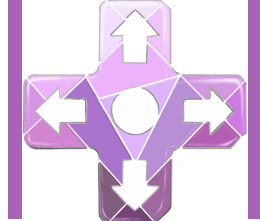


# 密码学： RSA加密和相关攻击

涅普高校人才培养计划

WEB | PWN | RE | MISC | CRYPTO | SEC

Nepnep联合战队 | 米斯特SRC | 泽雷科技 | 零组文库



Nepnep联合战队

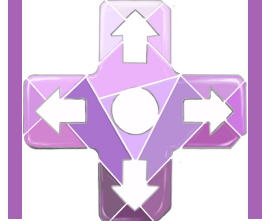
- 1 什么是 RSA
- 2 一点点数论基础
- 3 RSA 是如何加密和解密的
- 4 Python 简单实现 RSA 算法
- 5 RSA 相关的攻击算法

# 目录

1

# 什么是 RSA

Introduction to **RSA**

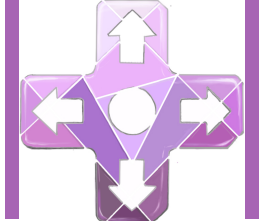


Nepnep联合战队

## 什么是 RSA

RSA是1977年由罗纳德·李维斯特 ( Ron Rivest )、阿迪·萨莫尔 ( Adi Shamir ) 和伦纳德·阿德曼 ( Leonard Adleman ) 一起提出的。当时他们三人都在麻省理工学院工作。RSA就是他们三人姓氏开头字母拼在一起组成的。



