SM2软件实现

山东大学网络空间安全学院

SM2

- SM2算法结构
- 基本优化思路
- 大数运算
- 固定点点乘
- 非固定点点乘
- 双倍点
- 协议实现

SM2算法结构

- 大数运算 （加 减 乘 模逆）
  - 256比特多精度运算，一般基于64比特/32比特无符号数组
  - p n两个素域
- 椭圆曲线 kP（固定点G 非固定点P）
  - DOUBLE
  - ADD
- 协议 签名、验签、加解密、密钥协商

椭圆曲线方程：$y^2 = x^3 + ax + b$

曲线参数：

```
p = FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFF
a = FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFC
b = 28E9FA9E 9D9F5E34 4D5A9E4B CF6509A7 F39789F5 15AB8F92 DDBCBD41 4D940E93
n = FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF 7203DF6B 21C6052B 53BBF409 39D54123
xG = 32C4AE2C 1F198119 5F990446 6A39C994 8FE30BBF F2660BE1 715A4589 334C74C7
yG = BC3736A2 F4F6779C 59BDCEE3 6B692153 D0A9877C C62A4740 02DF32E5 2139F0A0
```


SM2算法结构

- kP= P+P+.....+P 重复k次
- 二进制方法

Algorithm 13. (Left-to-right) binary method for point multiplication

INPUT: $k = (k_{m-1}, \ldots, k_1, k_0)_2$, $P \in E(\mathbb{F}_p)$.
OUTPUT: kP.

1. $Q \leftarrow \mathcal{O}$.
2. For i from $m-1$ downto 0 do
   2.1 $Q \leftarrow 2Q$.
   2.2 If $k_i = 1$ then $Q \leftarrow Q + P$.
3. Return(Q).

- 2个椭圆曲线点运算
  - 2Q
  - Q+P

SM2算法结构

- Point DOUBLE
  - Q=2P
  - 计算复杂度： 4M+4S + 10A
- Point ADD
  - Q=P+Q
  - 计算复杂度： 12M + 4S + 7A
- 对于Z=1的情况进一步减少计算复杂度
  - $PD_{Z=1}$ 2M+4S+7A
  - $PA_{Z2=1}$ 8M +3S +7A
- 加法的数量不能忽略

Point Doubling

Input: (X, Y, Z)
$S = 4XY^2$; $M = 3X^2 + aZ^4$
$X' = M^2 - 2S$
$Y' = M(S - X') - 8Y^4$
$Z' = 2YZ$
Output: (X', Y', Z')

Point Addition

Input: (X_1, Y_1, Z_1), (X_2, Y_2, Z_2)
$U_1 = X_1Z_2^2$; $U_2 = X_2Z_1^2$; $S_1 = Y_1Z_2^3$; $S_2 = Y_2Z_1^3$
if ($U_1 == U_2$) then
  if ($S_1 != S_2$) return POINT_AT_INFINITY
  else return POINT_DOUBLE (X_1, Y_1, Z_1)
  abort
end
$H = U_2 - U_1$; $R = S_2 - S_1$
$X_3 = R^2 - H^3 - 2U_1H^2$
$Y_3 = R(U_1H^2 - X_3) - S_1H^3$
$Z_3 = HZ_1Z_2$
Output: (X3, Y3, Z3)

基本优化思路

- 大数运算：基础，乘法和乘方是关键
- ECC坐标变换：
  - 标准为仿射坐标系（X, Y）：$Y^{2=} X^3+aX+b$
  - jacobian坐标系（x, y, z）： $y^2=x^3+ax.Z^4+bz^6$
    - $X = x/z^2$, $Y= y/z^3$
  - jacobian坐标系计算复杂度低
- ECC中固定点点乘kG：预计算表
  - 运算中G为固定点，可预计算nG并存储为表
- 非固定点点乘kP：预计算nP
  - 在线运算，计算代价较大，需要权衡预计算量。
- 协议层：复杂运算的组合

大数运算

- 256比特多精度运算 d=a+b
- 64比特无符号数组表示 a[4] b[4] d[4] 小端存储
  - c|d[0]= a[0]+b[0] ;//其中c为进位标志
  - c|d[1]= a[1]+b[1]+c;
  - c|d[2]= a[2]+b[2]+c;
  - c|d[3]= a[3]+b[3]+c;
- 结果为 c|d[3]|d[2]|d[1]|[d[0], < 2P （257位）
- 如果结果>=p， 需要减p, 完成模约
  - c|d[0]= d[0]-p[0];//其中c为借位标志
  - c|d[1]= d[1]-p[1]-c
  - c|d[2]= d[2]-p[2]-c
  - c|d[3]= d[3]-p[3]-c

大数运算

- 几个问题：
  - 带进位的加法/带借位的减法如何实现？
  - 如何判断结果是否>=p
  - 模约安全么？

- 带进位的加法/带借位的减法如何实现？
  - X86-64 内部指令

```
#include <intrin.h>
unsigned char carry = 0;
for (int i = 0; i < 4; i++) {
  carry = _addcarry_u64(carry, a[i], b[i], &d[i]);
} //adcq
unsigned char carry = 0;
for (int i = 0; i < 4; i++) {
carry = _subborrow_u64(carry, d[i], p[i], &d[i]);
} //sbbq
```


- 计算依赖标志寄存器，同时结果会影响标志寄存器
- 汇编实现的话效率稍高
- subq a,b #0.25
- sbbq c,d #1
- sbbq e,f
- sbbq g,h

ADD SUB	r,r/i	1	1	p0156	1	0.25
ADD SUB	r,m	1	2	p0156 p23		0.5
ADD SUB	m,r/i	2	4	2p0156 2p237 p4	5	1
ADC SBB	r,r/i	1	1	p06	1	1
ADC SBB	r,m	2	2	p06 p23		1
ADC SBB	m,r/i	4	6	3p0156 2p237 p4	5	2

- 汇编实现加法可利用bmi2 adcx adox优化
- adcx 只影响carry标志 ， adox只影响overflow标志
- 可同时执行两组多精度加法，内存加法效率较高
- xor z,z ;//清除标识位
- adcx a,b; adox p0(%rsp),j;
- adcx c,d; adox p0+8(%rsp),l;
- adcx e,f; adox p0+16(%rsp),n;
- adcx g,h; adox p0+24(%rsp),p;

ADCX ADOX	r,r	1	1	p06	1	1
ADC SBB	r,m	2	2	p06 p23		1
ADC SBB	m,r/i	4	6	3p0156 2p237 p4	5	2

- 如何判断结果是否>=p
- 模约安全么？
- 条件转移指令实现安全高效的模约
  - 将计算结果a,b,c,d复制到e,f,g,h寄存器
  - 将进位标志复制到一个寄存器g,然后执行多精度减法-p
  - 如果结果有借位，e,f,g,h为正确结果，否则a,b,c,d为正确结果
  - 利用条件转移指令cmovx将正确结果复制到结果寄存器

