《算法设计与分析》
RSA算法

殷合川
山东师范大学信息科学与工程学院

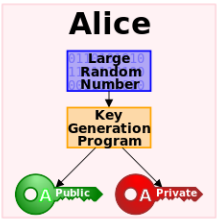
2014年12月

目录

概述—公钥密码学

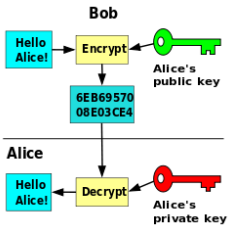
- Public-key cryptography, also known as asymmetric cryptography, is a class of cryptographic algorithms which requires two separate keys, one of which is secret (or private) and one of which is public. Although different, the two parts of this key pair are mathematically linked. The public key is used to encrypt plaintext or to verify a digital signature; whereas the private key is used to decrypt ciphertext or to create a digital signature. The term "asymmetric" stems from the use of different keys to perform these opposite functions, each the inverse of the other – as contrasted with conventional ("symmetric") cryptography which relies on the same key to perform both.

概述—公钥密码学



An unpredictable (typically large and random) number is used to begin generation of an acceptable pair of keys suitable for use by an asymmetric key algorithm

概述—公钥密码学



In an asymmetric key encryption scheme, anyone can encrypt messages using the public key, but only the holder of the paired private key can decrypt. Security depends on the secrecy of the private key

RSA概述

- RSA is one of the first practicable public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret. In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977.

RSA概述

- A user of RSA creates and then publishes a public key based on the two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message. Breaking RSA encryption is known as the RSA problem. It is an open question whether it is as hard as the factoring problem.

RSA概述

The RSA algorithm was publicly described in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT.
They won Turing Award in 2002 for their ingenious contribution to making public-key cryptography useful in practice.

目录

二进制加法算法的复杂度

- 两个n位的二进制数相加，其和最多有n+1位，最多需要n次按位的加法和n-1次因进位而导致的加法，因而其复杂度为$O(n)$
  - $53+35 = (32+16+4+1) + (32+2+1) = 110101_2 + 100011_2$
  - $63+63 = (64-1) + (64-1) = 111111_2 + 111111_2$

进位	1	0 0 1 1 1 0
		1 1 0 1 0 1
+		1 0 0 0 1 1
	1	0 1 1 0 0 0

进位	1	1 1 1 1 1 0
		1 1 1 1 1 1
+		1 1 1 1 1 1
	1	1 1 1 1 1 1

二进制乘法算法的复杂度

- 至多n-1次的左移1位操作，每次移位操作的复杂度为$\Theta(n)$，总复杂度为$O(n^2)$
- 至多n-1次的加法运算，每次加法运算至多对两个2n位数相加，总复杂度为$O(n^2)$
- 总之，二进制乘法的复杂度为$O(n^2)$

```
                1 1 0 1
              × 1 0 1 1
  ---------------------
                1 1 0 1   (1101 times 1)
              1 1 0 1     (1101 times 1, shifted once)
            0 0 0 0       (1101 times 0, shifted twice)
+ 1 1 0 1                 (1101 times 1, shifted thrice)
  ---------------------
1 0 0 0 1 1 1 1           (binary 143)
```


Al Khwarizmi乘法算法

- 算法表述见P17，举例：13×11
- 递归表示
  - $x \cdot y = \begin{cases} 2\left(x \cdot \left\lfloor \frac{y}{2} \right\rfloor\right) & \text{当}y\text{为偶数} \\ x + 2\left(x \cdot \left\lfloor \frac{y}{2} \right\rfloor\right) & \text{当}y\text{为奇数} \end{cases}$

11	13
5	26
~~2~~	~~52~~
1	104
	143

- 示例：$x = 13, y = 11$
  - $y_1 = 11$为奇数，$x \cdot y_1 = x + 2(x \cdot y_2), y_2 = \left\lfloor \frac{11}{2} \right\rfloor = 5$
  - $y_2 = 5$为奇数，$x \cdot y_2 = x + 2(x \cdot y_3), y_3 = \left\lfloor \frac{5}{2} \right\rfloor = 2$
  - $y_3 = 2$为偶数，$x \cdot y_3 = 2(x \cdot y_4), y_4 = \left\lfloor \frac{2}{2} \right\rfloor = 1$
  - $y_4 = 1$为奇数，$x \cdot y_4 = x + 2(x \cdot y_5), y_5 = \left\lfloor \frac{1}{2} \right\rfloor = 0$
  - $x \cdot y_1 = 13 + 2\left(13 + 2\left(2(13)\right)\right) = 13 + 26 + 104 = 143$