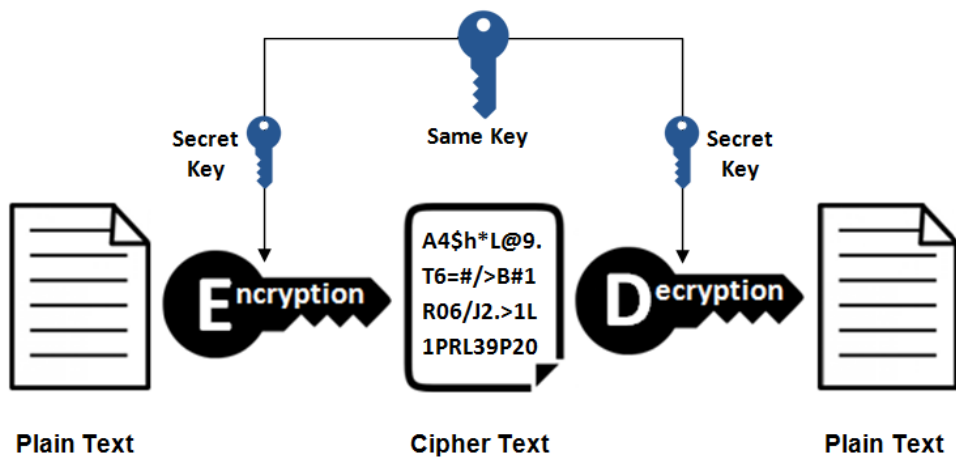


Nepnep联合战队

# 什么是 RSA

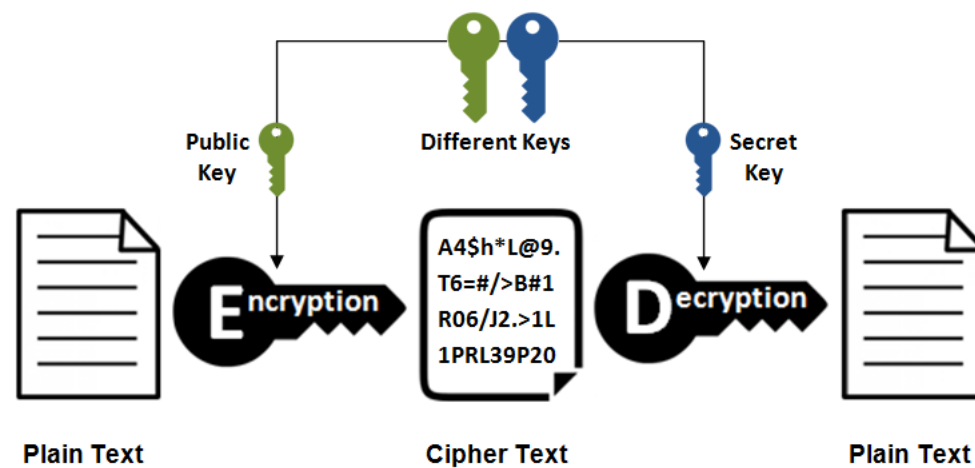
## 对称密码

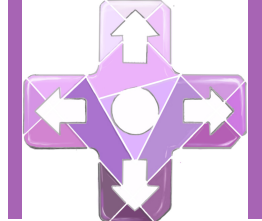
Symmetric Encryption



## 非对称密码

Asymmetric Encryption



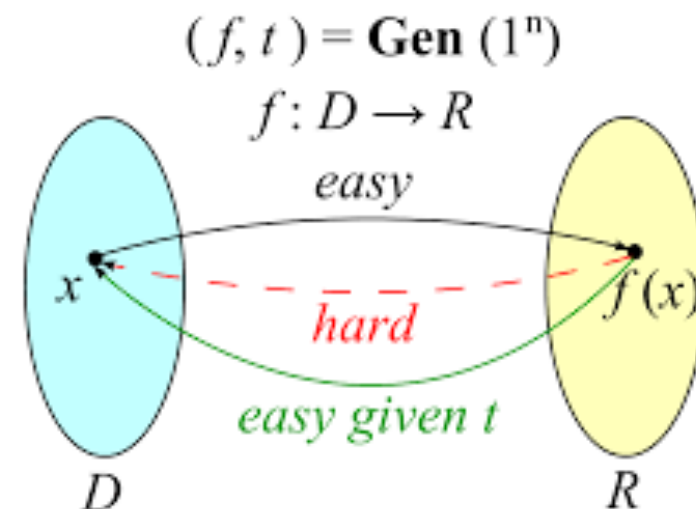


Nepnep联合战队

## 什么是 RSA

### 单向函数

- 对每一个输入  $x$  , 函数值  $f(x)$  都很容易计算
- 对随机给出的函数值  $f(x)$  , 算出原始输入  $x$  却比较困难
- 使用陷门信息 (*trapdoor information*) 则可以反逆

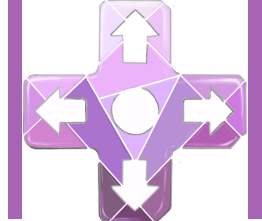


2

一点点数论基础

A very little

# Number Theory



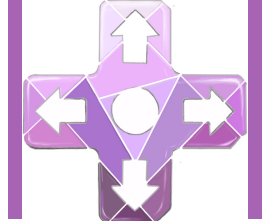
Nepnep联合战队

## 一点点数论基础

**同余：**

若  $a, b$  为两个整数，且它们的差  $a - b$  能被某个自然数  $m$  所整除，则称  $a$  就模  $m$  来说同余于  $b$ ，或者说  $a$  和  $b$  关于模  $m$  同余，记为： $a \equiv b \pmod{m}$ 。它意味着： $a - b = m * k$  ( $k$ 为某一个整数)。



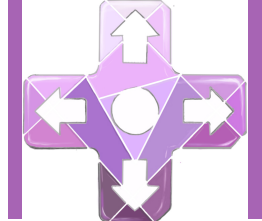


Nepnep联合战队

## 一点点数论基础

对于整数 $a, b, c$ 和自然数 $m, n$ ，对模 $m$ 同余具有以下一些性质：

1. 自反性： $a \equiv a \pmod{m}$
2. 对称性：若 $a \equiv b \pmod{m}$ ，则 $b \equiv a \pmod{m}$
3. 传递性：若 $a \equiv b \pmod{m}$ ， $b \equiv c \pmod{m}$ ，则 $a \equiv c \pmod{m}$
4. 同加性：若 $a \equiv b \pmod{m}$ ，则 $a + c \equiv b + c \pmod{m}$
5. 同乘性：若 $a \equiv b \pmod{m}$ ，则 $a * c \equiv b * c \pmod{m}$   
若 $a \equiv b \pmod{m}$ ， $c \equiv d \pmod{m}$ ，则 $a * c \equiv b * d \pmod{m}$
6. 同幂性：若 $a \equiv b \pmod{m}$ ，则 $a^n \equiv b^n \pmod{m}$
7. 推论1： $a * b \pmod{k} = (a \pmod{k}) * (b \pmod{k}) \pmod{k}$
8. 推论2：若 $a \pmod{p} = x$ ， $a \pmod{q} = x$ ， $p, q$  互质, 则  $a \pmod{p * q} = x$ 。



Nepnep联合战队

## 一点点数论基础

推论2 : 若  $a \bmod p = x$  ,  $a \bmod q = x$  ,  $p, q$  互质 , 则  $a \bmod p * q = x$ 。

