



山东大学
SHANDONG UNIVERSITY

RSA-CRT 的实现

密码工程

2023 年 12 月 1 日

李 昕 202100460065

21 级密码 2 班

目录

1	实验要求	1
2	实验准备	1
2.1	RSA 算法	1
2.1.1	公钥和私钥的生成	1
2.1.2	RSA 的常规加解密	1
2.2	中国剩余定理加速 RSA 解密	1
2.2.1	中国剩余定理	1
2.2.2	中国剩余定理加速解密	1
2.3	Constant-time ladders	2
3	实验过程	3
3.1	Montgomery ladder	3
3.2	RSA-CRT	5
3.2.1	生成 RSA 所需的参数	5
3.2.2	RSA 加密	7
3.2.3	RSA 解密	7
3.2.4	蒙哥马利模幂保证 Constant-time	8
3.2.5	测试加密结果	8

1 实验要求

问题一 实现 constant-time 的 RSA-CRT，即实现中国剩余定理加速 RSA 解密过程，使用 Montgomery 模幂加速 RSA 中的模幂运算，同时保证 Constant-time 特性。

2 实验准备

2.1 RSA 算法

2.1.1 公钥和私钥的生成

选取两个大素数 p, q ，计算 $n = pq$ 和 $\Phi(n) = (p-1)(q-1)$ 。取 e ，使得 $\gcd(\Phi(n), e) = 1$ 。求满足 $d^e \equiv 1 \pmod{\Phi(n)}$ 的整数 d 。此时 RSA 算法的公钥为 (n, e) ，私钥为 (n, d) 。

2.1.2 RSA 的常规加解密

设明文为 m ，密文为 c ，则计算 RSA 密文公式为 $c = m^e \pmod{n}$ ，相应的对应的解密公式为 $m = c^d \pmod{n}$ 。

2.2 中国剩余定理加速 RSA 解密

2.2.1 中国剩余定理

假设整数 m_1, m_2, \dots, m_n 两两互素，则对于任意的整数 a_1, a_2, \dots, a_n ，方程组：

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases} \quad (1)$$

存在整数解 $x \equiv \sum_{i=1}^n a_i \times \frac{N}{m_i} \times \left[\left(\frac{N}{m_i} \right)^{-1} \right]_{m_i} \pmod{N}$ ，其中 $N = \prod_{i=1}^n m_i$ 。

2.2.2 中国剩余定理加速解密

通过 CRT，我们可以将原本需要使用模数 N 进行解密的操作，转换为两个小模数 p 和 q 上的解密操作，从而提高了解密速度，因为在小模数上进行计算比在大模数上进

行计算更快，即计算 $m = c^d(modp)$ 和 $m = c^d(modq)$ ，具体操作为在生成密钥时，提前计算以下值

$$dP = e^{-1}mod(p-1)$$

$$dQ = e^{-1}mod(q-1)$$

$$qInv = q^{-1}modp(p > q)$$

则解密时只需计算 $m_1 = c^{dP}modp$ 和 $m_2 = c^{dQ}modq$ ，计算 $h = qInv * (m_1 - m_2)modp$ ，最后，明文 $m = m_2 + h * q$

2.3 Constant-time ladders

对于实际密码学应用中，要保持 scalar multiplication 执行时间的一致性，即执行时间与所选择的秘密数据 scalar m 无关，应具有 constant-time 特征，这是避免 timing attacks 的第一步，在使用蒙哥马利模乘的时候，我们需要保证其满足 Constant-time 的特性，除实验 5 学习的蒙哥马利模幂以外，老师在课堂上介绍了一些改进，譬如窗口、滑动窗口、w-非相邻形式的改进，前两者一次关注 w 比特，减少点加法运算；而后者通过改进 d 的表示形式为低密度表示形式，从而减少运算，在本报告中，我尝试使用 Montgomery ladder 的方法实现 constant-time 的模幂运算。

在蒙哥马利模幂算法中，循环的长度取决于 k 的位长，k 为 scalar m 的二进制表示，需要循环长度与 k 无关。

我们首先分析课堂上讲述的重现蒙哥马利 ladders 算法的步骤如下图：

Algorithm 9.5 Montgomery's ladder

INPUT: An element $x \in G$ and a positive integer $n = (n_{\ell-1} \dots n_0)_2$.OUTPUT: The element $x^n \in G$.

