蒙哥马利模乘简介

高级安全研究部 陈庆

关键词：大数模乘、蒙哥马利优化、模逆元

摘要：RSA、DH 密钥交换等算法都涉及大数模幂，一般通过大数模乘完成。1985 年数学家蒙哥马利 (Peter L. Montgomery) 提出一种与 CPU 强相关的优化算法用于计算 REDC(T)=T*R' mod N (N>1)，该算法避免了除以 N 的操作，称之为蒙哥马利约分。技巧性地组合使用蒙哥马利约分来实现大数模乘，可以规避大开销的除法、求模运算，一般称此时的大数模乘为 " 蒙哥马利模乘 "。本文未做严格数学推导，只是从程序员以及逆向分析人员的实用角度简介之。

一、求模运算

在初等代数里有一种很重要的运算，求模。用一般人能理解的表述方式，就是求余数。比如，9 除以 7 余 2，也可以说 9 模 7 等于 2，写作：

9 mod 7 = 2

还可以写作：

9 ≡ 2(mod 7)

假设都是整数，如果 a=k*n+b 对某些 k 成立，那么 a ≡ b(mod n)，a mod n = b。

(a + b) mod n = ((a mod n) + (b mod n)) mod n

(a - b) mod n = ((a mod n) - (b mod n)) mod n

(a * b) mod n = ((a mod n) * (b mod n)) mod n

(a * (b + c)) mod n = (((a * b) mod n) + ((a * c) mod n)) mod n

这里不做严格的数学定义及推导，只是大概介绍一下模运算。

逆元 (inverse)，也叫倒数，比如 4 的逆元是 1/4，因为 4*(1/4)=1。在模运算领域也有类似的概念。对于两个正整数 a、n，若存在正整数 b，满足：

(a * b) mod n = 1

或

a*b ≡ 1(mod n)

此时 b 称做 a 的模 n 逆元，也叫数论倒数。求模逆元即求解方程 (a*x) mod n=1，

也写作：

a^(-1) ≡ x(mod n)

a*x ≡ 1(mod n)

等价于求解二元一次不定方程：

ax-ny=1

比如：

5*3-14*1=1

5^(-1) ≡ 3(mod 14)

即 5 的模 14 逆元是 3。

如果模逆元存在，必然有无数个，若：

a*b ≡ 1(mod n)

必有：

a*(b+k*n) ≡ 1(mod n)

但是，模逆元并不总是存在，比如：

2^(-1) ≡ x(mod 14)

无解。

若两个正整数 a、n 互素，则 a 的模 n 逆元必然存在，可以用欧拉定理证明：

a^φ(n)=a*a^(φ(n)-1) ≡ 1(mod n)

此时 (a^(φ(n)-1) mod n) 就是 a 的模 n 逆元。比如，gcd(3,11)=1，有：

3^(φ(11)-1)=3^(10-1)=3^9=19683

19683 mod 11 = 4

3*4 ≡ 1(mod 11)

若 gcd(a,n)>1，a 的模 n 逆元不存在。当模逆元存在时，可以用 " 扩展欧几里得算法 " 求解。

二、32-bits 整数版本的蒙哥马利模乘

所谓模乘运算，指形如 "x*y mod N" 的运算，即两数相乘再求模。而模幂运算是模乘的升级，即求 x 的 y 次幂再求模。稍微了解点公开密钥体系的人都知道，RSA、Diffie-Hellman(DH 密钥交换) 等算法会涉及大数模幂，通常可以通过加法链算法将模幂运算转换成若干模乘运算。

模幂可以转换成模乘，模乘中开销最大的操作是求模，求模实际就是除法，一次除法实际包含了多次加法、减法和乘法。如果算法中能尽量减少、避免除法，则效率大大提高。

求模运算在计算机领域中相当普遍，除了 RSA、DH，其实更常见的是 x86 汇编中的 32-bits 整数回绕，这个问题的本质是 "mod 0x100000000"。可以说，你的计算机中天天进行着求模运算。