Al Khwarizmi乘法算法—伪代码

```
function multiply(x, y)
Input: Two n-bit integers x and y, where y ≥ 0
Output: Their product

if y = 0: return 0
z = multiply(x, ⌊y/2⌋)
if y is even:
   return 2z
else:
   return x + 2z
```

图1-1，P18

Al Khwarizmi乘法算法—复杂度

- 算法分析：
  - 对于n位二进制数x和y，算法在至多n次递归后结束，因为每次递归都使y折半，即使其二进制位数减少1位
  - 每次递归中包括一次除以2的操作和乘以2的操作，它们可用右移或左移1位来实现，复杂度均为$O(n)$，还可能有1次加法操作，复杂度也为$O(n)$
  - 因此总的复杂度为$O(n^2)$

二进制数的减法和除法运算的复杂度

- 由于采用补码表示后，二进制减法运算可以用加法来实现，因而其复杂度与加法运算相同，即$O(n)$
- 除法运算可以看成是一系列的减法运算，因而其复杂度与乘法运算相同，即$O(n^2)$

模运算与公钥机制(RSA)—目录

模运算

- x模N(x modulo N，记为x mod N)的结果定义为x除以N的余数。
  - 即如果$x = pN + r, 0 \le r < N$, 则x mod N=r
  - 如：3 mod 7 = 3, 10 mod 7 = 3, 11 mod 7 = 4
  - x可为任意整数，N为非0整数
  - 为了简化讨论，我们仅考虑x为0或正整数，N为正整数的情况
- 几乎所有的程序设计语言均提供了模运算
  - C/C++: %; Java/JavaScript: %; C#: %;
  - VB: Mod; MS Excel: =MOD()

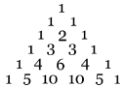
模运算的性质

- $(x + y) mod\ N = (x\ mod\ N + y\ mod\ N)\ mod\ N$
  - 令$x = pN + r, y = qN + s$
  - 则，$x + y = (p + q)N + r + s$
  - 因此，$(x + y) mod\ N = (r + s)\ mod\ N$
  - 而，$(x\ mod\ N + y\ mod\ N)\ mod\ N = (r + s)\ mod\ N$
- $(x - y) mod\ N = (x\ mod\ N - y\ mod\ N)\ mod\ N$
- $(x \cdot y) mod\ N = \big((x\ mod\ N) \cdot (y\ mod\ N)\big) mod\ N$

模运算的性质

- $x^y \bmod N = (x \bmod N)^y \bmod N$
  - 令$x = pN + r$
  - 由二项式定理(Blaise Pascal, 17th)

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$



  - $x^y = (pN + r)^y = \sum_{k=0}^{y} \binom{y}{k} (pN)^k r^{y-k}$
  - 显然，展开式中除去$k = 0$的项r^y外，其余各项均含有因子N
  - 因此，$x^y \bmod N = r^y \bmod N$
  - 而，$(x \bmod N)^y \bmod N = r^y \bmod N$

模运算的性质

- 结合律：
  - $((x + y) \bmod N + z) \bmod N = (x + (y + z) \bmod N) \bmod N$
  - $((x \cdot y) \bmod N \cdot z) \bmod N = (x \cdot (y \cdot z) \bmod N) \bmod N$
- 交换律：
  - $(x + y) \bmod N = (y + x) \bmod N$
  - $(x \cdot y) \bmod N = (y \cdot x) \bmod N$
- 分配率：
  - $(x(y + z) \bmod N) \bmod N = (xy \bmod N + xz \bmod N) \bmod N$

模同余

- 对于任意给定的N，任意整数经过模N运算后其结果均为0…N-1之间的整数值，因此可以说模N运算将所有的整数划分成了N个类
- 如果两个整数x和y关于N的模运算结果相同，则我们说x和y模N同余，记为
  - $x \equiv y(mod\ N)$
  - 如：$3 \equiv 10(mod\ 7), 10 \equiv 17(mod\ 7)$
- 显然，若x和y模N同余，其差一定可以被N整除
  - 因为$x = pN + r, y = qN + r \Rightarrow x - y = (p - q)N$
  - 即$x \equiv y(mod\ N) \Leftrightarrow N$整除$(x - y)$

