登录

Rabin加密算法和n次同余方程

r1ngs / 2019-05-14 09:18:00 / 浏览数 4212 安全技术 密码学 顶(0) 踩(0)

前言

Rabin算法是一种基于计算模合数平方根困难性问题的非对称加密算法。他和RSA加密的形式类似，本文主要探讨Rabin算法的特殊情况和n次同余方程的解法。

Rabin算法

加密

选择两个大素数p和q做为私钥

计算n = p * q做为公钥

若明文为m，则密文为c ≡ m^2 (mod n)

解密

1. 首先计算出r和s，目的是满足：

$$
\begin{aligned}
r &\equiv \sqrt{c}\ (mod\ p) \\
s &\equiv \sqrt{c}\ (mod\ q)
\end{aligned}
$$

这里的根号只是一种表示形式，表示 $r^2 \equiv c\ (mod\ p)$ 而已，并不是要真的去开方。通常选择p和q都是模4余3的数，那么由欧拉判别定理：

- 如果p为素数且gcd(a, p)=1，那么a是模p的平方剩余的充要条件是

$$a^{(p-1)/2} \equiv 1 (mod\ p)$$

把上式的a换成c，即
$c^{(p-1)/2} \equiv 1\ (mod\ p)$
，那么把它带入
$r^2 \equiv c\ (mod\ p)$
容易得到
$r^2 \equiv c \cdot c^{(p-1)/2} \equiv c^{(p+1)/2} \equiv (c^{(p+1)/4})^2$
，则易得：

$$
\begin{aligned}
r &\equiv c^{\frac{1}{4}(p+1)}\ (mod\ p) \\
s &\equiv c^{\frac{1}{4}(q+1)}\ (mod\ q)
\end{aligned}
$$

r和s的解其实有两个（一正一负），但这里只要满足条件就行了，所以任意取一个正的就可以了

从这里可以看出来如果p和q不是模4余3的话，c的指数就不是一个整数，也就不能用这个方法计算了

1. 用扩展欧几里得算法计算出a和b，使得：

ap + bq ≡ 1 (mod n)

1. 解出四个明文：

$$
\begin{aligned}
x_1 &= (a \cdot p \cdot s + b \cdot q \cdot r) \bmod n \\
x_2 &= n - x_1 \\
x_3 &= (a \cdot p \cdot s - b \cdot q \cdot r) \bmod n \\
x_4 &= n - x_3
\end{aligned}
$$

1. 算法正确性证明：

$$x_1^2 \equiv (a \cdot p \cdot s + b \cdot q \cdot r)^2 \equiv (a \cdot p \cdot s)^2 + (b \cdot q \cdot r)^2 \ (mod\ n)$$

$$由 s^2 = k_1 q + c 和 r^2 = k_2 p + c$$

$$则：\ x_1^2 \equiv a^2 \cdot p^2 \cdot s^2 + b^2 \cdot q^2 \cdot r^2 \equiv c(a^2 \cdot p^2 + b^2 \cdot q^2)\ (mod\ n)$$

$$则：\ x_1^2 \equiv c((a \cdot p + b \cdot q)^2 - 2a \cdot p \cdot b \cdot q) \equiv c\ (mod\ n)$$

对于其他的解也同理可证

p、q模4不余3

通常情况下选择p和q的时候，一般来说是选择模4余3的素数的，但是如果二者不是模4余3的话，也是可以解的：

> 由m^2 ≡ c (mod n) 分解为两个同余式：m^2 ≡ c (mod p)和m^2 ≡ c(mod q)
>
> 对于上面的方程，如果模数是合数n的话计算是困难的，但如果是素数的话就比较容易，可以用二次同余方程的通用解法得到m ≡ C1 (mod p)和m ≡ C2(mod q)
>
> 其中C1和C2均有两个不同的解，然后再用中国剩余定理联立上面两个方程就能解得m ≡ M (mod pq)了