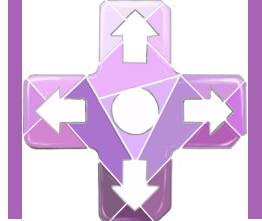
证明 : 因为  $a \bmod p = x$  ,  $a \bmod q = x$  ,  $p, q$  互质

则一定存在整数  $s, t$  , 使得  $a = s * p + x$  ,  $a = t * q + x$

所以 ,  $s * p = t * q$

则一定存在整数  $r$  , 使  $s = r * q$

所以 ,  $a = r * p * q + x$  , 得出 :  $a \bmod p * q = x$



Nepnep联合战队

## 一点点数论基础

### 模逆元：

模逆元也称为**模倒数**。

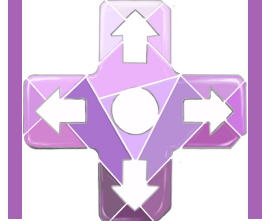
一整数  $a$  对同余  $n$  之模逆元是指满足以下公式的整数  $b$ ：

$$a^{-1} \equiv b \pmod{n}$$

也可以写成以下的式子：

$$ab \equiv 1 \pmod{n}$$

整数  $a$  对模数  $n$  之模逆元存在的充分必要条件是  $a$  和  $n$  互素，若此模逆元存在，在模数  $n$  下的除法可以用和对应模逆元的乘法来达成，此概念和实数除法的概念相同。



Nepnep联合战队

## 一点点数论基础

### 模逆元：

可以使用Python第三方包Crypto的 `inverse()` 函数求模逆元。

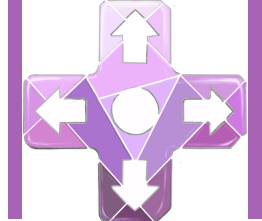
```
from Crypto.Util.number import inverse  
print(inverse(3, 7)) # 3 是要求逆元的数，7是模数
```

可以使用Python第三方包gmpy2的 `invert()` 函数求模逆元。

```
from gmpy2 import invert  
print(invert(3, 7)) # 3 是要求逆元的数，7是模数
```

可以在SageMath中直接用 `inverse_mod()` 函数求模逆元。

```
inverse_mod(3, 7) # 3 是要求逆元的数，7是模数
```



Nepnep联合战队

## 一点点数论基础

### 模运算：

#### 加法

$$3 + 4 \equiv 0 \pmod{7}, \quad 5 + 5 \equiv 3 \pmod{7}, \quad 1 + 5 \equiv 6 \pmod{7}$$

#### 减法

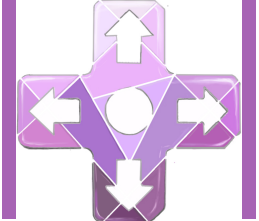
$$5 - 2 \equiv 3 \pmod{7}, \quad 3 - 5 \equiv 3 - 5 + 7 \equiv 3 + 2 \pmod{7}$$

#### 乘法

$$3 \times 4 \equiv 5 \pmod{7}, \quad 2 \times 2 \equiv 4 \pmod{7}$$

#### “除法”

$$“5 \div 4 \equiv 5 \times 4^{-1} \equiv 5 \times invert(4, 7) \equiv 5 \times 2 \equiv 3 \pmod{7}”$$



Nepnep联合战队

## 一点点数论基础

### 欧拉函数：

$\mathbb{Z}_m$  内与  $m$  互素的整数的个数可以表示为  $\Phi(m)$

示例：

假设  $m$  等于 6 时，现在对应的集合为  $\{0, 1, 2, 3, 4, 5\}$

$$\text{GCD}(0, 6) = 6$$

$$\text{GCD}(1, 6) = 1 \text{-----互素}$$

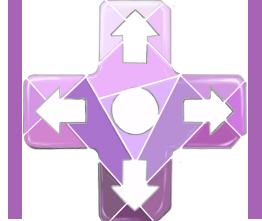
$$\text{GCD}(2, 6) = 2$$

$$\text{GCD}(3, 6) = 3$$

$$\text{GCD}(4, 6) = 2$$

$$\text{GCD}(5, 6) = 1 \text{-----互素}$$

由于该集合中，有两个与 6 互素的数字，即 1 和 5，所以欧拉函数的值为 2，即  $\Phi(6) = 2$ 。

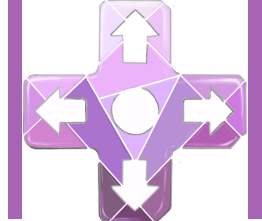


# 一点点数论基础

## 欧拉函数：

不难看出，如果数值非常大的话，将集合内的元素从头至尾都处理一遍并计算 gcd 的欧拉函数的计算方法会非常慢。实际上，使用这种最直接的方法计算公钥密码学中使用的非常大的整数对应的欧拉函数是非常困难的。幸运的是，如果  $m$  的因式分解是已知的，则存在一个更简单的计算方法，如下图所示。