模同余是一种等价关系

- 自反性
  - 显然：$x \equiv x\ (mod\ N)$
- 对称性
  - 显然：$x \equiv y(mod\ N) \Rightarrow y \equiv x(mod\ N)$
- 传递性
  - 若，$x \equiv y(mod\ N)$，$y \equiv z(mod\ N)$
  - 则有，$x - y = pN, y - z = qN$
  - 即，$x - z = (p + q)N$
  - 因而，$x \equiv z(mod\ N)$

模同余的性质，P20

- 如果$x \equiv x'(mod\ N)$，$y \equiv y'(mod\ N)$
  - 则：$x + y \equiv x' + y'(mod\ N)$
  - $\because x = pN + r, x' = p'N + r; y = qN + s, y' = q'N + s$
  - $\therefore x + y = (p + q)N + r + s, x' + y' = (p' + q')N + r + s$
  - $\therefore x + y \equiv r + s(mod\ N), x' + y' \equiv r + s(mod\ N)$
  - $\therefore x + y \equiv x' + y'(mod\ N)$

模同余的性质，P20

- 如果$x \equiv x'(mod\ N)$，$y \equiv y'(mod\ N)$
  - 则：$xy \equiv x'y'(mod\ N)$
  - $\because x = pN + r, x' = p'N + r; y = qN + s, y' = q'N + s$
  - $\therefore xy = (pq)N^2 + (ps + qr)N + rs, x'y' = (p'q')N^2 + (p's + q'r)N + rs$
  - $\therefore xy \equiv rs(mod\ N), x'y' \equiv rs(mod\ N)$
  - $\therefore xy \equiv x'y'(mod\ N)$

模同余的指数运算

- 如果$x \equiv x'(mod\ N)$， $y \equiv y'(mod\ N)$
  - 则：$xy \equiv x'y'(mod\ N)$
- 因此有： $x^2 \equiv (x')^2(mod\ N)$
- 同理： $x^n \equiv (x')^n(mod\ N)$
- 例：
  - $7 \equiv 2(mod\ 5)$
  - $7^2 = 49 = 9 \times 5 + 4, 2^2 = 4, \therefore 7^2 \equiv 2^2(mod\ 5)$
  - $7^3 = 343 = 68 \times 5 + 3, 2^3 = 8 = 1 \times 5 + 3,$ $\therefore 7^3 \equiv 2^3(mod\ 5).$

模同余的指数运算, P21

- 如果$x \equiv x'(mod\ N)$， 则$x^n \equiv (x')^n(mod\ N)$
- 例：
  - $6 \equiv 1(mod\ 5)$
  - $6^2 = 36 = 7 \times 5 + 1, \therefore 6^2 \equiv 1^2(mod\ 5) = 1(mod\ 5)$
  - $6^3 = 216 = 43 \times 5 + 1, \therefore 6^3 \equiv 1^3(mod\ 5) = 1(mod\ 5)$
  - $6^6 = 6^3 \times 6^3 = (43 \times 5 + 1)(43 \times 5 + 1) = p \times 5 + 1,$ $\therefore 6^6 \equiv 1^6(mod\ 5) = 1(mod\ 5).$
  - $2^{345} = (2^5)^{69} = (32)^{69}$
  - $32 \equiv 1(mod\ 31) \Rightarrow (32)^{69} \equiv 1^{69}(mod\ 31) = 1(mod\ 31)$

模加法、乘法和除法的算法复杂度

- 模加法的算法复杂度
  - $(x\ mod\ N + y\ mod\ N)\ mod\ N$首先对两个$0 \sim N-1$之间的数相加，如果结果大于$N-1$，则需要再执行一次减去$N$的减法运算，因此算法复杂度为$O(n)$，其中$n = \lceil logN \rceil$
- 模乘法的算法复杂度
  - $(x\ mod\ N \cdot y\ mod\ N)\ mod\ N$首先对两个$0 \sim N-1$之间的数相乘再除以$N$，因此算法复杂度为$O(n^2)$
- 模除法的算法复杂度
  - 模除法要较乘法复杂一些，算法复杂度为$O(n^3)$

