楕円 ElGamal 暗号の実装

Implementation of ElGamal Cryptosystem on Elliptic Curves

奥　秋　清　次 *1

Seiji　OKUAKI

要 約　インターネットの普及により，データの盗聴，改ざんを防ぐ暗号技術が重要である．暗号には素因数分解問題をもとにしたRSA暗号や，離散対数問題をもとにしたElGamal暗号がある．楕円曲線上の点演算は，離散対数問題に帰着でき，RSA などに比較し，短い鍵長で高い安全性が得られる．この楕円曲線を用いたElGamal 暗号を実用的な鍵長で実装する．

1　はじめに

近年，インターネットは世界的に普及し，情報交換だけではなく商取引など社会のインフラとして利用されている．しかし，インターネットは不特定多数の人が利用するため，悪意のある第三者がデータを盗聴する場合がある．第三者からの盗聴を防ぐためには，データの暗号化が有効である．

楕円曲線暗号は，楕円曲線上の点演算を用いて暗号化する．この点演算は離散対数問題に帰着される一方向性関数であり，RSA などに比べて短い鍵長で高い安全性が得られる．たとえば，鍵長 173 ビットの楕円曲線暗号は，鍵長 1024 ビットのRSA暗号と同等である．楕円曲線暗号は鍵長が短いため，計算能力の低い，限られた記憶容量の装置でも実装が可能である．

本稿では，多倍長演算ライブラ GNU Multiple Precision Arithmetic Library（GNU MP）[1]を用いて，楕円曲線上の ElGamal 暗号[2]を実装する．

2　楕円曲線暗号

暗号に用いられる楕円曲線は，有限体F_p上で定義され，pを素数，整数$a, b \in Z_p$，$x, y \in F_p$として次式で表される．

$$E:\ y^2 = x^3 + ax + b\ \ (mod\ p)$$

ただし，暗号に用いる楕円曲線は $p \geq 5$，$4a^3 + 27b^2 \neq 0$を満たす必要がある．

2.1　楕円曲線上の点演算

有限体F_p上の有理点の集合は，次の演算により可換群を成す．この有理点の集合を$E(F_p)$で示す．ここで，単位元は無限遠点$O = (\infty, \infty)$ である．

$$E(F_p) = \{(x, y) \in F_P^2 | y^2 = x^3 + ax + b\} \cup \{O\}$$

定義1．Pと QをE上の異なる有理点とする．Pと Q を結ぶ直線を lとし，lとEの交わる点を$\acute{R}$とする．点$\acute{R}$についてx軸と対称な点を$R = P + Q$ とする．$P = Q$ の場合，直線lはPにおける接線とする．

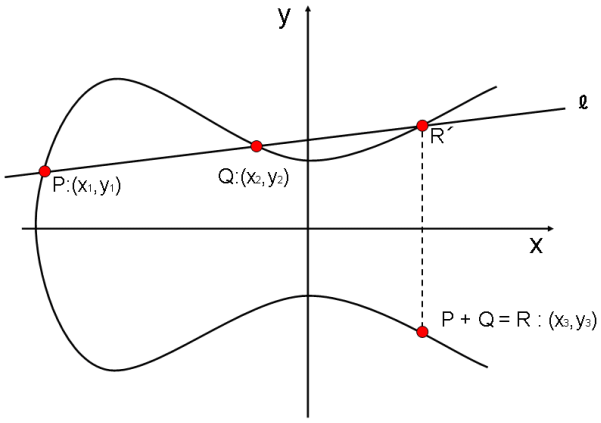


図１　楕円曲線上の点演算

楕円曲線E上の点を$P = (x_1, y_1)$，$Q = (x_2, y_2)$ とした場合，定義 1 の演算により求められる点 $R = P + Q = (x_3, y_3)$は，次式で与えられる[3]．

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

*1　電子情報技術科
Department of Electronics and Information Engineering

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & if \;\; P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} & if \;\; P = Q \end{cases}$$

無限遠点Oは単位元なので，$P + O = O + P = P$である．Pの逆元$-P$は，$-P = (x_1, -y_1)$である．集合$E(F_p)$の要素数, すなわち位数$\#E(F_p)$ は, Hasse の定理 [2,4]より

$$p + 1 - 2\sqrt{p} \leq \#E(F_p) \leq p + 1 + 2\sqrt{p}$$

である．暗号では $\#E(F_p) = nN$とした場合，nが小さな整数，かつNが大きな素数となる 楕円曲線を用いる．通常， $n = 1$でNが大きな素数となる楕円曲線を選ぶ.