- $\varphi(1) = 1$ 。
- $\varphi(N) = N \cdot \prod_{p|N} \left( \frac{p-1}{p} \right)$ 。
- $\varphi(p^k) = p^k - p^{k-1} = (p-1) \cdot p^{k-1}$ ，其中  $p$  为质数。
- $\varphi(mn) = \varphi(m) \cdot \varphi(n)$ ，其中  $\gcd(m, n) = 1$ 。



Nepnep联合战队

## 一点点数论基础

**欧拉函数：**

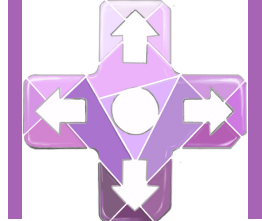
假设  $m = 240$ ，240 因式分解对应的素因数相乘形式为：

$$m = 240 = 16 \cdot 15 = 2^4 \cdot 3 \cdot 5$$

$$\Phi(m) = 2^3 \cdot (2 - 1) \cdot (3 - 1) \cdot (5 - 1) = 8 \cdot 1 \cdot 2 \cdot 4 = 64$$

所以想要计算出一个合数的欧拉函数，需要先知道这个数的因式分解，一些大整数的乘积难以被分解的特点也保证了 RSA 公钥加密的安全性。





Nepnep联合战队

## 一点点数论基础

### 费马小定理：

假如  $a$  为一个整数， $p$  为一个素数，则

$$a^{p-1} \equiv 1 \pmod{p}$$

### 欧拉定理：

假设  $a$  和  $m$  都是整数，且  $\gcd(a, m) = 1$  则有

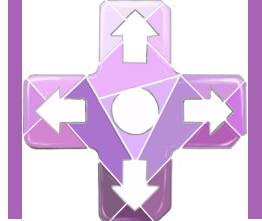
$$a^{\Phi(m)} \equiv 1 \pmod{m}$$

3

RSA 是如何加密和解密的

# Details

of RSA encryption algorithm

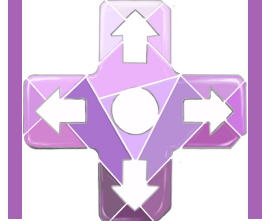


Nepnep联合战队

## RSA 是如何加密和解密的

RSA加密中会出现以下几个参数：

- 两个大的素数  $p$  和  $q$ ，以及它们的积  $n$ ， $n$  是加解密过程中的模数
- 欧拉函数  $\varphi(n)=(p-1)*(q-1)$
- 加密指数  $e$ ，和解密指数  $d = \text{invert}(e, \varphi(n))$
- 密文  $c$ ，明文  $m$



Nepnep联合战队

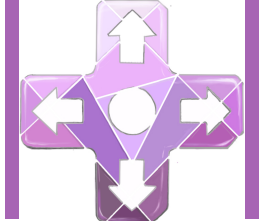
## RSA 是如何加密和解密的

**加密：**

$$c = m^e \bmod n$$

**解密：**

$$m = c^d \bmod n$$



Nepnep联合战队

## RSA 是如何加密和解密的

因为  $d$  是  $e$  对  $\varphi(n)$  的逆元，所以  $e$  和  $d$  在模  $\varphi(n)$  的运算下，互为倒数，所以

$$e \cdot d \equiv 1 \pmod{\Phi(n)}$$

所以存在正整数  $r$  满足：


$$e \cdot d = 1 + r \cdot \Phi(n)$$

$$\begin{aligned} c &= m^e \pmod{n} \\ c^d &\equiv m^{e \cdot d} \equiv m^{1+r \cdot \Phi(n)} \equiv m \cdot m^{r \cdot \Phi(n)} \equiv m \cdot (m^{\Phi(n)})^r \equiv m \cdot 1 \equiv m \pmod{n} \end{aligned}$$