模指数运算的算法

- 指数运算可以根据递归式$x^y =$

$$\begin{cases} \left(x^{\lfloor y/2 \rfloor}\right)^2 & y\text{为偶数} \\ x\left(x^{\lfloor y/2 \rfloor}\right)^2 & y\text{为奇数} \end{cases}$$

- function modexp(x, y, N)
  Input: Two n–bit integers x and N, an integer exponent y
  Output: $x^y \bmod N$

```
if y = 0: return 1
z = modexp(x, ⌊y/2⌋, N)
if y is even:
    return z^2 mod N
else:
    return x · z^2 mod N
```


模指数运算算法的复杂度

- 设n为x, y, N中最长的位数，则算法至多递归执行n次
- 每次递归调用中最多执行一次n位数的乘法
  - 注意，取模运算使每次的乘数限制在n位以内，如果不是取模运算，则每次的乘数可能会便得很大
- 因而算法复杂度为$O(n^3)$

- 注意这里的结论与之前计算x^n的log$n$算法并不矛盾
  - 因为前述计算a^N的算法，当$N = 2^n$时，复杂度变为log$N = n$

模运算与公钥机制(RSA)—目录

最大公约数

- 最大公约数(Greatest Common Divisor，简写为GCD)，指某几个整数共有约数中最大的一个
  - $48 = 2 \times 2 \times 2 \times 2 \times 3$
    $180 = 2 \times 2 \times 3 \times 3 \times 5.$
  - 它们的最大公约数为：$2 \times 2 \times 3 = 12$
- 求取GCD的蛮力算法是分别对两个整数进行因式分解，然后将公共因数相乘
  - 然而，整数因式分解是困难的，因而不可行

欧几里德最大公约数算法

- 欧几里德2000多年前在其《几何原本》第VII卷中提出的一个GCD算法，即欧几里德算法，是最古老而直到今天依然广泛使用的数值算法之一，该算法建立在如下的Euclid规则基础上
  - 如果x和y为正整数，且有$x \geq y$，则
    $\gcd(x, y) = \gcd(y, x \bmod y)$，或
    $\gcd(x, y) = \gcd(x - y, y)$
  - 证明参见P23-24

欧几里德最大公约数算法—伪代码

- 欧几里德GCD算法也是一个典型的演示递归程序设计的算法

```
function Euclid(a, b)
Input: Two integers a and b with a ≥ b ≥ 0
Output: gcd(a, b)

if b = 0: return a
return Euclid(b, a mod b)
```

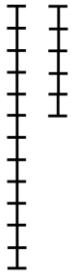
- 该算法的复杂度为$O(n^3)$，P24

欧几里德最大公约数算法—示例

- $252 = 21 \times 12$
  $105 = 21 \times 5.$

次数	a	b
1	252	105
2	105	42
3	42	21
4	21	0

- Euclid算法又被形象地称为辗转相除法



检验GCD的引理

- 假定d是a和b的GCD，怎样检验它呢？
  - 仅仅验证d能整除a和b是不够的
- 引理
  - d能整除a和b，同时又存在整数x和y，使$d = ax + by$，则一定有$d = \gcd(a, b)$
  - 该式称为Bézout等式，证明见P25
  - $\gcd(13,4) = 1, 1 = 13 \times 1 + 4 \times (-3)$

检验GCD的引理—演示

- $\gcd(252,105) = 21$
- Euclid算法演示

次数	a	b	$\lfloor a/b \rfloor$
1	252	105	2
2	105	42	2
3	42	21	2
4	21	0	

- Euclid算法回推

(3) $21 = 105 - 2 \times 42$
(2) $21 = 105 - 2 \times$
$(252 - 2 \times 105)$
$\therefore 21 = 252 \times (-2) + 105 \times 5.$
即，$x = -2, y = 5$

检验GCD的引理—演示

- gcd(180,48) = 12
- Euclid算法演示

次数	a	b	$\lfloor a/b \rfloor$
1	180	48	3
2	48	36	1
3	36	12	2
4	12	0	

- Euclid算法回推

(3) $12 = 48 - 36$
(2) $12 = 48 - (180 - 3 \times 48)$
$\therefore 12 = 180 \times (-1) + 48 \times 4.$
即，$x = -1, y = 4$