群$E(F_p)$から$P \in E(F_p)$を取り，整数 $x \in Z$ との積を$Q = xP$として, このPとQから x を求めることを楕円離散対数問題と呼び, 解くことが非常に難しいと考えられている．現在, x のサイズが 160 ビット以上あれば現実的な時間内に解けないと信じられている．楕円離散対数問題を解く手法には， Baby step Giant step, ρ 法, λ 法などがあるが, x のサイズが大きいとPとQから x を求めることはできない.

２．２　楕円 ElGamal 暗号

ElGamal 暗号 は, 暗号文が2成分で構成される暗号である. Alice から Bob にメッセージ m を送る場合, Bob が秘密鍵（Private Key）を生成し，その秘密鍵をもとに公開鍵（Public Key）を作成する．Bob は秘密鍵を他者に漏洩しないように保管し，公開鍵を Alice に渡す．Alice は Bob の公開鍵を用いて，mを暗号化（Encrypt）して Bob に送信する．Bob は自身の秘密鍵を用いて，暗号文（Ciphertext）を復号（Decrypt）してmに戻す.

［暗号の準備　by　Bob］

1. 有限体 F_p 上の楕円曲線であり，位数 $\#E(F_p)$ が大きな素数となる楕円曲線を選ぶ.
2. 楕円曲線上の点 $P \in E(F_p)$を選ぶ.
3. Bob は秘密鍵となる整数$s \in Z_N$を選び $B = sP$を計算する.
4. 楕円曲線 E , 有限体 F_p, Bob の公開鍵としP, Bを公開する.

離散対数問題は難しいので，PとBを公開しても，秘密鍵sは計算できない.

［暗号化　by　Alice］

1. Alice は Bob の公開鍵P，Bを得る.
2. メッセージ mを楕円曲線上の点$M \in E(F_p)$に対応付ける.
3. ランダムな整数 $k \in Z_N$ を選び，$M_1 = kP$, $M_2 = M + kB$を計算する.
4. 暗号文 M_1 , M_2 を Bob に送信する.

［復号　by　Bob］

1. Bob は Alice の暗号文M_1 , M_2を受信する.
2. $M = M_2 - sM_1$ を計算する.
   暗号文は以下に計算により復号できる.
$$\begin{aligned} M_2 - sM_1 &= (M + kB) - s(kP) \\ &= M + k(sP) - skP \\ &= M \end{aligned}$$
3. 楕円曲線上の点 Mをメッセージmに変換する.

３　アルゴリズム

３．１　楕円曲線上の点演算

$PointAdd(R, P, Q)$は楕円曲線上の有理点$P = (x_1, y_1)$，$Q = (x_2, y_2)$が与えられると$R = P + Q = (x_3, y_3)$を計算する.

アルゴリズム1　　$PointAdd(R, P, Q)$

if $x_1 = \infty$ **and** $y_1 = \infty$
 then $R \leftarrow (x_2, y_2)$
  return
if $x_2 = \infty$ **and** $y_2 = \infty$
 then $R \leftarrow (x_1, y_1)$
  return
if $x_1 = x_2$
 then if $y_1 = -y_2$
  then $R \leftarrow (\infty, \infty)$
   return
  else $\lambda \leftarrow \dfrac{3x_1^2 + a}{2y_1}$
 else $\lambda \leftarrow \frac{y_2 - y_1}{x_2 - x_1}$
$x_3 \leftarrow \lambda^2 - x_1 - x_2$
$y_3 \leftarrow \lambda(x_1 - x_3) - y_1$
$R \leftarrow (x_3, y_3)$
return

