\n')
  return
```

Pollard's P-1 攻击

```
# 'SageMath version 9.4, Release Date: 2021-08-22'
import _thread
from math import inf
from gmpy2 import gcd, powmod
import time

# 分解N6
p6 = pollard(n6) # 约半天
q6 = n6 // p6
assert p6 * q6 == n6

# RSA解密
import libnum
phi6 = (p6-1) * (q6-1)
d6 = e6.inverse_mod(phi6)
m6pad = libnum.n2s(int(pow(c6, d6, n6)))
m6 = m6pad[-(128//8):]
```

最终结果

```
m1 = 0x56eebe53d69efebd4505540305375f07
m2 = 0xa35cacb606a75034da5e08922dc0cefd
m3 = 0x7cd11086cad330d2cbbe4fc7e59ee2e5
m4 = 0x76114187a6afac7b315847ee4736d545
m5 = 0x160360acc4078a0b5d46e3860255d30a
m6 = 0xa9ad7e791d2e9d7ee3c11330851f5897
```

CPU： Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
数据1-5： VirtualBox6.1.0 ubuntu18.04 SageMath9.4
数据6： Windows10 Python3.10.7

数据	理论复杂度	实际运行时间	备注
1	$O(\log(N_1))$	$1.98s$	N_1与N_3同规模
2	$O(k_{21}\log(e_2))$	$3.67s$	k_{21}为K_2中8比特以上的最小因子
3	$O(\log(N_3))$	$1.98s$	N_1与N_3同规模
4	$O(\log^2(N_4))$	$4.40s$	
5	$O(\log(e_5))$	$1.95s$	
6	$O(B_6\log(B_6))$	$12h15m09s$	B_6为P_6-1（或Q_6-1）最大素因子的上界

报告总结

总结

- 数据1、3和4的素因子存在信息重叠的问题，其中数据1和数据3的素因子完全一致，可通过求解公因数分解；数据3和数据4的素因子一半以上重叠，可通过Coppersmith算法分解；
- 数据2和数据5存在使用小加密指数的问题，其中数据5可通过直接开方解密，数据2可通过枚举小因子后开方解密；
- 数据6存在使用不安全素因子的问题，由于$P_6 - 1$（或$Q_6 - 1$）素因子过小，故可使用Pollard's P-1算法分解。

改进建议

针对以上问题给出改进该RSA体制的建议：

1) 使用更优的随机数生成算法，以保证每个随机性不被多次使用；
2) 使用恰当的加密指数，建议取$e = 65537$；
3) 使用不可被去除的填充方法，建议填充随机比特；
4) 使用形如$2p + 1$的安全素数，以避免被大整数分解算法分解。

参考文献

[Gal12] Galbraith S D. Mathematics of public key cryptography [M]. Cambridge University Press, 2012.

[HPS14] Hoffstein, Jeffrey, et al. An introduction to mathematical cryptography. Vol. 2. New York: springer, 2014.

[HG97] N. A. Howgrave-Graham, Finding small roots of univariate modular equations revisited. In Cryptography and Coding, volume 1355 of LNCS, pp. 131-142. Springer Verlag, 1997.

[MH20] De Micheli G, Heninger N. Recovering cryptographic keys from partial information, by example [J]. Cryptology ePrint Archive, 2020.

感谢倾听