1. $x_1 \leftarrow x$ and $x_2 \leftarrow x^2$
 2. **for** $i = \ell - 2$ **down to** 0 **do**
 3. **if** $n_i = 0$ **then**
 4. $x_1 \leftarrow x_1^2$ and $x_2 \leftarrow x_1 \times x_2$
 5. **else**
 6. $x_1 \leftarrow x_1 \times x_2$ and $x_2 \leftarrow x_2^2$
 7. **return** x_1
-

我们应该修改算法的第一步，设置 $(x_1, x_2) \mapsto (1, x)$ ，而不是 $(x_1, x_2) \mapsto (x, x^2)$ ，这样当 i 从 $\ell-2$ 到 0 循环时，只有遇到第一个非零位时， (x_1, x_2) 的值才会发生改变，否则将一直保持不变。从而 scalar 具有固定的长度时，可保证循环的长度也是固定的。该方法来自 Curve25519 曲线中的 `MontgomeryPoint` 及与 `Scalar` 乘积的实现。

3 实验过程

3.1 Montgomery ladder

按照 PPT 中的伪代码，首先实现其 py 代码：

```
1 # coding=utf-8
2 from math import gcd
3 import random
4
5 def Montgomery_ladder(x,n,p):#蒙哥马利阶梯算法
6     # 将y转换为二进制表示形式
7     l = bin(n)[2:]
8
9     # 初始化变量
10    t1 = x
11    t2 = x*x
```

```

12 for i in range(1,len(l)):
13     if l[i] == '0':
14         # 如果当前位为0, 执行加法步骤
15         t2 = (t1 * t2) %p
16         t1 = (t1 * t1)%p
17     elif l[i] == '1':
18         # 如果当前位为1, 执行乘法步骤
19         t1= (t1 * t2) %p
20         t2= (t2 * t2) %p
21 return t1
22
23 for i in range(1,11):
24     print(Montgomery_ladder(2,i,12345))

```

算法核心即为 $k_j = 0$ 时, $X_0 = X_0^2, X_1 = X_0 * X_1, k_j = 1$ 时, $X_1 = X_1^2, X_0 = X_0 * X_1$, 由此, 其运算时间只会随 d 的长度而变化, 不会因为 d 的各位元内容而变化。这可以抵抗旁路攻击中的功率攻击或是计时攻击。

按照实验准备中的分析, 设置 $(x_1, x_2) \mapsto (1, x)$, 而不是 $(x_1, x_2) \mapsto (x, x^2)$, 这样当 i 从 $l-2$ 到 0 循环时, 只有遇到第一个非零位时, (x_1, x_2) 的值才会发生改变, 否则将一直保持不变。从而 scalar 具有固定的长度时, 可保证循环的长度也是固定的。