３．２　２進展開法

楕円曲線暗号では，点 $P \in E(F_p)$ の α倍の計算が多く発生する．２進展開法[3,4]は，この αPの計算を高速化するアルゴリズムである．２進展開法は，整数αを

$$\alpha = \sum_{i=0}^{len-1} \alpha_i 2^i$$

として展開し，$\alpha_i = 1$のときPを加算，繰り返すたびに倍算する．ここで，len はビット長
$len = \lfloor \log_2 \alpha \rfloor$，　$\alpha_i \in \{0,1\}$ である．
$Binary(Q, \alpha, P)$は，$Q = \alpha P$を計算する．

アルゴリズム2　　$Binary(Q, \alpha, P)$
$Q \leftarrow (\infty, \infty)$
$len \leftarrow BitsLength(\alpha)$
for $i \leftarrow len - 1$ **downto** 0
　do $PointAdd(R, Q, Q)$
　　$Q \leftarrow R$
　　if $mpz_tsbit(\alpha, i) = 1$
　　　then $PointAdd(R, Q, P)$
　　　　$Q \leftarrow R$
return

$mpz_$で始まる関数はGNU MPの関数である．
$mpz_tsbit(\alpha, i)$は，αのビット位置iの１ビットを，
$BitsLength(\alpha)$はαのビット長 $\lfloor \log_2 \alpha \rfloor$を返す．

アルゴリズム3　　$BitsLength(\alpha)$
$hw \leftarrow mpz_popcount(\alpha)$
if $hw = 0$
　then return 0
$pos \leftarrow mpz_scan1(\alpha, 0)$
for $i \leftarrow 1$ *to* $hw - 1$
　do $pos \leftarrow mpz_scan1(\alpha, pos + 1)$
return $pos + 1$

$mpz_popcount(\alpha)$はαのビット数，すなわちハミング重みを計算する．$mpz_scan1(\alpha, st)$は，αについてビット位置stからMSB(Most Significant Bit)に向けてビット’1’を検索し，検索したビット’1’の位置を返す．

３．３　pを法とした平方根

次の合同式において解がある場合，qを p での平方剰余，そうでない場合を平方非剰余とよぶ．

$$x^2 \equiv q \pmod{p}$$

平方剰余，平方非剰余であるかはLegendre記号により計算する．

$$\left(\frac{q}{p}\right) = q^{\frac{p-1}{2}} \pmod{p}$$

$$\left(\frac{q}{p}\right) = \begin{cases} 1 & \text{平方剰余} \\ -1 & \text{平方非剰余} \\ 0 & q\text{が}p\text{の倍数} \end{cases}$$

pを法としたqの平方根の計算は，文献[5]のアルゴリズムを用いる．$SqrtMod(result, q, p)$はq，pから平方根$result$を求める関数である．この関数の中で$GetQNR(qnr, p)$，$SqrtModSub(z, p, h, b, r)$ を用いる．

アルゴリズム4　　$SqrtMod(result, q, p)$
$GetQNR(g, p)$
$t \leftarrow p - 1$
$rem \leftarrow t \pmod 2$
$r \leftarrow 0$
while $rem = 0$
　do $t \leftarrow \frac{t}{2}$
　　$rem \leftarrow t \pmod 2$
　　$r \leftarrow r + 1$
$h \leftarrow g^t \pmod p$
$b \leftarrow q^t \pmod p$
$SqrtModSub(z, p, h, b, r)$
$result \leftarrow q^{\frac{t+1}{2}} z \pmod p$
return

アルゴリズム５　　$GetQNP(qnr, p)$
for $q \leftarrow 2$ **to** $p - 1$
　do if $mpz_legendre(q, p) = -1$
　　then $qnr \leftarrow q$
　　　return
return *error*

アルゴリズム６　　$SqrtModSub(z, p, h, b, r)$
$z \leftarrow 1$
$mpz_invert(h, h, p)$
for $i \leftarrow r - 2$ **downto** 0
　do if $b^{2^i} \pmod p = -1$
　　then $z \leftarrow zh \pmod p$
　　　$b \leftarrow bh^2 \pmod p$
　　$h \leftarrow h^2 \pmod p$
return

$GetQNP(qnr,p)$はpを法とした平方非剰余 qnr を計算する．$mpz_legrendre(q,p)$ は$\left(\frac{q}{p}\right)$ を計算し，$mpz_invert(h,h,p)$はpを法としたh（2 番目の引数）の逆数を h（1 番目の引数）に返す.