扩展欧几里德算法

- 引理
  - 对于任意的正整数a和b，利用下面的扩展Euclid算法可以求得整数x, y和d，使 $\gcd(a,b) = d = ax + by$
  - 该式称为Bézout等式，证明见P25-26
  - 显然，扩展Euclid算法与Euclid算法有相同的复杂度，即$O(n^3)$

```
function extended-Euclid(a, b)
Input: Two positive integers a and b with a ≥ b ≥ 0
Output: Integers x, y, d such that d = gcd(a, b) and ax + by = d

if b = 0: return (1, 0, a)
(x', y', d) = extended-Euclid(b, a mod b)
return (y', x' − ⌊a/b⌋y', d)
```


扩展欧几里德算法—演示

- extended_Euclid(252,105)返回-2, 5, 21
- $\gcd(252,105) = 21 = 252 \times (-2) + 105 \times 5$

次数	a	b	$\lfloor a/b \rfloor$	x'	y'	d
1	252	105	2	-2	5	21
2	105	42	2	1	-2	21
3	42	21	2	0	1	21
4	21	0		1	0	21

```
if b = 0: return (1, 0, a)
(x', y', d) = extended-Euclid(b, a mod b)
return (y', x' − ⌊a/b⌋y', d)
```


扩展欧几里德算法—演示

- extended_Euclid(25,11)返回4, -9, 1
- $\gcd(25,11) = 1 = 25 \times 4 + 11 \times (-9)$

P27,
gcd(25,11)
= 1
$= 25 \times 15$
$+11 \times (-34)$

次数	a	b	$\lfloor a/b \rfloor$	x'	y'	d
1	25	11	2	4	-9	1
2	11	3	3	-1	4	1
3	3	2	1	1	-1	1
4	2	1	2	0	1	1
5	1	0		1	0	1

$4 \bmod 11$
$\equiv 15 \bmod 11$

```
if b = 0: return (1, 0, a)
(x', y', d) = extended-Euclid(b, a mod b)
return (y', x' − ⌊a/b⌋y', d)
```

$(-9) \bmod 25$
$\equiv 16 \bmod 25$
$\equiv (-34) \bmod 25$

模运算与公钥机制(RSA)—目录

模的乘法逆元

- a模N的乘法逆元定义为：
  - 如果存在一个x满足：$ax = 1 (mod\ N)$
  - 则称x为a的逆元，记为：$a^{-1} \equiv x (mod\ N)$
- 它与普通乘法中的倒数有很相似的性质
  - $3x = 1$, 则$x = 3^{-1}$为3的乘法逆
  - 有了逆后，除法运算就可以转换为对逆的乘法运算
  - $5 \div 3 = 5 \times 3^{-1}$

模的乘法逆元

- 但是，a模N的乘法逆与普通乘法的逆有很大的不同：
  - 在普通乘法中，大于1的数n的逆n^{-1}是小于1的
  - 而a模N的乘法的逆a^{-1}的取值范围与a的取值范围相同，都是在$1 \sim N-1$之间
  - 与普通乘法相同的是，0元素没有逆
- 举例
  - 由于$3 \times 4 = 12 \equiv 1(mod\ 11)$
  - 则$3^{-1} \equiv 4(mod\ 11), 4^{-1} \equiv 3(mod\ 11)$

模的乘法逆元

- 并非任何的a都有模N的逆元
  - 如，当$a = 2, N = 6$时就不存在x，使$ax \equiv 1(mod\ 6)$，因为对于任何的整数x，ax都是偶数，因而对6进行模运算的结果不可能是奇数，也就不可能是1
- 如果a与N互素，即$\gcd(a, N) = 1$
  - 则一定存在a模N的逆元

模的乘法逆元

- 模的乘法逆元可以将除法运算转化为乘法运算，因而在数论和密码学中有广泛的应用，RSA算法就是运用模乘法逆元的典型例子
- 扩展的欧几里德算法给出了求模的乘法逆元的多项式方法，这可由模的除法定理表述，P27
  - 对于任意的$a \bmod N$，a有一个模N的乘法逆元，当且仅当a与N互素。如果逆元存在，则可以通过扩展的欧几里德算法在$O(n^3)$时间内求得