大数运算-模约

- setc g ;//1
- movq a,e; movq b, f; movq c,g; movq d,h; //1
- subq p[0],a; //1
- sbbq p[1],b; //1
- sbbq p[2],c; //1
- sbbq p[3],d; //1
- sbbq $0, g; //1
- cmovc e,a; cmovc f,b; cmovc g,c; cmovc h,d; //2

- 常量时间实现

大数运算-乘法

- 乘法相对复杂，利用BMI2 优化
- MULX ADOX ADCX SHLX SHRX
- mulx a,o0,o1
  - 其中一个乘数 固定为rdx，另一个乘数a可以为内存或寄存器，输出到两个寄存器o0,o1中,不影响标志寄存器
  - lo=mulx_u64(uint64_t a, uint64_t b, uint64_t * hi);
  - hi|lo= a*b;

MULX	r32,r32,r32	3	3	p1 2p056	4	1
MULX	r32,r32,m32	3	4	p1 2p056 p23		1
MULX	r64,r64,r64	2	2	p1 p5	4	1
MULX	r64,r64,m64	2	3	p1 p6 p23		1

大数运算-乘法

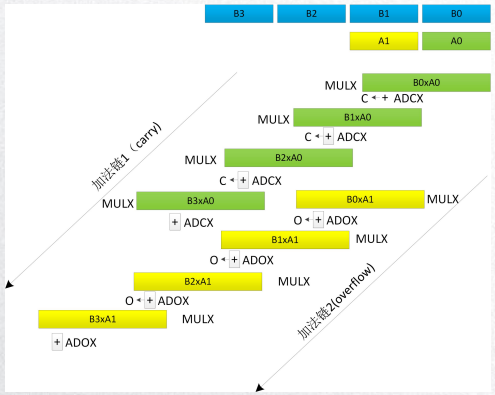
- SHLX/SHRX 不影响标记寄存器
  - shlx sft, a,b ;// b= (a<<<sft)
  - shrx sft, a,b ;// b= (a>>>sft)
- 用在小数乘法 2a,3a....
- 蒙哥马利模约

SHLX SHRX SARX	r,r,r	1	1	p06	1	0.5
SHLX SHRX SARX	r,m,r	2	2	p06 p23		0.5

大数运算-乘法

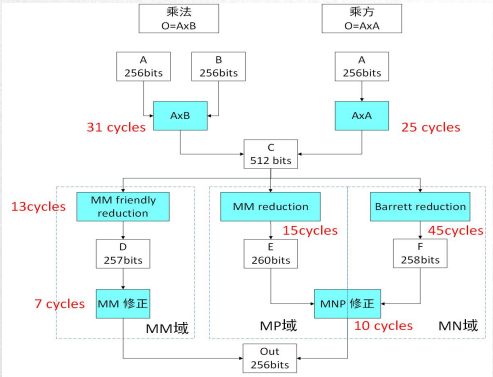
- X86-64架构下schoolbook 足够好
- A0x[B3..B0]
- A1x[B3..B0]
- A2x[B3..B0]
- A3x[B3..B0]

- 结果不超过512比特
- 乘方时间约为乘法80%



大数运算-乘法

- 如何将512比特约简到256比特素域内？
- 三种情况
  - 蒙哥马利快速模约（p）
  - 巴洛特模约（n, p）
  - 蒙哥马利域模约
    - 蒙哥马利友好(p)
    - 一般情况(n, p)
    - SM2曲线优化(sm2)



大数运算-乘法

- 蒙哥马利快速模约（p）
- p bin : 2^256 - 2^224 - 2^96 + 2^64 - 1

+ : 01 0000000000000000 0000000000000000 0000000000000001 0000000000000000

- : 0000000100000000 0000000000000000 0000000100000000 0000000000000001

- 按32bits分组
- 将模约转换为加法和减法
- 最多减去13P

0 :	a0	a1	a2	a3	a4	a5	a6	a7	+
1 :	a8	a9	0	a8	a9	a10	a11	a8	+
2 :	a9	a10	0	a11	a12	a13	a14	a9	+
3 :	a10	a11	0	a12	a13	a14	a15	a10	+
4 :	a11	a12	0	a13	a14	a15	0	a11	+
5 :	a12	a13	0	a13	a14	a15	0	a12	+
6 :	a13	a14	0	a14	a15	0	0	a12	+
7 :	a13	a14	0	a15	0	0	0	a13	+
8 :	a14	a15	0	0	0	0	0	a13	+
9 :	a14	a15	0	0	0	0	0	a14	+
0 :	a15	0	0	0	0	0	0	a14	+
1 :	a15	0	0	0	0	0	0	a15	+
2 :	0	0	0	0	0	0	0	a15	+
3 :	0	0	0	0	0	0	0	a15	+
0 :	0	0	a8	0	0	0	0	0	-
1 :	0	0	a9	0	0	0	0	0	-
2 :	0	0	a13	0	0	0	0	0	-
3 :	0	0	a14	0	0	0	0	0	-

大数运算-乘法

- 蒙哥马利快速模约（sm2）
- 首先处理高256比特
- 低256比特执行多精度加法

0 :	a0	a1	a2	a3	a4	a5	a6	a7	+
1 :	a8	a9	0	a8	a9	a10	a11	a8	+
2 :	a9	a10	0	a11	a12	a13	a14	a9	+
3 :	a10	a11	0	a12	a13	a14	a15	a10	+
4 :	a11	a12	0	a13	a14	a15	0	a11	+
5 :	a12	a13	0	a13	a14	a15	0	a12	+
6 :	a13	a14	0	a14	a15	0	0	a12	+
7 :	a13	a14	0	a15	0	0	0	a13	+
8 :	a14	a15	0	0	0	0	0	a13	+
9 :	a14	a15	0	0	0	0	0	a14	+
0 :	a15	0	0	0	0	0	0	a14	+
1 :	a15	0	0	0	0	0	0	a15	+
2 :	0	0	0	0	0	0	0	a15	+
3 :	0	0	0	0	0	0	0	a15	+
0 :	0	0	a8	0	0	0	0	0	-
1 :	0	0	a9	0	0	0	0	0	-
2 :	0	0	a13	0	0	0	0	0	-
3 :	0	0	a14	0	0	0	0	0	-

```
movl T15d, T14d ;  SHRX Sft32, T15, T15   ;      movq T14, tmp0;;\
movl T9d, T8d ;  SHRX Sft32, T9, T9       ;    movq T8, T16 ;\
movl T13d, T12d ;  SHRX Sft32, T13, T13   ;       ;\
movl T11d ,T10d ;  SHRX Sft32, T11, T11   ; movq T13, tmp1;;\
adox T15, T14;       ;   adcx T9, T16;   ;\
adox T14, T13;   ;     adcx T10, T16;;\
adox T13, T12;   ;     adcx T11, T16;  ;\
adox T15,T10;      ;\
adox T13,T10;   adcx tmp0, T9;;\
adcx tmp1, T8;  adox T11,T14;   ;\
adox T12 ,T11;  adcx T12, T16;    ;\
adcx T12, T15;  adox T8 ,T11;     \
 adox T9 , T12;       adcx T16, T15;\
 adcx T13, T16;      adox T8 ,T9; \
SHLX  Sft32, T16, T13;\
adcx  T16, T13; adox    posC(%rsp), T13;\
SHRX Sft32, T16,tmp0;\
SHLX Sft32, T11, X00;\
SHRX Sft32, T11, tmp1;\
adcx X00,tmp0;  adox    (posC+8)(%rsp), tmp0;\
SHLX Sft32, T10, X00;\
SHRX Sft32, T10, T16;\
adcx tmp1, X00; adcx T12,X00   ;\
adox (posC+16)(%rsp), X00    ;\
SHLX Sft32,  T15, T12;\
SHRX Sft32, T15, T11;\
adcx T12, T16;\
adcx T14,T16 ;  adox (posC+24)(%rsp), T16 ;\
 movq $0,Sft32;\
adcx Sft32,T11   ;  adox Sft32,T11    ;\
movl T8d, T15d;\
SALQ  $32,T15;\
SHRQ  $32,T8   ;\
addq T9, T8    ;\
setc T9b;\
movzx T9b, T12;\
subq    T15, T13 ;\
sbbq    T8, tmp0   ;\
sbbq    T12, X00   ;\
sbbq    $0, T16    ;\
sbbq    $0, T11  ;\
```