其中二次同余方程解法的通用算法可以参考Cipolla's algorithm，这里做一个简单的解释：

这个算法主要处理形如x^2 ≡ n (mod p)， 其中p是奇素数的方程的x的解，解法如下：

设a满足w=a^2-n不是模p的二次剩余，即由欧拉定理

$$w^{\frac{p-1}{2}} \equiv -1\ (mod\ p)$$

，那么

$$x \equiv (a + \sqrt{w})^{\frac{p+1}{2}}$$

即为解，实现这个算法的话，主要的问题是根号w可能是负数，可以定义实部和虚部利用快速幂算法进行计算，使得最终结果抵消掉根号，时间复杂度为O(log N)

测试的demo如下：

```
from Crypto.Util.number import getPrime
from random import randint
from libnum import *
from gmpy2 import *


def power(s1, s2, k1, k2, w, p):
   return ((s1*k1+s2*k2*w)%p, (s1*k2+s2*k1)%p)

def Cipolla_algorithm(p, n):
   a = randint(1, p)
   w = a ** 2 - n

   while pow(w, (p-1)/2, p) !=p-1 :
       a = randint(1, p)
       w = a ** 2 - n

   times = (p+1)/2

   k1 = 1
   k2 = 0

   first = True

   sum1 = 1
   sum2 = 0
   while times != 0:
       if first:
           k1, k2=power(k1, k2, a, 1, w, p)
           first = False
```

```
        else:
            k1, k2=power(k1, k2, k1, k2, w, p)

        if times & 1:
            sum1, sum2 = power(sum1, sum2, k1, k2, w, p)
        times >>= 1

    return sum1

def CRT(c, n):
    for i in range(len(n)):
        for j in range(i + 1, len(n)):
            assert gcd(n[i], n[j]) == 1
    assert len(c) == len(n)

    N = reduce(lambda a, b: a*b, n)
    x = 0
    for i, j in zip(c, n):
        N_i = N/j
        N_i_1 = invert(N_i, j)
        x+=i*N_i*N_i_1
    return x % N

if __name__ == '__main__':
    p = getPrime(512)
    while p % 4 != 1:
        p = getPrime(512)

    q = getPrime(512)
    while q % 4 != 1:
        q = getPrime(512)
    n = p * q
    m = s2n('this is plaintext')
    c = pow(m, 2, n)
    print 'm is '+str(m)

    get_x1 = Cipolla_algorithm(p, c)
    get_x2 = Cipolla_algorithm(q, c)


    assert  pow(get_x1, 2, p) == c % p
    assert  pow(get_x2, 2, q) == c % q

    c11 = get_x1
    c12 = p-get_x1
    c21 = get_x2
    c22 = q-get_x2

    print 'possible m :' + str(CRT([c11, c21], [p, q]))
    print 'possible m :' + str(CRT([c11, c22], [p, q]))
    print 'possible m :' + str(CRT([c12, c21], [p, q]))
    print 'possible m :' + str(CRT([c12, c22], [p, q]))
```

n次同余方程

这样的话也就衍生了一个通用的解法，就是如果题目给的是m^e ≡ c (mod n)，其中p、q已知但是e和phi(n)不互素的话，就可以利用上面的算法先解出m ≡ C1 (mod p)和m ≡ C2(mod q)再用CRT联合

但也许你会问m^3 ≡ c (mod p)或者m^n ≡ c (mod p)，这类方程如何求解，事实上这类方程的计算是困难的，针对这个问题，python有sympy库可以进行处理，算法应该是利用的原根进行计算，所以当p比较大的时候计算t

```
from Crypto.Util.number import getPrime
from random import randint
from sympy.ntheory.residue_ntheory import nthroot_mod

p = getPrime(150)
x = randint(1, p)
n = pow(x, 4, p)
```

```
x1 = nthroot_mod(n,4,p,all_roots=False)
assert pow(x1, 4, p) == n
print x1
```

但是如果遇到这样的题目场景，且p、q不是特别大的话，也可以尝试一下，说不定就能解呢 :D

总结

Rabin加密算法通常在CTF里和RSA结合起来考，如果上面的看的不是很明白的话，在以后遇到这种类似的情况(p、q模4不余3 或者上面说到n次同余方程的场景)，就可以直接用我的exp解了，不用再去暴破了 :D

上一篇：pwn return-to-dl-... 下一篇：Uber Bug Bounty：将...