1985 年数学家蒙哥马利 (Peter L. Montgomery) 提出一种与 CPU 强相关的优化算法用于计算 REDC(T)=T*R' mod N (N>1)，该算法避免了除以 N 的操作，称之为蒙哥马利约分。技巧性地组合使用蒙哥马利约分来实现大数模乘，可以规避大开销的除法、求模运算，一般称此时的大数模乘为 " 蒙哥马利模乘 "。

设 N>1 是整数，选一个 R>N，满足 gcd(R,N)=1，此时必存在整数 R'、N' 满足：

R*R' ≡ 1(mod N)

R*R'-N*N'=1 ≡ 1(mod R)

-N*N' ≡ 1(mod R)

(R-N)*N' ≡ 1(mod R)

0<R'<N

0<N'<R

即 R' 是 R 的模 N 逆元, N' 是 -N(不是 N) 的模 R 逆元。

定义蒙哥马利约分 (Montgomery Reduction):

x*R' mod N

设 0<=T<R*N，有 :

T=T*R*R'-T*N*N'

T+T*N'*N=T*R*R'

令 m=T*N'，有 :

T+m*N=T*R*R'

(T+m*N)/R=T*R'

这表示对于任意 0<=T<R*N，存在整数 m，使得 (T+m*N)/R 没有余数。用如下算法计算蒙哥马利约分 :

REDC(T)=T*R' mod N

```
int REDC ( int T )
{
   int m, t;
   /*
    * R*R'-N*N'=1
    */
   m   = ( ( T % R ) * N' ) % R;
   t   = ( T + m * N ) / R;
   if ( t >= N )
   {
      t  -= N;
   }
   return( t );
}
```

T*R' mod N=T*R*R'/R mod N=T(N'*N+1)/R mod N=(T*N'*N+T)/R mod N

注意到对于任意整数 k，有 :

((T * N ' + k * R) * N + T) / R mod N=(T*N'*N+T)/R mod N

从而在计算 "T*R' mod N" 的过程中，不必计算 m=T*N'，可以计算 "m=T*N' mod R"。对于 0<=T<R*N，t=(T+m*N)/R 小于 2*N，计算 "t mod N" 时，可以用一个判断及减法完成。

仔细观察 REDC(T)，若 R=2^i (i>1)，则 REDC(T) 中的除法、求模都是廉价的位操作。这个算法就是蒙哥马利约分算法 (Montgomery Reduction Algorithm)。

定义蒙哥马利剩余 (Montgomery Residue):

x'=x*R mod N

欲求 :

z=x*y mod N

有 :

z'=z*R mod N=x*y*R mod N=(x'*y')*R' mod N=REDC(x'*y')

z = z ' * R ' mod N=REDC(z')=REDC(REDC(x'*y'))

上式表明为求得 z，只需要几次 REDC()，当 R 为 2 的幂时 REDC() 很廉价。我们仍需要计算 "x'=x*R mod N"，这个可以优化 :

x'=x*R mod N=(x*R*R)*R' mod N=REDC(x*(R*R mod N))

新问题出现，为了计算 x'，需要计算 "R*R mod N"。由于 R、N 是常量，只需预计算一次 "R*R mod N" 即可。

现在我们有 :

x'=REDC(x*(R*R mod N))

y'=REDC(y*(R*R mod N))

z'=REDC(x'*y')

z=REDC(z')

z=x*y mod N=REDC(REDC(REDC(x*(R*R mod N))*REDC(y*(R*R mod N))))

这个算法就是蒙哥马利模乘算法。

三、32-bits 整数版本的蒙哥马利模乘示例

已知：

x=6

y=10

r=2

R=16=2^4 (比 N 大的最小 r 幂)