大数运算-乘法

- 巴洛特模约 通用，性能较差
- 需要执行 2次 256比特乘法

Algorithm 2.14 Barrett reduction

INPUT: p, $b \geq 3$, $k = \lfloor \log_b p \rfloor + 1$, $0 \leq z < b^{2k}$, and $\mu = \lfloor b^{2k}/p \rfloor$.
OUTPUT: $z \bmod p$.

1. $\widehat{q} \leftarrow \lfloor \lfloor z/b^{k-1} \rfloor \cdot \mu / b^{k+1} \rfloor$.
2. $r \leftarrow (z \bmod b^{k+1}) - (\widehat{q} \cdot p \bmod b^{k+1})$.
3. If $r < 0$ then $r \leftarrow r + b^{k+1}$.
4. While $r \geq p$ do: $r \leftarrow r - p$.
5. Return(r).

refs: guide to elliptic curve cryptography

大数运算-乘法

- 需要减去多次p时如何处理？
- while(a>p) a=a-p;
-
- 构造一个预计算表，只保存低256比特
- np[0]：0　　0
- np[1]：p　　0
- np[2]：2p　　1
- np[3]：3p　　2
- np[4]：4p　　3
- 如计算结果为 c|b3|b2|b1|b0
- b= b- np[c] ， 如 有借位 + p,否则 +0

大数运算-乘法

- 预计算表 index 0 存储全0
- index1 存储p
- 只需要根据借位标志设置序号即可
- const time实现

```
shlq $5, Sft32 ;\
xorq SIN,SIN;\
movq $32, X00;\
leaq    SM2nXN_p64(%rip), T9  ;\
subq ( 32)(T9 , Sft32), T13 ;\
sbbq ( 40)(T9 , Sft32), DIN ;\
sbbq ( 48)(T9 , Sft32), tmp1 ;\
sbbq ( 56)(T9 , Sft32), T16;\
cmovc X00, SIN;\
movq rspParS(%rsp), T8;\
addq  (T9 , SIN),T13;\
adcq ( 8)(T9 ,SIN),DIN ;\
adcq ( 16)(T9 ,SIN),tmp1;\
adcq ( 24)(T9 ,SIN),T16 ; \
movq T13, (T8);\
movq DIN, 8(T8);\
movq tmp1, 16(T8);\
movq T16, 24(T8)
```


大数运算-乘法

- 蒙哥马利乘法 推荐算法
- 运算规则
- $c= axbx2^{-256}$

  - k0 64比特常数
  - k=4
  - T3/2^s直接丢弃低64位
  - 结果不超过2p
  - SM2 p为MM友好素数

Algorithm 1: Word-by-Word Montgomery Multiplication (WW-MM)

Input: $p < 2^l$ (odd modulus), $0 \le a, b < p$, $l=s\times k$

Output: $a\times b\times 2^{-l} \bmod p$

Pre-computed: $k0 = -p^{-1} \bmod 2^s$

Flow

1. T = a×b
2. For i = 1 to k do
3. T1 = T mod 2^s
4. Y = T1 × k0 mod 2^s
5. T2 = Y × p
6. T3 = (T + T2)
7. T = T3 / 2^s
8. End For
9. If T ≥ p then X = T − p; else X = T

Return X

Algorithm 2: Word-by-Word Montgomery Multiplication for a Montgomery Friendly modulus p (Montgomery Friendly Multiplication)

Satisfying $-p^{-1} \bmod 2^s=1$.

Input: $p < 2^l$ (Montgomery Friendly modulus), $0 \le a, b < p$, $l=s\times k$

Output: $a\times b\times 2^{-l} \bmod p$

Flow

1. T = a×b
2. For i = 1 to k do
3. T1 = T mod 2^s
4. T2 = T1 × p
5. T3 = (T + T1)
6. T = T3 / 2^s
7. End For
8. If T ≥ p then X = T − p; else X = T

Return X

Fast Prime Field Elliptic Curve Cryptography with 256 Bit Primes @2013

大数运算-乘法

- SM2优化
- p bin : 2^256 - 2^224 - 2^96 + 2^64 - 1

+ : 01 0000000000000000 0000000000000000 0000000000000001 0000000000000000

- :　0000000100000000 0000000000000000 0000000100000000 0000000000000001

- SM2 步骤3 可退化为 加法 和减法
- 可用mulx adcx adox 构造多流水线

Algorithm 2: Word-by-Word Montgomery Multiplication for a Montgomery Friendly modulus p (Montgomery Friendly Multiplication)

Satisfying $-p^{-1} \bmod 2^s=1$.

Input: $p < 2^l$ (Montgomery Friendly modulus), $0 \le a, b < p$, $l=s\times k$

Output: $a\times b\times 2^{-l} \bmod p$

Flow

1. T = a×b
2. For i = 1 to k do
3. T1 = T mod 2^s
4. T2 = T1 × p
5. T3 = (T + T1)
6. T = T3 / 2^s
7. End For
8. If T ≥ p then X = T − p; else X = T

Return X

大数运算-乘法

- 运算规则
- c= axbx2^{-256}
- 转入MM域
  - 将 a-> ax2^{256}
  - 将 b-> ax2^{256}
    - a=MULP(a, 2^{256})
    - b= MULMM(a, 2^{512})
  - MULMM(ax2^{256}, bx2^{256}) = ax2^{256}xbx2^{256}x2^{-256} =
- 转出MM域
  - MULP(ax2^{256}, 2^{-256})
  - MULMM(ax2^{256}, 1)

Algorithm 1: Word-by-Word Montgomery Multiplication (WW-MM)