为什么说扩展Euclid算法能够求模的乘法逆元？

- 扩展欧几里德算法在求得a与N的最大公约数d的同时，还能求得Bézout等式中的x和y，即
  - $\gcd(a, N) = d = ax + Ny$
  - 则由于当a与N互素时，$d = 1 = ax + Ny$
  - 我们有，$ax = -Ny + 1$
    即，$ax \bmod N = 1$，或$ax \equiv 1(mod\ N)$
    显然x就是a模N的逆元
  - 同理，由于$Ny = -ax + 1$
    即，$Ny \bmod a = 1$，或$Ny \equiv 1(mod\ a)$
    显然y就是N模a的逆元

模运算与公钥机制(RSA)—目录

费马小定理(Fermat's little theorem)

- 如果p是一个素数，那么对于任意的$1 \leq a < p$，有
  - $a^{p-1} \equiv 1(mod\ p)$
- 举例，取$p = 5$
  - $a = 1, 1^4 = 1 \equiv 1(mod\ 5)$
  - $a = 2, 2^4 = 16 = 3 \times 5 + 1 \equiv 1(mod\ 5)$
  - $a = 3, 3^4 = 81 = 16 \times 5 + 1 \equiv 1(mod\ 5)$
  - $a = 4, 4^4 = 256 = 51 \times 5 + 1 \equiv 1(mod\ 5)$
- 费马小定理提供了一种素性测试方法

费马大定理

- 费马大定理来自于费马猜想(Fermat's conjecture, 1637年)
  - 不定方程$a^n + b^n = c^n$(n为整数)在$n > 2$时只有平凡的整数解
  - 即，当n为偶数时，只有形如$(0, \pm m, \pm m)$或$(\pm m, 0, \pm m)$的整数解，如(0, -3, 3), (3, 0, -3)等
  - 当n为奇数时，只有形如$(0, m, m)$或$(m, 0, m)$的整数解，如(0, -3, -3), (3, 0, 3)等
- 当$n = 2$时，该方程就是著名的勾股定理
  - $a^2 + b^2 = c^2$
  - 业已证明，有无穷多的整数3元组满足勾股定理
  - 如：(3, 4, 5), (5, 12, 13)等

费马大定理

- 费马猜想提出后，直到300多年后，才由英国数学家安德鲁 怀尔斯（Andrew John Wiles）及其学生理查 泰勒（Richard Taylor）于1995年完全证明
  - 猜想证明后，被称为“费马最后的定理”(Fermat's Last Theorem)，也称为“费马大定理”
  - 在冲击这个数论世纪难题的过程中，无论是不完全的还是最后完整的证明，都给数学界带来很大的影响；
  - 很多的数学结果、甚至数学分支在这个过程中诞生了，包括代数几何中的椭圆曲线和模形式，以及伽罗瓦理论和赫克代数等。
  - 安德鲁 怀尔斯由于成功证明此定理，获得了2005年度邵逸夫奖。

皮埃尔 德 费马

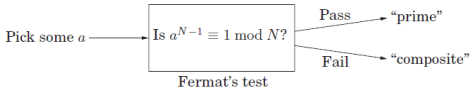
- 费马是法国律师和业余数学家
  - 被誉为“业余数学家之王”
- 然而他在数学上的成就却是大师级的
  - 无穷小量微积分的先驱者
  - 在数论领域有重大贡献
  - 在解析几何、概率论、光学等领域都有重要成就
  - 以费马大定理而闻名
  - 名言：“我发现了一个美妙的证明，但由于空白太小而写不下。”

Pierre de Fermat
1601.8.17—1665.1.12

模运算与公钥机制(RSA)—目录

基于费马小定理的素数测试

- 费马小定理提供了一种可以不用因子分解而测试素数的方法
  - 对于给定的N，选取任意的$1 \le a < p$，判断$a^{p-1} \equiv 1 (mod\ p)$是否成立，如果成立，则$N$为素数，如果不成立，则$N$为合数
  - 由于模的指数运算可以在$O(n^3)$时间内完成，基于费马小定理的素数测试是算法可行的



Fermat's test

基于费马小定理的素数测试

- 基于费马小定理可以写出如下所示的素数测试算法伪代码
  - 该算法的复杂度就是模指数运算的复杂度，即$O(n^3)$

```
function primality(N)
Input: Positive integer N
Output: yes/no

Pick a positive integer a < N at random
if a^(N-1) ≡ 1 (mod N):
   return yes
else:
   return no
```