$e \cdot d = 1 + r \cdot \Phi(n)$

$m^{\Phi(n)} \equiv 1 \pmod{n}$

$1^r = 1$

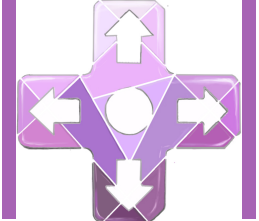


4

Python 简单实现 RSA 算法

# Implementation

of RSA encryption algorithm



Nepnep联合战队

## 简单实现 RSA 算法

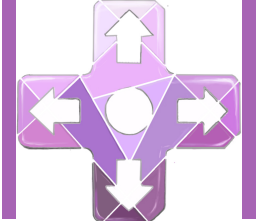
### 生成随机素数

getPrime()函数，括号里的参数意义为位长度，下面示例表示生成一个 512bits 的随机素数。

```
1  from Crypto.Util.number import *  
2  p = getPrime(512)
```

getStrongPrime() 函数，括号里的参数意义为位长度，生成一个更安全的素数。

```
1  from Crypto.Util.number import *  
2  p = getStrongPrime(512)
```



Nepnep联合战队

## 简单实现 RSA 算法

### 计算模逆元的两个函数的区别

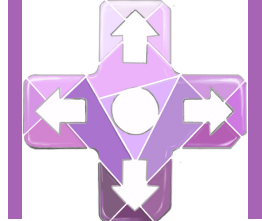
使用 Crypto 包里的 `inverse()` 函数，两个参数不互素的时候返回的是除以最大公因数之后的逆元。互素的情况下和 `gmpy2` 的 `invert` 返回值相同。

```
from Crypto.Util.number import *  
d = inverse(e, (p-1)*(q-1))
```

使用 `gmpy2` 包里的 `invert()` 函数，两个参数不满足互素时会报错，只有满足互素时正常求逆元。

```
from gmpy2 import invert  
d = invert(e, (p-1)*(q-1))
```





Nepnep联合战队

## 简单实现 RSA 算法

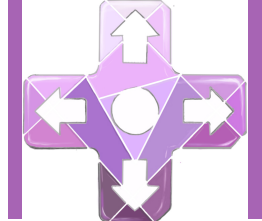
### 判断素数

*isPrime()* 可以用来判断素数

```
1  from Crypto.Util.number import *
2  print(isPrime(7))
```

### 求最大公因数

```
1  from Crypto.Util.number import *
2  print(GCD(12, 18)) # 6
```



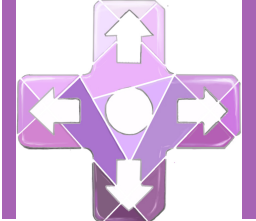
Nepnep联合战队

## 简单实现 RSA 算法

### 开 n 次方根

使用 gmpy2 的 `iroot` 函数，可以开  $n$  次方根，返回一个数字，一个布尔值。数字表示开根的结果，布尔值表示结果的  $n$  次方是否刚好等于原来的数。

```
1  from gmpy2 import iroot
2  print(iroot(4,2)) # 表示对 4 开平方根
```

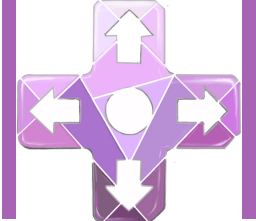


Nepnep联合战队

## 简单实现 RSA 算法

### RSA加密

```
1  from Crypto.Util.number import *
2
3  m = 123456
4  e = 65537
5  p, q = getPrime(128), getPrime(128)
6  n = p*q
7  c = pow(m, e, n)
8  print(c)
9  #46446567530734328956895050621451855413068614241783489657583370527982913123577
```

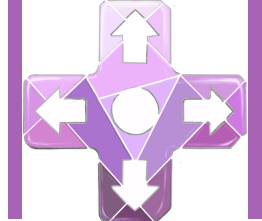


Nepnep联合战队

## 简单实现 RSA 算法

### RSA解密

```
1  from Crypto.Util.number import *
2
3  p = 307677955863441770267761398816442898269
4  q = 234956087764792853945937117492863651101
5
6  e = 65537
7  c = 25009787683260330186467878431892979552086452988450015957799934262727022882703
8  d = inverse(e, (p-1)*(q-1))
9  n = p*q
10 m = pow(c, d, n)
11 print(m)
12 # 123456
```