Input: $p < 2^l$ (odd modulus),
$0 \le a, b < p$, $l=s\times k$
Output: $a\times b\times 2^{-l} \bmod p$
Pre-computed: $k0 = -p^{-1} \bmod 2^s$
Flow
1. T = a×b
For i = 1 to k do
2. T1 = T mod 2^s
3. Y = T1 × k0 mod 2^s
4. T2 = Y × p
5. T3 = (T + T2)
6. T = T3 / 2^s
End For
7. If T ≥ p then X = T − p;
else X = T
Return X

Algorithm 2: Word-by-Word Montgomery Multiplication for a Montgomery Friendly modulus p (Montgomery Friendly Multiplication)

Satisfying $-p^{-1} \bmod 2^s=1$.
Input: $p < 2^l$
(Montgomery Friendly modulus)
$0 \le a, b < p$, $l=s\times k$
Output: $a\times b\times 2^{-l} \bmod p$
Flow
1. T = a×b
For i = 1 to k do
2. T1 = T mod 2^s
3. T2 = T1 × p
4. T3 = (T + T1)
5. T = T3 / 2^s
End For
6. If T ≥ p then X = T − p;
else X = T
Return X

大数运算-乘法

- 对于SM2算法，n域的运算较少，barrett模约更合适
- 可构造全MM域的大数运算库
  - 性能优，通用性好，支持多条256比特曲线
  - 无预计算表
  - 大数乘法/乘方运算最少可低至1个函数

	N MP	P MP	P MM	N MM	GP MM	FP MM
MUL	85.98	53.95	46.24	54.64	54.64	51.36
SQR	82.36	48.99	41.88	50.22	50.39	47.08

0 /10000000 error

大数运算-模逆

- bEEA 二进制扩展欧几里得
- FLT 费马小定理
- 蒙哥马利模逆
- SafeGCD ***

大数运算-模逆

- bEEA 二进制扩展欧几里得
  - 简单，通用
  - 非常量时间

Algorithm 2.22 Binary algorithm for inversion in $\mathbb{F}_p$

INPUT: Prime p and $a \in [1, p-1]$.
OUTPUT: $a^{-1} \bmod p$.
1. $u \leftarrow a, v \leftarrow p$.
2. $x_1 \leftarrow 1, x_2 \leftarrow 0$.
3. While ($u \neq 1$ and $v \neq 1$) do
   3.1 While u is even do
   $u \leftarrow u/2$.
   If x_1 is even then $x_1 \leftarrow x_1/2$; else $x_1 \leftarrow (x_1+p)/2$.
   3.2 While v is even do
   $v \leftarrow v/2$.
   If x_2 is even then $x_2 \leftarrow x_2/2$; else $x_2 \leftarrow (x_2+p)/2$.
   3.3 If $u \ge v$ then: $u \leftarrow u-v$, $x_1 \leftarrow x_1 - x_2$;
   Else: $v \leftarrow v-u$, $x_2 \leftarrow x_2 - x_1$.
4. If $u = 1$ then return($x_1 \bmod p$); else return($x_2 \bmod p$).

大数运算-模逆

- FLT 费马小定理 p
- $a^{-1}= a^{p-2}$

```
felem_square(tmp, in); /* 2^1 */
felem_mul(tmp, in, ftmp); /* 2^2 - 2^0 */
felem_assign(e2, ftmp);
felem_square(tmp, ftmp); /* 2^3 - 2^1 */
felem_square(tmp, ftmp); /* 2^4 - 2^2 */
felem_mul(tmp, ftmp, e2); /* 2^4 - 2^0 */
felem_assign(e4, ftmp);
felem_square(tmp, ftmp); /* 2^5 - 2^1 */
felem_square(tmp, ftmp); /* 2^6 - 2^2 */
felem_square(tmp, ftmp); /* 2^7 - 2^3 */
felem_square(tmp, ftmp); /* 2^8 - 2^4 */
felem_mul(tmp, ftmp, e4);/* 2^8 - 2^0 */
felem_assign(e8, ftmp);
for (i = 0; i < 8; i++)
felem_square(tmp, ftmp); /* 2^16 - 2^8 */
felem_mul(tmp, ftmp, e8); /* 2^16 - 2^0 */
felem_assign(e16, ftmp);
for (i = 0; i < 16; i++) {
felem_square(tmp, ftmp);} /* 2^32 - 2^16 */
felem_mul(tmp, ftmp, e16);/* 2^32 - 2^0 */
felem_assign(e32, ftmp);
for (i = 0; i < 32; i++) {
felem_square(tmp, ftmp);} /* 2^64 - 2^32 */
felem_assign(e64, ftmp);
felem_mul(tmp, ftmp, in);/* 2^64 - 2^32 + 2^0 */
for (i = 0; i < 192; i++) {
felem_square(tmp, ftmp);} /* 2^256 - 2^224 + 2^192 */
felem_mul(tmp, e64, e32); /* 2^64 - 2^0 */
for (i = 0; i < 16; i++) {
felem_square(tmp, ftmp2);} /* 2^80 - 2^16 */
felem_mul(tmp, ftmp2, e16); /* 2^80 - 2^0 */
for (i = 0; i < 8; i++) {
felem_square(tmp, ftmp2);} /* 2^88 - 2^8 */
felem_mul(tmp, ftmp2, e8); /* 2^88 - 2^0 */
for (i = 0; i < 4; i++) {
felem_square(tmp, ftmp2);} /* 2^92 - 2^4 */
felem_mul(tmp, ftmp2, e4); /* 2^92 - 2^0 */
felem_square(tmp, ftmp2); /* 2^93 - 2^1 */
felem_square(tmp, ftmp2); /* 2^94 - 2^2 */
felem_mul(tmp, ftmp2, e2); /* 2^94 - 2^0 */
felem_square(tmp, ftmp2); /* 2^95 - 2^1 */
felem_square(tmp, ftmp2); /* 2^96 - 2^2 */
felem_mul(tmp, ftmp2, in); /* 2^96 - 3 */
felem_mul(tmp, ftmp2, ftmp);
/* 2^256 - 2^224 + 2^192 + 2^96 - 3 */
```

255S +15M

大数运算-FLT模逆

- FLT 费马小定理 n
- $a^{-1}= a^{p-2}$

一种国密SM2算法曲线中对于阶n的高速等时模逆算法
2021 . 12 . 31 观源（上海）科技有限公司
顾星远 邓超国 许森 钱铮 M + S 291

可用蒙哥马利乘法加速

大数运算-蒙哥马利模逆

蒙哥马利模逆
通用寄存器下最高效算法
最终结果需要修正

Algorithm 2.23 Partial Montgomery inversion in $\mathbb{F}_p$

INPUT: Odd integer $p > 2$, $a \in [1, p-1]$, and $n = \lceil \log_2 p \rceil$.
OUTPUT: Either "not invertible" or (x, k) where $n \le k \le 2n$ and $x = a^{-1}2^k \bmod p$.