基于费马小定理的素数测试

- 然而，费马小定理不是测试素数的充分必要条件
- 因为存在通过该测试的合数
- 如：341=11×31，而$2^{340} \equiv 1(mod\ 341)$
- 计算：$2^{340} = \left((2^{17})^5\right)^4$

  $2^{17}\ mod\ 341 = 128, 128^5\ mod\ 341 = 32$

  $32^4\ mod\ 341 = 1$
- 但是：3^{340}并不与1模341同余
- 计算：$3^{340} = \left((3^{17})^5\right)^4$

  $3^{17}\ mod\ 341 = 53, 53^5\ mod\ 341 = 254$

  $254^4\ mod\ 341 = 56$

基于费马小定理的素数测试—改进

- 事实上，存在一类及其罕见的合数N，Carmichael数，对所有与N互素的a，N都将通过费马测试
  - 最小的Carmichael数：$561 = 3 \times 11 \times 17$
  - $5^{560} \equiv 1\ mod\ 561$

    $5^{560}\ mod\ 561 = (5^7\ mod\ 561)^{80}\ mod\ 561$

    $= (146^5 \bmod 561)^{16} \bmod 561 = (23^4 \bmod 561)^4 \bmod 561$

    $= 463^4\ mod\ 561 = 1$
  - $97^{560} \equiv 1\ mod\ 561$

    $97^{560}\ mod\ 561 = (97^7\ mod\ 561)^{80}\ mod\ 561$

    $= (466^5 \bmod 561)^{16} \bmod 561$

    $= (232^4 \bmod 561)^4 \bmod 561$

    $= 463^4\ mod\ 561 = 1$

基于费马小定理的素数测试—改进

- 事实上，存在一类及其罕见的合数N，Carmichael数，对所有与N互素的a，N都将通过费马测试
  - 最小的Carmichael数：$561 = 3 \times 11 \times 17$
  - 在不考虑Carmichael数的情况下(Rabin-Miller测试可以排除Carmichael数)，以下引理可改进费马测试:
  - 如果对于某些与N互素的a，有a^{N-1}不与1模N同余，那么对于$a < N$的至少一半的可能取值，N将无法通过费马测试
  - 证明，P30-31

基于费马小定理的素数测试—改进

- 在忽略Carmichael数的情况下，我们有如下结论
  - 如果N是素数，那么对于所有的$a < N$，$a^{N-1} \equiv 1(mod\ N)$成立
  - 如果N不是素数，那么对于所有的$a < N$，至多有一半满足$a^{N-1} \equiv 1(mod\ N)$
- 因而前述算法primality具有以下的概率表现
  - Pr(当N为素数时，$primality$算法返回yes) $= 1$
  - Pr(当N为合数时，$primality$算法返回yes) $\leq \frac{1}{2}$

基于费马小定理的素数测试—改进

- 将算法primality重复执行k次，可以使通过测试的合数降低到$1/2^k$的概率水平
  - 改进后的算法如下所示，其失误概率为
  - Pr(当N为合数时，$primality2$算法返回yes) $\leq \frac{1}{2^k}$

```
function primality2(N)
Input: Positive integer N
Output: yes/no

Pick positive integers a_1, a_2, ..., a_k < N at random
if a_i^{N-1} ≡ 1 (mod N) for all i = 1, 2, ..., k:
   return yes
else:
   return no
```


素数的随机生成

- 前述的费马测试基本解决了判断单个整数素性的问题。在此基础上，我们可以构造一个随机的素数生成快速算法。
- 根据素数基本定理，一个随机的n位长的数字为素数的概论约为$1.44/n$，即素数足够多
- 由此，可以构造随机生成n位长素数的方法
  1. 随机选定一个n位长的数N
  2. 对N进行素性测试
  3. 如果通过测试则输出N，否则转1

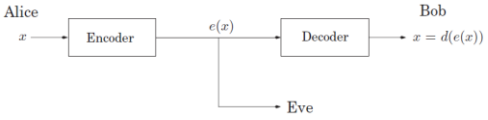
PRIMES is in P

- The AKS primality test is a deterministic primality-proving algorithm created and published by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, computer scientists at the Indian Institute of Technology Kanpur, on August 6, 2002, in a paper titled "PRIMES is in P". The algorithm determines whether a number is prime or composite within polynomial time, $\tilde{O}(\log^{12} n)$. The authors received the 2006 Gödel Prize and the 2006