N=11

gcd(16,11)=1

求：

z=x*y mod N=6*10 mod 11=5

计算：

R'=9

N'=(16*9-1)/11=13

R*R mod N=256 mod 11=3

x'=REDC(x*(R*R mod N))=REDC(6*3)=REDC(18)=(18+m*11)/16

m=((T%R)*N')%R=((18%16)*13)%16=(2*13)%16=10

x'=(18+10*11)/16=8

y'=REDC(y*(R*R mod N))=REDC(10*3)=REDC(30)=(30+m*11)/16

m=((T%R)*N')%R=((30%16)*13)%16=(14*13)%16=6

y'=(30+6*11)/16=6

z'=REDC(x'*y')=REDC(8*6)=REDC(48)=(48+m*11)/16

m=((T%R)*N')%R=((48%16)*13)%16=0

z'=(48+0*11)/16=3

z=REDC(z')=REDC(3)=(3+m*11)/16

m=((T%R)*N')%R=((3%16)*13)%16=(3*13)%16=7

z=(3+7*11)/16=5

虽然当 R 为 2 的幂时 REDC() 很廉价，但算法本身并没有要求 R 为 2 的幂，下例 R=100。

已知：

x=17

y=26

N=79

r=10

R=100=10^2 (比 N 大的最小 r 幂)

gcd(100,79)=1

求：

z=x*y mod N=17*26 mod 79=47

计算：

R'=64

N'=(100*64-1)/79=81

R*R mod N=100*100 mod 79=46

x'=REDC(x*(R*R mod N))=REDC(17*46)=REDC(782)=(782+m*79)/100

m=((T%R)*N')%R=((782%100)*81)%100=(82*81)%100=6642%100=42

x'=(782+42*79)/100=4100/100=41

y'=REDC(y*(R*R mod N))=REDC(26*46)=REDC(1196)=(1196+m*79)/100

m=((T%R)*N')%R=((1196%100)*81)%100=(96*81)%100=7776%100=76

y'=(1196+76*79)/100=7200/100=72

z'=REDC(x'*y')=REDC(41*72)=REDC(2952)=(2952+m*79)/100

m=((T%R)*N')%R=((2952%100)*81)%100=(52*81)%100=4212%100=12

z'=(2952+12*79)/100=3900/100=39

z=REDC(z')=REDC(39)=(39+m*79)/100

m=((T%R)*N')%R=((39%100)*81)%100=(39*81)%100=3159%100=59

z=(39+59*79)/100=4700/100=47

R 为 100 时，对于 CPU 来说 REDC() 并不廉价，但口算时除以 100 或模 100 会很容易，相当于规避了除法、求模。这正是 REDC() 乃至蒙哥马利模乘算法的精髓所在。上两例 REDC() 的中间结果 t 都小于 N，因此没有减 N 操作。

四、大数版的蒙哥马利模乘

设：

gcd(r,N)=1

R=r^m

R'=r^(-m)

N=sum[i=0 to m-1](N[i]*r^i),0<=N[i]<r,N<R

N[0]=N mod r

X=sum[i=0 to m-1](X[i]*r^i),0<=X[i]<r,X<N

X[0]=X mod r

Y=sum[i=0 to m-1](Y[i]*r^i),0<=Y[i]<r,Y<N

Y[0]=Y mod r

S=X*Y*R' mod N,S<N

S=sum[i=0 to m-1](S[i]*r^i),0<=S[i]<r,S<N

S[0]=S mod r

用如下算法计算 S:

S=Montgomery(X,Y)=X*Y*R' mod N

```
Montgomery ( X, Y )
{
   S  = 0;
```

```
/*
 * (r-N[0])'*(r-N[0]) ≡ 1(mod r)
 */
p   = ( r - N[0] )'
for ( i = 0; i < m; i++ )
{
   q   = ( ( S[0] + X[i] * Y[0] ) * p ) mod r;
   S   = S + X[i] * Y + q * N;
   S   = S / r;
   if ( S >= N )
   {
      S   = S - N;
      if ( S >= N )
      {
         S   = S - N;
      }
   }
}
   return( S );
}
```

仔细观察 Montgomery(X,Y)，若 r=2^i (i>1)，则 Montgomery(X,Y) 中的除法、求模都是廉价的位操作。这个算法就是大数版的蒙哥马利约分算法。