1. $u \leftarrow a, v \leftarrow p, x_1 \leftarrow 1, x_2 \leftarrow 0, k \leftarrow 0$.
2. While $v > 0$ do
   2.1 If v is even then $v \leftarrow v/2, x_1 \leftarrow 2x_1$;
   else if u is even then $u \leftarrow u/2, x_2 \leftarrow 2x_2$;
   else if $v \ge u$ then $v \leftarrow (v-u)/2, x_2 \leftarrow x_2 + x_1, x_1 \leftarrow 2x_1$;
   else $u \leftarrow (u-v)/2, x_1 \leftarrow x_2 + x_1, x_2 \leftarrow 2x_2$.
   2.2 $k \leftarrow k+1$.
3. If $u \neq 1$ then return("not invertible").
4. If $x_1 > p$ then $x_1 \leftarrow x_1 - p$.
5. Return(x_1, k).

大数运算-蒙哥马利模逆

- BMI指令 tzcnt
- 获取尾部0的数量n m=64-n
- 结合shlx shrx 同步执行 v>>n和 x1<<n

```
    tzcnt tmp0, tmp4;\
    jz .Lbl##1bout ;\
    movq $64, Sft63; subq tmp4, Sft63;\
    ADDq  tmp4, par3(%rsp);\
     SHRx tmp4,tmp0,tmp0;      \
     SHLX Sft63, Sft01, tmp3;  orq tmp3, tmp0;  \
     SHRx tmp4,Sft01, Sft01 ;  \
SHLX Sft63, A3, tmp3;  orq tmp3, Sft01;  \
     SHRx tmp4,A3, A3;\
     SHLX Sft63, A2, tmp3;  orq tmp3, A3; \
          SHRX Sft63, A0, A1;\
           SHLx tmp4,A0,  A0; \
     SHRx tmp4,A2,   A2;\
```

Algorithm 2.23 Partial Montgomery inversion in $\mathbb{F}_p$

INPUT: Odd integer $p > 2$, $a \in [1, p-1]$, and $n = \lceil \log_2 p \rceil$.
OUTPUT: Either "not invertible" or (x, k) where $n \le k \le 2n$ and $x = a^{-1}2^k \bmod p$.

1. $u \leftarrow a, v \leftarrow p, x_1 \leftarrow 1, x_2 \leftarrow 0, k \leftarrow 0$.
2. While $v > 0$ do
   2.1 If v is even then $v \leftarrow v/2, x_1 \leftarrow 2x_1$;
   else if u is even then $u \leftarrow u/2, x_2 \leftarrow 2x_2$;
   else if $v \ge u$ then $v \leftarrow (v-u)/2, x_2 \leftarrow x_2 + x_1, x_1 \leftarrow 2x_1$;
   else $u \leftarrow (u-v)/2, x_1 \leftarrow x_2 + x_1, x_2 \leftarrow 2x_2$.
   2.2 $k \leftarrow k+1$.
3. If $u \neq 1$ then return("not invertible").
4. If $x_1 > p$ then $x_1 \leftarrow x_1 - p$.
5. Return(x_1, k).

大数运算-SafeGCD

- 基于simd实现的最高效、安全方法
- CHES 2019
- Thomas Pornin 2020提出改进方法，通用寄存器下效率更优

Algorithm 2 Extended Binary GCD (optimized algorithm)

Require: Odd modulus m ($m \geq 3$, $m \bmod 2 = 1$), value y ($0 \leq y < m$), and $k > 1$
Ensure: $1/y \bmod m$ (if $\gcd(y, m) = 1$)

```
1:  a ← y, u ← 1, b ← m, v ← 0
2:  for i ≤ i ≤ ⌈(2len(m) − 1)/k⌉ do
3:      n ← max(len(a), len(b), 2k)
4:      ã ← (a mod 2^(k−1)) + 2^(k−1)⌊a/2^(n−k−1)⌋        ▷ ã < 2^2k
5:      b̃ ← (b mod 2^(k−1)) + 2^(k−1)⌊b/2^(n−k−1)⌋        ▷ b̃ < 2^2k
6:      f0 ← 1, g0 ← 0, f1 ← 0, g1 ← 1
7:      for 1 ≤ j ≤ k − 1 do
8:          if ã = 0 mod 2 then
9:              ã ← ã/2
10:         else
11:             if ã < b̃ then
12:                 (ã, b̃) ← (b̃, ã)
13:                 (f0, g0, f1, g1) ← (f1, g1, f0, g0)
14:             ã ← (ã − b̃)/2
15:             (f0, g0) ← (f0 − f1, g0 − g1)
16:         (f1, g1) ← (2f1, 2g1)
17:     (a, b) ← ((af0 + bg0)/2^(k−1), (af1 + bg1)/2^(k−1))
18:     if a < 0 then
19:         (a, f0, g0) ← (−a, −f0, −g0)
20:     if b < 0 then
21:         (b, f1, g1) ← (−b, −f1, −g1)
22:     (u, v) ← (uf0 + vg0 mod m, uf1 + vg1 mod m)
23: v ← v/2^((k−1)⌈(2len(m)−1)/(k−1)⌉) mod m
24: if b ≠ 1 then
25:     return 0 (value y is not invertible)
26: return v
```

大数运算-SafeGCD

- 常量时间　SafeGCD
- 非x86-64 非常量时间 蒙哥马利模逆
- 基于蒙哥马利乘法的FLT效率优于FLT

	SafeGCD	FLT	MMINV	FLTMM
N	3165.50	24235.67	4499.07	15132.78
P	3122.54	11668.61	4418.40	10217.80

固定点点乘

- kG
- 核心为系数k的编码
- 利用预计算表将DOUBLE和ADD简化为查表
- 对k每32比特抽取一个1比特 共8比特 32组
  - kbits[0,32,64,96,...] --> kb[0]
  - kbits[1,33,65,97,...] --> kb[1]
  - kbits[2,34,66,98,...] --> kb[2]
  - kbits[3,35,67,99,...] --> kb[3]
  - kbits[4,36,68,100,..] --> kb[4]
  - ...
  - kbits[31,63,95,127,.] --> kb[31] 31(DOUBLE + ADD)

```
oV= TBL[kb[31];
for(i=30;i>=0;i--)
{
  oV= DOUBLE(oV);
  oV= oV+ TBL(kb[i]);
}
```


固定点点乘

- 预计算表：
- 对于宽度为n的编码，需要存储 G..2^n-1.G个点坐标
- 需要处理系数为0的情况
- 随着n的增加，预计算表尺寸变大
- 大的预计算表收益变低

n	存储点数	计算复杂度
8	256	31 DOUBLE + ADD
9	512	29 DOUBLE + ADD
10	1024	25 DOUBLE + ADD
11	2048	23 DOUBLE + ADD
12	4096	21 DOUBLE + ADD