模运算与公钥机制(RSA)—目录

密码学应用的基本思路

- 主流的密码学建立在算法公开，密钥保密的共识之上
- 密码机制分为对称密钥和非对称密钥两大类
  - 在对称密钥机制中，加密和解密使用相同的密钥，通常算法也是相同的
  - 典型的对称密钥算法有AES，3DES，IDEA等



公钥密码机制

- 公钥密码机制是一种非对称密钥机制，其加密密钥和解密密钥是不相同的，但算法是公开的
  - 任何人都可以持有两个密钥，一个加密密钥和一个解密密钥，这两个密钥是相对的，即可以用任何一个加密，用另一个解密
  - 他可以公开其中的一个密钥，称为公钥，不公开的密钥称为私钥
  - 给他发消息的人可以用其公钥加密，但用公钥加密的数据只能用对应的私钥才能解开，即只有他本人才能解密用其公钥加密的数据
  - 他发给别人的消息可以用私钥加密，其他人可以用其公钥解密，这一方面保证了信息的保真，另一方面，实现了不可抵赖

RSA的基本原理

- 随机选取两个不同的素数p和q，并令$N = pq$。对任意的与$(p-1)(q-1)$互素的整数e，有以下数论性质：
  1. 映射$x \mapsto x^e \bmod N$是定义在集合$\{0, 1, \dots, N-1\}$上的双向映射。
  2. 容易得到该映射的逆映射：令d为$e \bmod (p-1)(q-1)$的逆，则对于所有的$x \in \{0, 1, \dots, N-1\}$有，
     $$(x^e)^d \equiv x \bmod N$$
- 显然性质1说明可以使用$x \mapsto x^e \bmod N$进行加密操作，而性质2保证了可以解密

RSA的基本原理

- 下面证明$(x^e)^d \equiv x \bmod N$
  - 由于e和d模$(p-1)(q-1)$互逆，
  - $ed = 1 \bmod (p-1)(q-1)$
  - 即存在整数k，使$ed = (p-1)(q-1)k + 1$
  - 即$x^{ed} - x = x(x^{(p-1)(q-1)k} - 1)$
  - 根据费马小定理，$x^{p-1} \equiv 1 \bmod p$
  - 则有$\left(x^{(p-1)}\right)^{(q-1)k} = (1 \bmod p)^{(q-1)k} = 1 \bmod p$
  - 因而$x^{(p-1)(q-1)k} - 1$能够被p整除
  - 同理可证$x^{(p-1)(q-1)k} - 1$能够被q整除
  - 因此$x^{(p-1)(q-1)k} - 1$可以被$N = pq$整除
  - 因此$x^{ed} - x$可以被N整除，即$(x^e)^d \equiv x \bmod N$

RSA是怎样不可攻破的？

- 使用RSA的人将d作为私钥保存，而公开e和N
  - 加密者用$x^e \ mod \ N = y$将信息x加密为y
  - 解密这用$y^d \ mod \ N = (x^e)^d \ mod \ N = x$进行解密
  - 攻击者获得的信息是e, N和y
  - 攻击者要想获得d，就必须获得$(p-1)(q-1)$，即要获得p和q，而这需要对大整数N进行因子分解，而这是算法上不可行的

RSA的运行过程为什么是算法上可行的？

1. 获取大素数p, q可以在多项式时间内完成，如前述的费马素性测试；
2. 进行大整数乘法$N = pq$和$(p-1)(q-1)$可以在多项式时间内$(O(n^2))$完成；
3. 找一个与$(p-1)(q-1)$互素的整数e是多项式时间的，如Euclid GCD算法；
4. 找e的模$(p-1)(q-1)$的逆d可以用扩展的Euclid GCD算法在多项式时间内完成。
5. 计算$x^e \ mod \ N$、$y^d \ mod \ N$属于指数的模N运算，这也是多项式的。

RSA示例

- 令$p = 5, q = 11$，则$N = pq = 55$，$N' = (p-1)(q-1) = 40$
- 取$e = 3$，显然$\gcd(3,40) = 1$
- 由于$3 \times 27 = 81$，因而$d = 3^{-1} \ mod \ 40 = 27$
- 设$x = 13$，则$y = 13^3 \ mod \ 55 = 52$
- 而$52^{27} \ mod \ 55 = 13$

模运算与公钥机制(RSA)—目录

The End
Thanks