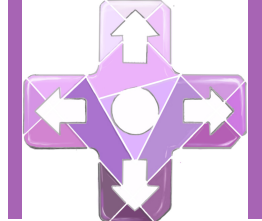
Nepnep联合战队

## 简单实现 RSA 算法

现在 Alice 想要接收 Bob 的一串数字 123456，他们的通信线路是不安全的，可以被攻击者 Eve 窃听到，所以他们可以使用 RSA公钥加密算法，使信息安全的传输。

Alice 首先需要生成公钥  $(e, n)$ ，和不发送的私钥  $d$ 。Alice 会将  $(e, n)$  发送给 Bob：

```
1  from Crypto.Util.number import *
2
3  e = 65537
4  p, q = getPrime(128), getPrime(128)
5  n = p*q
6  d = inverse(e, (p-1)*(q-1))
7  print("n =",n)
8  print("d =",d)
9  #n = 64119097861467025841314869723386401172380445670197869142261509855521615781313
10 #d = 34488339696882282190704342167311503848344429551595618555980250777974774475073
```

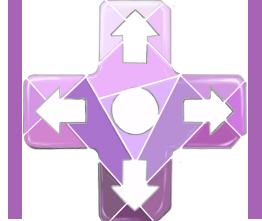


Nepnep联合战队

## 简单实现 RSA 算法

现在 Bob 接收到了 Alice 发送的公钥  $(e, n)$ ，他使用这组公钥加密自己的明文，并把密文  $c$  发送给 Alice：

```
1  from Crypto.Util.number import *
2
3  e = 65537
4  n = 64119097861467025841314869723386401172380445670197869142261509855521615781313
5  m = 123456
6  c = pow(m, e, n)
7  print("c =", c)
8  #c = 35175039627001706475239565374293351303180888840171603334474497768320397005778
```



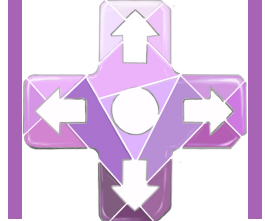
Nepnep联合战队

## 简单实现 RSA 算法

Alice 接收到了  $c$ ，她还有之前生成的解密指数  $d$ ，她可以用私钥  $(d, n)$  解开密文：

```
1  from Crypto.Util.number import *
2
3  d = 34488339696882282190704342167311503848344429551595618555980250777974774475073
4  n = 64119097861467025841314869723386401172380445670197869142261509855521615781313
5  c = 35175039627001706475239565374293351303180888840171603334474497768320397005778
6
7  m = pow(c, d, n)
8  print("m =", m)
9  #m = 123456
```

所以 Alice 最终得到了 Bob 的明文，并且他们在这条不安全的通信线路中传递了两次信息，这两次都被 Eve 成功窃听到。



Nepnep联合战队

## 简单实现 RSA 算法

那么 Eve 现在掌握的信息是：

- Alice 和 Bob 在使用 RSA 公钥加密算法传递信息。
- 窃听到了 Alice 发送的  $e$  和  $n$ 。
- 窃听到了 Bob 发送的  $c$ 。

Eve 想要窃取到明文  $m$ ，需要从这三个参数入手。

在很多 CTF 密码学题目中，我们解题就相当于 Eve 做的事情——攻击加密算法。

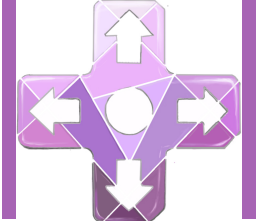


5

RSA 相关的攻击算法

# Attacks

on RSA cryptosystem



Nepnep联合战队

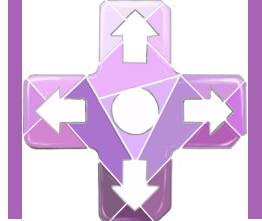
## RSA 相关的攻击算法

### 分解素因数攻击

以上面的 Alice 和 Bob 泄露的信息为例，开始第一种攻击，我们现在已知的参数：

```
e = 65537
n = 64119097861467025841314869723386401172380445670197869142261509855521615781313
c = 35175039627001706475239565374293351303180888840171603334474497768320397005778
```

解密需要计算  $\text{pow}(c, d, n)$ ，所以我们需要知道  $d$ ，然而  $d = \text{invert}(e, \varphi(n))$ ，这个式子中我们已知了  $e$  和  $n$ ， $\text{invert}$ 很好计算，就需要算  $\varphi(n)$ ，现在问题是如何计算  $n$  的欧拉函数，我们需要知道  $n$  的素因数分解。