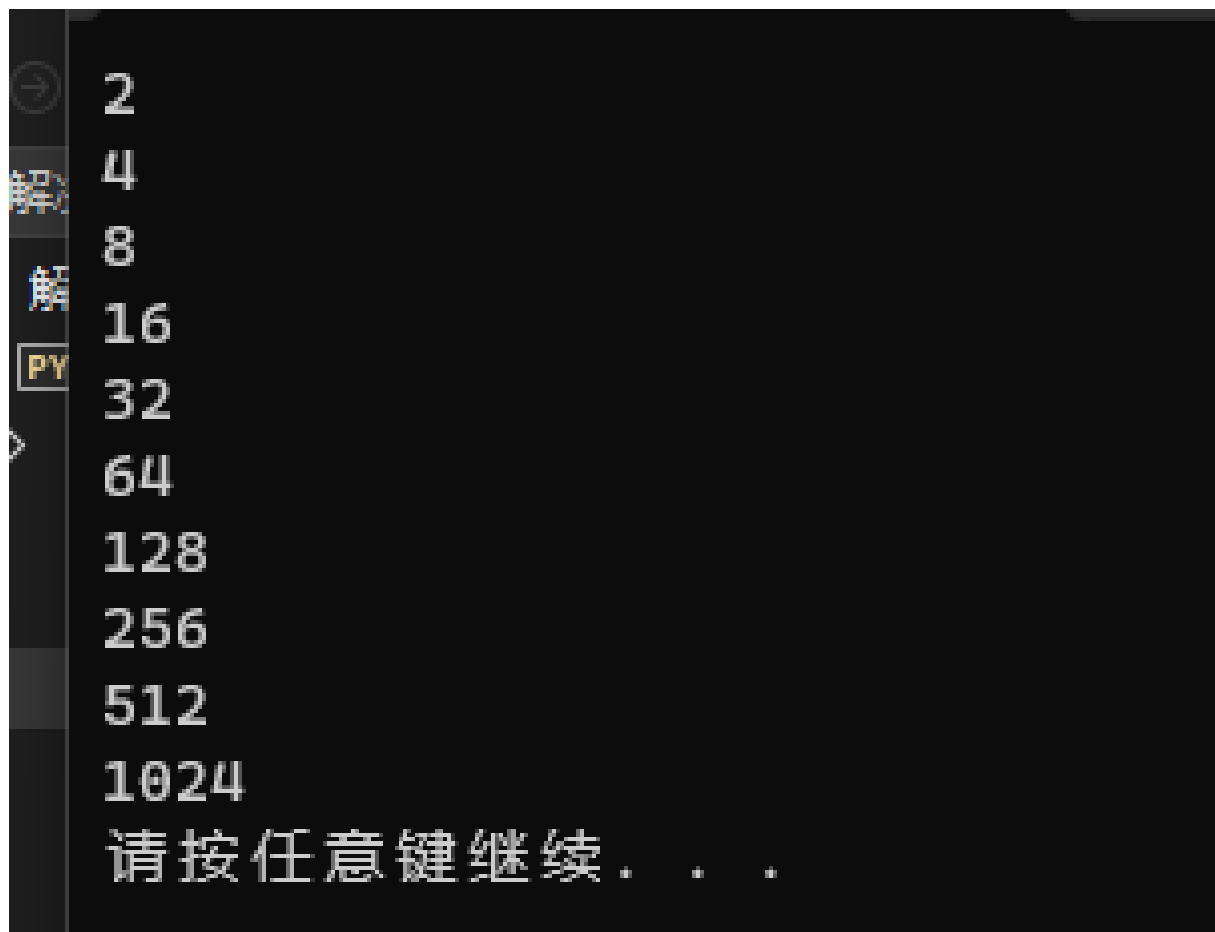
```

1 # coding=utf-8
2 from math import gcd
3 import random
4
5 def Montgomery_ladder(x,n,p):#蒙哥马利阶梯算法
6     # 将y转换为二进制表示形式
7     l = bin(n)[2:]
8
9     # 初始化变量
10    t1 = 1
11    t2 = x
12    for i in range(0,len(l)):
13        if l[i] == '0':
14            # 如果当前位为0, 执行加法步骤
15            t2 = (t1 * t2) %p
16            t1 = (t1 * t1)%p

```

```
17     elif l[i] == '1':
18         # 如果当前位为1, 执行乘法步骤
19         t1 = (t1 * t2) % p
20         t2 = (t2 * t2) % p
21     return t1
22
23 for i in range(1,11):
24     print(Montgomery_ladder(2,i,12345))
```

运行代码, 可以看到成功计算出来了 2 的各个幂次:



```
2
4
8
16
32
64
128
256
512
1024
请按任意键继续 . . .
```

3.2 RSA-CRT

3.2.1 生成 RSA 所需的参数

定义函数 $\text{mod_inverse}(a, m)$ 来求解模逆, 给定两个整数 a 和 m , 使用扩展欧几里德算法计算 $a \bmod m$ 的反元素 x , 使得 $ax \equiv 1 \pmod{m}$ 。:

```

1 def mod_inverse(a, m):
2     def egcd(a, b):
3         if b == 0:
4             return (a, 1, 0)
5         else:
6             (d, x, y) = egcd(b, a % b)
7             return (d, y, x - (a // b) * y)
8
9     d, x, y = egcd(a, m)
10    if d != 1:
11        return None
12    else:
13        return x % m