要点在于寻找一个 q，使得：

(S + X[i] * Y + q * N) mod r = 0

令：

q = ((S + X[i] * Y) * N') mod r

有：

(q * N) mod r = ((S + X[i] * Y) * (R * R' - 1)) mod r

(S + X[i] * Y + q * N) mod r = ((S + X[i] * Y) * R * R') mod r = 0

化简 q:

q = ((S + X[i] * Y) * N') mod r = ((S[0] + X[i] * Y[0]) * N') mod r

= ((S[0] + X[i] * Y[0]) * (r - N[0])') mod r

定义：

X'=X*R mod N

欲求：

Z=X*Y mod N

有：

Z'=Z*R mod N=X*Y*R mod N=X'*Y'*R' mod N=Montgomery(X',Y')

Z=Z'*R' mod N= Montgomery(Z',1)=Montgomery(Montgomery(X',Y'),1)

上式表明为求得 Z，只需要几次 Montgomery()，当 r 为 2 的幂时 Montgomery() 很廉价。我们仍需要计算 "X'=X*R mod N"，这个

可以优化：

X'=X*R mod N=X*(R*R)*R' mod N=Montgomery(X,(R*R mod N))

新问题出现，为了计算 X'，需要计算 "R*R mod N"。由于 R、N 是常量，只需预计算一次 "R*R mod N" 即可。

现在我们有：

X'=Montgomery(X,(R*R mod N))

Y'=Montgomery(Y,(R*R mod N))

Z'=Montgomery(X',Y')

Z=Montgomery(Z',1)

Z=X*Y mod N=Montgomery(Montgomery(Montgomery(X,(R*R mod N)),Montgomery(Y,(R*R mod N))),1)

这个算法就是大数版的蒙哥马利模乘算法。

五、大数版的蒙哥马利模乘示例

已知：

X=17

Y=26

N=79

r=10

m=2

R=100=r^m (比 N 大的最小 r 幂)

gcd(10,79)=1

R'=64

求：

Z=X*Y mod N=17*26 mod 79=47

计算：

p=(r-N[0])'=(10-9)'=1'=1

R*R mod N=100*100 mod 79=46

X'=Montgomery(X,(R*R mod N))=Montgomery(17,46)=17*46*64 mod 79=41

S=0

S[0]=0

q=(0+7*6)*1 mod 10=2

S=0+7*46+2*79=480

S=480/10=48

S[0]=8

q=(8+1*6)*1 mod 10=4

S=48+1*46+4*79=410

S=410/10=41

Y'=Montgomery(Y,(R*R mod N))=Montgomery(26,46)=26*46*64 mod 79=72

S=0

S[0]=0

q=(0+6*6)*1 mod 10=6

S=0+6*46+6*79=750

S=750/10=75

S[0]=5

q=(5+2*6)*1 mod 10=7

S=75+2*46+7*79=720

S=720/10=72

Z'=Montgomery(X',Y')=Montgomery(41,72)=41*72*64 mod 79=39

S=0

S[0]=0

q=(0+1*2)*1 mod 10=2

S=0+1*72+2*79=230

S=230/10=23

S[0]=3

q=(3+4*2)*1 mod 10=1

S=23+4*72+1*79=390

S=390/10=39

Z=Montgomery(Z',1)=Z=Montgomery(39,1)=39*1*64 mod 79=47

S=0

S[0]=0

q=(0+9*1)*1 mod 10=9

S=0+9*1+9*79=720

S=720/10=72

S[0]=2

q=(2+3*1)*1 mod 10=5

S=72+3*1+5*79=470

S=470/10=47

r 等于 10 时，对于 CPU 来说 Montgomery() 并不廉价，但口算时除以 10 或模 10 会很容易，相当于规避了除法、求模。这正是 Montgomery() 乃至蒙哥马利模乘算法的精髓所在。上例 Montgomery() 的中间结果 S 都小于 N，因此没有减 N 操作。

六、小结

蒙哥马利模乘的关键在于蒙哥马利约分，后者将 div N、mod N 转换成 div R、mod R、div r、mod r；当 R、r 是 2 的幂时，除法、求模都是廉价的位操作。

蒙哥马利模乘要求预计算 "R*R mod N"，大数版本额外要求预计算 (r-N[0])'。

七、参考资源

Montgomery modular multiplication

https://en.wikipedia.org/wiki/Montgomery_modular_multiplication

Modular Multiplication Without Trial Division - Peter L. Montgomery [1985-04]

http://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777282-X/