Nepnep联合战队

## RSA 相关的攻击算法

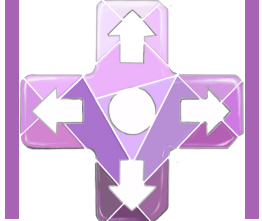
### 分解素因数攻击

$n$  的数值很大，RSA中常用的数量级往往不能通过枚举的方法（试除法）分解因数。但是如果生成的素数是不安全的，有可能导致  $n$  很容易被分解。

在这一个例子中，我们可以知道  $n$  是 256 位的，这种长度完全是不安全的（通常生成的素数约 2048 位），所以我们尝试使用一些算法或工具来尝试分解  $n$ 。

<https://www.alpertron.com.ar/ECM.HTM>

<http://www.factordb.com/index.php>



## RSA 相关的攻击算法

### 共模攻击

如果在 RSA 的使用中使用了相同的模  $n$  对相同的明文  $m$  进行了加密，那么就可以在不分解  $n$  的情况下还原出明文  $m$  的值。

$$m^{e_1} \equiv c_1 \pmod{n}, m^{e_2} \equiv c_2 \pmod{n}$$

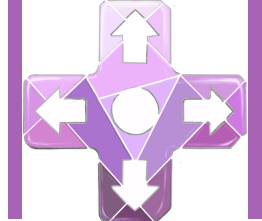
通过扩展欧几里德算法，可以计算出：

这样就有：  $c_1^r c_2^s \equiv m^{re_1+se_2} \equiv m^1 \pmod{n}$ .

但是  $r$  和  $s$  中必有一个是负数，所以需要逆元来处理一下(假设  $s < 0$ )：

$$m^1 \equiv m^{re_1+se_2} \equiv (m^{e_1})^r + (m^{e_2})^{-s^+} \equiv c_1^r (c_2^{-1})^{s^+} \pmod{n}$$

$s^+$  即  $|s| = -s$



Nepnep联合战队

## RSA 相关的攻击算法

### 已知 $p+q$ 或 $p-q$

或者是题目给了其他的pq之间的关系，通过解方程组或推导来求出p和q。

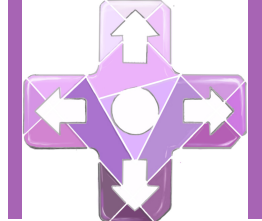
$$p * q = n$$

$$p + q = a$$

使用SageMath解方程组：

```
var('p q')
```

```
solve([p*q == n, p+q == a], [p, q])
```



Nepnep联合战队

## RSA 相关的攻击算法

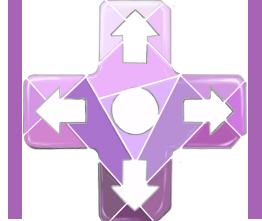
### 小公钥指数攻击

当加密指数  $e$  很小，比如  $e = 3$  时， $c$  可能不比  $n$  大很多（可能是  $n$  的几十倍或者几百倍，在可枚举的范围之内）。

这样就存在一个较小的可枚举的  $k$  满足：

$$m^3 = c + k \cdot n$$

尝试枚举  $k$  并开根，能刚好开根的就是解。



Nepnep联合战队

## RSA 相关的攻击算法

已知 $e$  ,  $d$  分解  $n$

$$ed \equiv 1 \pmod{\Phi(n)}$$

$$ed = 1 + k \cdot \Phi(n), k < e$$

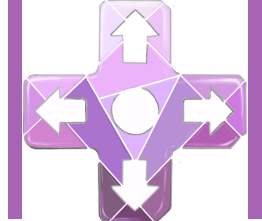
穷举 $k$  , 计算出 $\Phi(n)$

$$\Phi(n) = (p - 1)(q - 1) = n - (p + q) + 1$$

解一个二元二次方程组

$$p + q = n - \Phi(n) + 1$$

$$p \cdot q = n$$



Nepnep联合战队

## RSA 相关的攻击算法

### 已知明文高位攻击

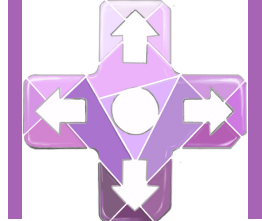
Coppersmith定理指出在一个  $e$  阶  $\text{mod } n$  的多项式  $f(x)$  中，如果有一个根小于  $n^{\frac{1}{e}}$ ，就可以运用一个  $O(\log n)$  的算法求出这些根。

所以在  $e$  等于 3，并且已知明文高位，可以尝试使用Coppersmith攻击。

```
PR.<x> =  
PolynomialRing(Zmod(n))  
f = (m0 + x) ^ 3 - c
```

```
ans = f.small_roots(X=2 ^ 500)  
m = ans[0]+m0
```