```

generate_prime(bitlength): 用于生成指定位数的质数。它使用随机数生成器生成一个 *bit_length* 位的整数 *p*，并检查 *p* 是否为奇数且满足费马小定理条件（即 $2^{p-1} = 1 \pmod{p}$ ）。如果满足条件，则返回 *p* 作为质数。

generate_keypair(bitlength): 用于生成 RSA 公钥和私钥对。它首先调用 *generate_prime* 函数生成两个素数 *p* 和 *q*，然后计算 $n = p * q$ 和 $\phi_n = (p - 1) * (q - 1)$ 。选取一个较大的素数 *e* 作为公共指数 *e*，并调用 *mod_inverse* 函数计算私钥 *d*。最后计算 $dp = d \pmod{p-1}$ ， $dq = d \pmod{q-1}$ 和 $qinv = q^{-1} \pmod{p}$ ，并返回公钥 (*n*, *e*) 和私钥 (*n*, *d*, *p*, *q*, *dp*, *dq*, *qinv*)。

```

1 # 生成质数
2 def generate_prime(bit_length):
3     while True:
4         p = random.getrandbits(bit_length)
5         if p % 2 != 0 and pow(2, p-1, p) == 1:
6             #print("find prime")
7             return p
8
9 # 生成公钥和私钥
10 def generate_keypair(bit_length):
11     p = generate_prime(bit_length // 2)
12     q = generate_prime(bit_length // 2)
13     #print(2)

```



```

14 n = p * q
15 phi_n = (p - 1) * (q - 1)
16 e = 65537 # 选取一个较大的素数作为公共指数e
17 d = mod_inverse(e, phi_n)
18 #print(1)
19 dp = d % (p - 1)
20 dq = d % (q - 1)
21 qinv = mod_inverse(q, p)
22 # print("generate_keypair")
23 return (n, e), (n, d, p, q, dp, dq, qinv)

```

3.2.2 RSA 加密

RSA-CRT 对加密没有影响，只需要实现最普通的 RSA 加密即可：

```

1 # RSA加密
2 def rsa_encrypt(m, public_key):
3     n, e = public_key
4     c = pow(m, e, n)
5     return c

```

3.2.3 RSA 解密

由实验准备中的分析可知，给定密文 c 和私钥 $(n, d, p, q, dp, dq, qinv)$ ，使用中国剩余定理计算解密后的明文 m ，只需计算 $m_1 = c^{dP} \bmod p$ 和 $m_2 = c^{dQ} \bmod q$ ，计算 $h = qInv * (m_1 - m_2) \bmod p$ ，最后，明文 $m = m_2 + h * q$ 。代码如下：

```

1 # RSA解密
2 def rsa_decrypt(c, private_key):
3     n, d, p, q, dp, dq, qinv = private_key
4     m1 = pow(c, dp, p)
5     m2 = pow(c, dq, q)
6     h = (qinv * (m1 - m2)) % p
7     m = m2 + h * q
8     return m

```

3.2.4 蒙哥马利模幂保证 Constant-time

将加密和解密中的模幂运算均替换为我们实现的蒙哥马利阶梯算法如下：

```
1 # RSA加密
2 def rsa_encrypt(m, public_key):
3     n, e = public_key
4     #c = pow(m, e, n)
5     c=Montgomery_ladder(m, e, n)
6     return c
7
8 # RSA解密
9 def rsa_decrypt(c, private_key):
10    n, d, p, q, dp, dq, qinv = private_key
11    #m1 = pow(c, dp, p)
12    m1=Montgomery_ladder(c, dp, p)
13    #m2 = pow(c, dq, q)
14    m2=Montgomery_ladder(c, dq, q)
15    h = (qinv * (m1 - m2)) % p
16    m = m2 + h * q
17    return m
```

3.2.5 测试加密结果

运行代码加密 12345，可以看到结果正确。说明我们实现了 Constant-time 的 RSA-CRT 算法：

```
c: 305445888349715127642684258981324265954794352825829806903922000889237582653021417954214704025004240657004308026576763
4835828998372531198198211600641077094
p: 12345
请按任意键继续...
```

参考文献

- [1] WIKI 百科-椭圆曲线的标量乘法.<https://zh.wikipedia.org/wiki/橢圓曲綫的純量乘法>
- [2] Koc C K, Paar C. The Montgomery Powering Ladder[C]. CHES 2002, LNCS 2523, pp. 291–302, 2003.