固定点点乘

- DOUBLE+ADD 4M+4S + 8M +3S
- 2P+Q= P+Q+P

固定点点乘

- 如何抽取比特？
- BMI PEXT
  - 将SRC1中MASK为1的比特 依次放入 DEST中，即mask里有几个1放几位
-
-

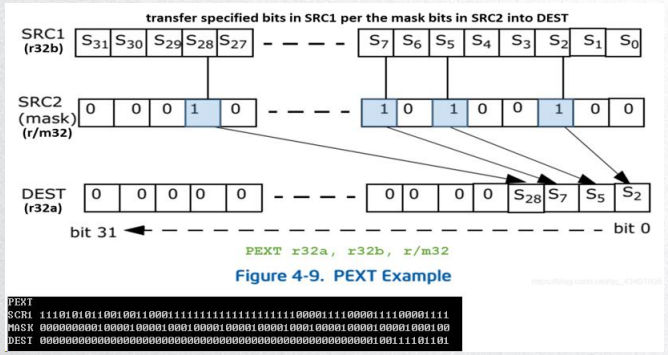


Figure 4-9. PEXT Example

```
PEXT
SCR1 11101010110010011000111111111111111111111110000111100001111000001111
MASK 00000000010000100001000010000100001000010000100001000010000100001000
DEST 00000000000000000000000000000000000000000000000000000000001011101101
```

非固定点点乘

- 固定255次DOUBLE
- ADD次数由非0比特数决定
- 标准实现 1/2非0比特， 127次ADD
- NAF编码 能够大幅降低非0元素数量
  - NAF编码：Non-Adjacent Form 非相邻形式编码,编码后不会出现连续的1
  - 一个简单的例子
  - 7 binary 0 1 1 1 = 0.2^3 + 1.2^2 + 1.2^1 + 1.2^0
  - NAF 1 0 0 -1 = 1.2^3 + 0.2^3 + 0.2^2 + -1. 2^0

非固定点点乘-NAF

- 对于256bit来说，长度最长为257

```
 1  0  0  0  0  1  0  0  0  0 -1  0  0  0 -1  0  0  1  0  0  0  0 -1  0 -1  0 -1  0 -1  0  0  0
 0 -1  0  0  1  0  0  0  0  0  0 -1  0  0 -1  0  1  0  1  0  1  0  1  0  1  0 -1  0 -1  0  0  1
 0 -1  0  0  1  0 -1  0  1  0  1  0  0 -1  0  0  0  0 -1  0  0 -1  0  0  1  0 -1  0  0  0  1  0
 0  1  0 -1  0  0  0  0  0  1  0  0  0  0 -1  0  1  0 -1  0  0  0  1  0 -1  0  0  0 -1  0  0  0
 1  0 -1  0  1  0 -1  0  0 -1  0  0 -1  0  1  0  0  0  0  0  0 -1  0  0  1  0 -1  0 -1  0  0
-1  0 -1  0  0  0  1  0 -1  0  0  0  1  0  0  0  0  0  0  1  0 -1  0  1  0 -1  0 -1  0  1  0  0
 0  1  0 -1  0  1  0  0  0 -1  0  1  0  0  0  1  0 -1  0  1  0  0  0  0  1  0  1  0  1  0 -1  0
 0  0  0  1  0  1  0  0  0 -1  0  0 -1  0  0 -1  0  0  0  1  0  1  0  0  1  0  0 -1  0 -1  0  1
```

非0元素数量≈L/3 = 85

一个良好的随机数非0元素数量约L/2=128 点加运算减少1/3

- 如果我们把窗口放大，如w=3 那么3个比特最多有一个非0值，该值为-3，-1，1，3
- w=4，4比特最多1个非0值，-7 -5 -3 -1 1 3 5 7
- . . .

- NAF编码算法

Algorithm 3.35 Computing the width-w NAF of a positive integer

INPUT: Window width w, positive integer k.
OUTPUT: $\text{NAF}_w(k)$.
1. $i \leftarrow 0$.
2. While $k \geq 1$ do
   2.1 If k is odd then: $k_i \leftarrow k \bmod 2^w$, $k \leftarrow k - k_i$;
   2.2 Else: $k_i \leftarrow 0$.
   2.3 $k \leftarrow k/2$, $i \leftarrow i + 1$.
3. Return$(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$.

- 对应的需要预计算nP

- 对于w>2的情况,宽度约大，非零元素数量越少

W	非零元素数量	非0元素范围 （-2^(w-1)+1, .. -5, -3, -1, 1,3,5,..2^(w-1) -1 ）	预计算表坐标数量
4	50	-7 ~ 7	8/4
5	42	-15 ~ 15	16 /8
6	37	-31 ~ 31	32 /16
7	31	-63 ~ 63	64/32
8	29	-127 ~ 127	128/64
9	25	-255 ~ 255	256/128
10	24	-511 ~ 511	512/256
11	22	-1023 ~ 1023	1024/512
12	19	-2047 ~ 2047	2048/1024

- W= 5, 分割为 257/w个块后，每个块最多一个非零元素

```
 1  0  0  0  0  1  0  0  0  0 15  0  0  0  0  3  0  0  0  0  0  0 11  0  0  0  0 -3  0  0  0  0
 0  7  0  0  0  0  0  0  0  0  0 -9  0  0  0  0  0 -11  0  0  0  0 11  0  0  0 -5  0  0  0  0 -3
 0  0  0  0 13  0  0  0  0  0 -7  0  0  0  0  0  0  0 -9  0  0  0  0  0 -3  0  0  0  0  0  9  0
 0  0  0 -1  0  0  0  0  0  1  0  0  0  0 -13  0  0  0  0  0  0  0  0 -3  0  0  0  0  0 15  0  0  0
 0  0 -13  0  0  0  0  0  0 -9  0  0  0  0  1  0  0  0  0  0  0  0  0  7  0  0  0  0 -5  0  0  0  0
-5  0  0  0  0  0 -3  0  0  0  0  0  1  0  0  0  0  0  0  0 13  0  0  0  0  0  0 11  0  0  0  0  0  0
 0 13  0  0  0  0  0  0  0  3  0  0  0  0  0 13  0  0  0  0  0  0  0  0  0 -11  0  0  0  0 -1  0  0
 0  0  0  5  0  0  0  0  0 -9  0  0  0  0  0 15  0  0  0  0  0  0  9  0  0  0  0  0 11
```