Nepnep联合战队

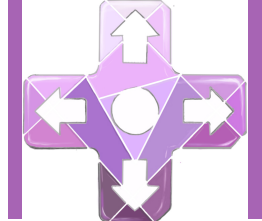
## RSA 相关的攻击算法

### Partial p 攻击

$PR.<x> = PolynomialRing(Zmod(n))$

$f = x + p0$

$roots = f.small\_roots(X=2^{256}, beta=0.3)$



Nepnep联合战队

## RSA 相关的攻击算法

### RSA Last Bit Oracle Attack

假设存在一个 Oracle，它会对一个给定的密文进行解密，并且会检查解密的明文的奇偶性，并根据奇偶性返回相应的值，比如 1 表示奇数，0 表示偶数。那么给定一个加密后的密文，我们只需要  $\log(N)$  次就可以知道这个密文对应的明文消息

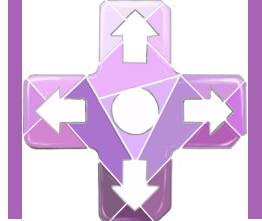
$$C = P^e \bmod N$$

$$C * 2^e = (2P)^e \bmod N$$

服务器会计算得到  $2P \bmod N$

$2P$  是偶数，它的幂次也是偶数。

$N$  是奇数，因为它是由两个大素数相乘得到。



## RSA 相关的攻击算法

### RSA Last Bit Oracle Attack

如果服务器返回奇数，即  $2P \bmod N$  为奇数，则说明  $2P$  大于  $N$ ，且减去了奇数个  $N$ ，又因为  $2P < 2N$ ，因此减去了一个  $N$ ，即  $\frac{N}{2} \leq P < N$ ，我们还可以向下取整。

服务器返回偶数，则说明  $2P$  小于  $N$ 。即  $0 \leq P < \frac{N}{2}$ ，我们还可以向下取整。

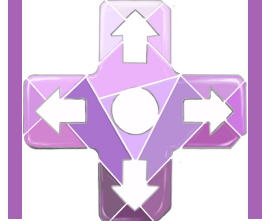
那么第  $i$  次时：

$$\frac{xN}{2^i} \leq P < \frac{xN+N}{2^i}$$

第  $i+1$  次，我们向服务器发送  $C * 2^{(i+1)e}$ ，服务器会计算得到  $2^{i+1}P \bmod N = 2^{i+1}P - kN$

$$0 \leq 2^{i+1}P - kN < N$$

$$\frac{kN}{2^{i+1}} \leq P < \frac{kN+N}{2^{i+1}}$$



Nepnep联合战队

## RSA 相关的攻击算法

### RSA Last Bit Oracle Attack

对于第  $i$  次：

$$\frac{2xN}{2^{i+1}} \leq P < \frac{2xN+2N}{2^{i+1}}$$

对于第  $i+1$  次：

$$\frac{kN}{2^{i+1}} \leq P < \frac{kN+N}{2^{i+1}}$$

如果返回1，那么  $k=2y+1$ ，带入得到：

$$\frac{2yN+N}{2^{i+1}} \leq P < \frac{2yN+2N}{2^{i+1}}$$

所以  $y=x$ ，更新下界。

如果返回0，那么  $k=2y$ ，带入得到：

$$\frac{2xN}{2^{i+1}} \leq P < \frac{2xN+N}{2^{i+1}}$$

所以  $y=x$ ，更新上界。



## Reference

- [1] 林厚从著. 信息学奥赛之数学一本通: 东南大学出版社, 2016
- [2] (美) Richard Spillman著. 叶阮健 曹英 张长富译. 经典密码学与现代密码学: 清华大学出版社, 2005
- [3] Jeffrey Hoffstein. Jill Pipher. Joseph H. Silverman. An Introduction to Mathematical Cryptography: Springer, 2000
- [4] RSA. In Wikipedia. [https://en.wikipedia.org/wiki/RSA\\_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
- [5] Select plaintext attack. In CTF Wiki. [https://ctf-wiki.github.io/ctf-wiki-en/crypto/asymmetric/rsa/rsa\\_chosen\\_plain\\_cipher/](https://ctf-wiki.github.io/ctf-wiki-en/crypto/asymmetric/rsa/rsa_chosen_plain_cipher/)

# 后记

Thanks a lot

涅普期待与你下次见面