- w=16 , 预计算的点数量迅速增加，为了降低点加数量预计算代价越来越高

```
unsigned w-NAF 256b { n=244; No. of N-0=15 }
-17375   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 -2719   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0 23559   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 -25771   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0 -16819   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 -11801   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0 -24991   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 -9009   0   0   0
  0   0   0   0   0   0   0   0   0   0   0 -25597   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 31191   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0 -12093   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0 -4667   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```


- 计算复杂度= 预计算计算复杂度 + 255 DOUBLE + n ADD
- 预计算标准流程 TBL[m]={P,3P,5P,7P,...};
  - TBL[0]=P;
  - Q= 2P；
  - for(int i=1;i<w;i++)
  - {
  - Q = Q+P; TBL[i]= Q;
  - }
- 计算结果在雅可比坐标系下(x, y, z) 其运算过程中的ADD复杂度为 12M+ 4S+ 7A
- 如果我们把预计算点转换为仿射坐标系，计算复杂度降低为 8M+ 3S+ 7A ，但需要增加模逆开销

非固定点点乘

- 预计算-基于Co-Z方法
- 如果P，Q具有相同的Z坐标，计算复杂度 5M +2S

Addition:

$P_1 = (X_1, Y_1, Z), P_2 = (X_2, Y_2, Z)$ and $P_1 + P_2 = (X_3', Y_3', Z_3')$

$$A = (X_2 - X_1)^2,\ B = X_1 A,\ C = X_2 A,\ D = (Y_2 - Y_1)^2$$

and

$$X_3' = D - B - C\ ,$$
$$Y_3' = (Y_2 - Y_1)(B - X_3) - Y_1(C - B)\ ,$$
$$Z_3' = Z(X_2 - X_1)\ .$$

This addition involves 5M and 2S.

- 计算 P+Q，同时更新P，使得P+Q 和 P Z坐标保持相同 +1M+1S

$$(X_1(X_2 - X_1)^2, Y_1(X_2 - X_1)^3, Z(X_2 - X_1)) \sim (X_1, Y_1, Z).$$

- 总复杂度为 6M+ 3S ＜ 12M + 4S

非固定点点乘-预计算

- SafeGCD 耗时 3800 cycles
- DOUBLE 450cycles ADD 570 cycles
- 可利用同步模逆转化到仿射坐标系
- 输入（x1, y1, z1） (x2, y2, z2)
- 输出 (X1, Y1) (X2，Y2)
  - Z=z1. z2
  - Z= Z^{-1} //$(z1. z2)^{-1}$
  - T=Z. z2 //$z1^{-1}$
  - T1=T. T
  - X1= x1. T1
  - T1=T1. T
  - Y1= Y1. T1
  - T=Z. z1 //$z2^{-1}$
  - T1=T. T
  - X2= x2. T1
  - T1=T1. T
  - Y2= Y2. T1

1个INV + 2S + 9M

非固定点点乘-预计算

- 输入（x1, y1, z1） (x2, y2, z2) (x3, y3, z3)
- 输出 (X1, Y1) (X2，Y2) (X3，Y3)
  - Z0=z1. z2
  - Z=Z0. z3
  - Z= Z^{-1} //$(z1. z2. z3)^{-1}$
  - T=Z. Z0 //$z3^{-1}$
  - T1=T. T
  - X3= x3. T1
  - T1=T1. T
  - Y3= Y3. T1
  - T=z2. z3
  - T=Z. T //$z1^{-1}$
  - T1=T. T
  - X1= x1. T1
  - T1=T1. T
  - Y1= Y1. T1
  - T=z1. z3
  - T=Z. T //$z2^{-1}$
  - T1=T. T
  - X2= x2. T1
  - T1=T1. T
  - Y2= Y2. T1

1个INV + 3S + 16M

随着坐标数增加，计算复杂度显著增加 1INV+nS+ $(n+1)^2$M

非固定点点乘-总计算复杂度

w	非零元素	预计算点	计算复杂度	
4	50	3	3(6M+3S) + INV + 3S+16M + 255 (4M+4S) + 50 (8M+3S)	INV + 1454M + 1182S
5	42	7	7(6M+3S) + INV + 4S+25M + 255 (4M+4S) + 42 (8M+3S)	**INV + 1423M + 1171S**
6	37	15	15(6M+3S) + INV + 5S+36M + 255 (4M+4S) + 37(8M+3S)	INV + 1442M + 1181S
7	31	31	31(6M+3S) + INV + 6S+49M + 255 (4M+4S) + 31(8M+3S)	INV + 1503M + 1212S
8	29	63	63(6M+3S) + INV + 7S+64M + 255 (4M+4S) + 29(8M+3S)	INV + 1694M + 1303S

- 非模逆实现 w=5
  - 7*(6M+3S) + 255 (4M+4S) + 42 (12M+4S) +143M+38S-INV

非固定点点乘- 安全实现

- Co-Z XY方法
- Faster Montgomery and double-add ladders for short Weierstrass curves @CHES2020 Mike Hamburg
- 三个阶段
- 1. setup
- CoZMH_MM_Setup(R,Q);
- 2. 255次 update
- nb=(k[i/64]&(((u64i)1)<<(i%64)))?1:0;
- CoZMH_MM_Update(R+(1-nb)*4,R+nb*4 ,M , YP);
- 3. finish
- CoZMH_MM_FinishXYZ(iout, XQP,XRP,M , YP, Q);

非固定点点乘- 安全实现

- Co-Z XY方法
- 常量时间 8M+3S+7A /bits 优于 5-NAF
- 无需预计算

CoZMH_MM_Setup

```
SM2_SQR_ModP(T, xp);
SM2_SUB1(T,T);
SM2_3X(M,T);   // M= 3.xp^2 + a
SM2_2X(Z,xp+4);
 SM2_SQR_ModP(Z, Z);  //Z= Z^2
 SM2_SQR_ModP(Yp, Z);  //Yq=Yp= Z^4

SM2_SQR_ModP(Xrp, M);  //M^2
SM2_MUL_ModP(T, xp, Z);
SM2_3X(T,T);   // M= 3xp.Z^2
SM2_SUB(Xrp, Xrp, T);
```

CoZMH_MM_Update

```
SM2_MUL_ModP(YR1, Xrp, M);
SM2_2X(YR1,YR1);
SM2_ADD(YR1,Yp, YR1);   //1
SM2_SUB(E, Xqp, Xrp);
SM2_MUL_ModP(F, YR1, E);    //3
SM2_SQR_ModP(G, E);
SM2_MUL_ModP(XRP1, Xrp, G);
SM2_SQR_ModP(H, YR1);     //6
SM2_MUL_ModP(M1, M, F);   //7
SM2_MUL_ModP(YP1, Yp, F);
SM2_MUL_ModP(YP1, YP1, G);   //8  Yp1
 SM2_ADD(K,H, M1);
 SM2_ADD(L,K, M1);       //10
SM2_SUB(M11, XRP1, K);   //11
SM2_MUL_ModP(XSP, H, L);  //12
SM2_SQR_ModP(XTP, XRP1);
  SM2_ADD(XTP,XTP, YP1);
 SM2 MUL ModP(YP11, YP1, H);
```

CoZMH_MM_FinishXYZ

```
    SM2_MUL_ModP(Z, xp, YP);
    SM2_3X(Z,Z);   //
    inverse256_SM2_p(Z,Z);
     SM2_2X(Z,Z);
SM2_SQR_ModP(M1,  M);
 SM2_ADD(T,XQP, XRP);
 SM2_SUB(T, M1, T);
    SM2_MUL_ModP(T, T, xp+4);
    SM2_MUL_ModP(Z, Z, T);    //Z^-1

SM2_SQR_ModP(T, Z);
SM2_MUL_ModP(Z, Z,T);
SM2_MUL_ModP(T, XQP,T);
 SM2_ADD(oxp,T, xp); // X  done  2022

SM2_MUL_ModP(T, XQP,M);
 SM2_MUL_ModP(Z, Z, T);
 SM2 ADD(oxp+4,Z,xp+4);
```


固定点点乘-续

- DOUBLE+ADD 4M+4S +10A+ 8M +3S+7A 12M +7S + 17A
- 2P+Q= （P+Q）+P 可以利用CO-Z加速 13M +5S +13A

```
SM2_SQR_ModP(C, Z0);
SM2_MUL_ModP(D, C, Z0);
SM2_MUL_ModP(X2,Xp,C);
SM2_MUL_ModP(Y2,Yp,D);

SM2_SUB(T, X2, X1);
SM2_SQR_ModP(A, T);
SM2_MUL_ModP(B, X1,A);
SM2_MUL_ModP(C, X2,A);
SM2_SUB(A, C, B);
 SM2_ADD(C, B, C);
SM2_MUL_ModP(A, A,Y1);

SM2_SUB(D, Y2, Y1);
SM2_SQR_ModP(TmpReg, D);
//X3=  D- (B+C)
SM2_SUB(X3, TmpReg, C);
//Y3= (Y2-Y1)(B-X3)-E
SM2_SUB(M, B, X3);
SM2_MUL_ModP(Y3, D,M);
SM2_SUB(Y3, Y3, A);
//Z3= Z(X2-X1)
 SM2_MUL_ModP(ZO, Z0,T);
```

```
//---------------------------------
//B A
//X3 Y3 Z0
SM2_SUB(D, A, Y3);
SM2_SQR_ModP(A, M);
SM2_MUL_ModP(C, B,A);
SM2_MUL_ModP(B, X3,A);
SM2_SUB(A, C, B);
 SM2_ADD(C, B, C);
SM2_MUL_ModP(A, A,Y3);
SM2_SQR_ModP(TmpReg, D);
//X3=  D- (B+C)
SM2_SUB(X3, TmpReg, C);
//Y3= (Y2-Y1)(B-X3)-E
SM2_SUB(M, B, X3);
SM2_MUL_ModP(Y3, D,M);
SM2_SUB(Y3, Y3, A);
//Z3= Z(X2-X1)
 SM2_MUL_ModP(ZO, Z0,M);
```


双倍点

- 对于SM2 一般为kG +lQ形式
- 1. 单独计算kG lQ 然后执行一个 点加
- 2. k,l w-naf编码，同步计算
  - 对于k 可以设置一个大的w,需要一个较大的预计算表
  - l 按照前述非固定点点乘方法
- 3. JSF-5 联合编码
  - 具有更低的汉明重量，计算复杂度较高
  - CO-Z 和 SafeGCD提升了JSF的实用性

双倍点

- 3. JSF-5 联合编码
  - 共12种组合

操作	编码	操作	编码
P	1	Q	5
Q-P	4	P+Q	6
3P	2	3Q	10
3Q+P	11	3Q-P	9
Q+3P	7	Q-3P	3
3Q-3P	8	3P+3Q	12

- 可利用Co-Z方法 同时计算 P+Q 和 P-Q
- 然后利用SafeGCD 将预计算点坐标转到仿射坐标系

协议实现-签名

-

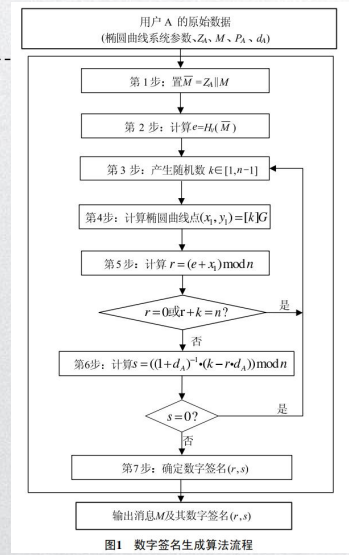


图1 数字签名生成算法流程

协议实现-验签

- x1= r-e mod n

等效于

x1 = (r-e) mod p

or x1= (r-e)+N mod p

- 步骤6得到

x1= X.Z^{-2}

- 可不求逆，验证如下公式：

X =(r-e).Z^2 mod p

or X = ((r-e)+N).Z^2mod p

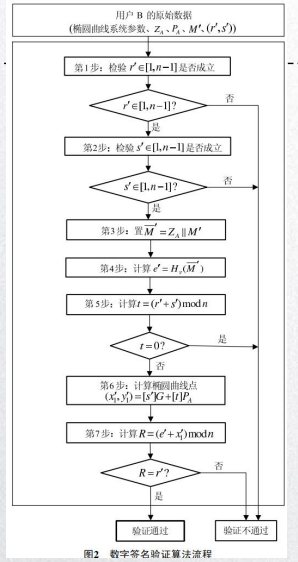


图2 数字签名验证算法流程

协议实现-加解密

-

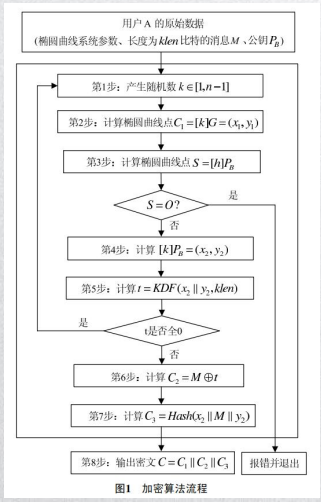


图1 加密算法流程

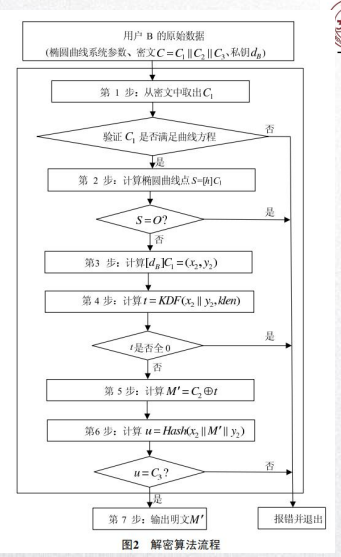


图2 解密算法流程

协议实现-密钥协商

- w=127
- 步骤6 7 可优化

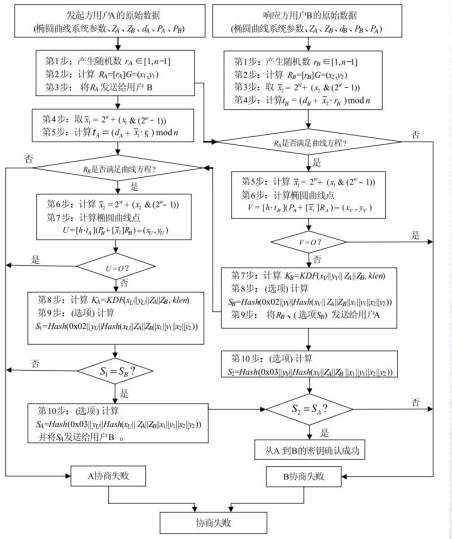


图1 密钥交换协议流程