３．４　メッセージと楕円曲線上の点の対応

メッセージ mは整数で表される. このmを楕円曲線上の点 Mに対応付ける必要がある．Koblitz が提案した方法 [2] は，楕円曲線 $y^2 = x^3 + ax + b \ (mod\ P)$と$m\ (0 \le m \le \frac{p}{100})$が与えられると，$m$を 100 倍し，$i$（$0 \le i \le 99$）を可算して異なる 100 個の$x_i$を生成する．この 100 個の$x_i$から平方剰余となる$x_i$を探す.

アルゴリズム7　　$MsgToPnt(M,m)$
for $i \leftarrow 0$ **upto** 99
  do $x_i \leftarrow 100m + i$
    $z \leftarrow x_i^3 + ax_i + b$
    if $mpz_legendre(z,p) = 1$
      then $SqrtMod(y_i,z,p)$
        $M \leftarrow (x_i,y_i)$
        return
return *error*

楕円曲線上の点 $M = (x,y)$ からメッセージmを得る場合, m = $\left\lfloor \frac{x}{100} \right\rfloor$で計算する. $PntToMsg(m,M)$は楕円曲線上の点Mをメッセージmに変換する.

アルゴリズム8　$PntToMsg(m,M)$
$m \leftarrow \lfloor x/100 \rfloor$
return

３．５　群の生成元

楕円曲線上の点の集合$E(F_p)$から任意の元を選び，その元に演算を繰り返すと集合$E(F_p)$のすべての元が生成できる．このような元を生成元（Generator）とよぶ．$GetGenerator(M)$は，群の生成元を作成する関数である．乱数$x_i \in Z_N$を$mpz_urandomb(x_i, state, LEN)$を用いて発生させ, $z = x_i^3 + ax_i + b$を計算し，zがpを法とした平方剰余であるか調べる．平方剰余であれば, 楕円曲線上の点であるので, 平方根y_iを計算し，(x_i,y_i)を生成元とする．平方非剰余ならば, 乱数を再生成して繰り返す.

アルゴリズム 9　　$GetGenerator(M)$
for $i \leftarrow 0$ **upto** 99
  do $mpz_urandomb(x_i,state,LEN)$
    $z \leftarrow x_i^3 + ax_i + b$
    if $mpz_legendre(z,p) = 1$
      then $SqrtMod(y_i,z,p)$
        $M \leftarrow (x_i,y_i)$
        return
return *error*

３．６　暗号化と復号

暗号化と復号のアルゴリズムは, それぞれ$ElGamal\ Enc(M_1,M_2,P,B,m)$，$ElGamalDec(m,M_1,M_2,P,B,s)$である.

アルゴリズム10　　$ElGamalEnc(M_1,M_2,P,B,m)$
$MsgToPnt(M,m)$
$RandomNumber(k)$
$Binary(M_1,k,P)$
$Binary(tmp,k,B)$
$PointAdd(M_2,tmp,M)$
return

アルゴリズム11　　$ElGamalDec(m,M_1,M_2,P,B,s)$
$Binary(M,s,M_1)$
$M \leftarrow -M$
$PointAdd(M,M_2,M)$
$PntToMsg(m,M)$
return

４　実装

楕円 ElGamal 暗号は，オペレーティングシステム CentOS6.6, 多倍長演算ライブラリ GNU MP6.1.1 を用い，C 言語で実装する．GNU MP には多倍長整数での四則演算，剰余演算，ベキ乗演算があるが，p を法とした平方根求める関数がないので 3.3 節のアルゴリズムで自作する.

４．１　楕円曲線の選定

楕円曲線暗号は鍵長 160 ビット以上で安全とされている．鍵長を 160 ビット以上にするためには$N > 2^{160}$

となる楕円曲線を選択する必要がある．今回，文献[4]の楕円曲線を利用する．

Elliptic Curve parameters:

$$y^2 = x^3 + ax + b \ (mod\ p)$$

$$a = 10$$

$$b = 1348462411414361312611054113116931\\ 0875806949186774222942 74$$

$$p = 2^{190} + 129\\ = 1569275433846670190958947355801916\\ 6040255888611160086283 53$$

$$\#E(F_p) = N\\ = 1569275433846670190958947355780 28\\ 7040305255540896946997883$$

$$\infty = 9999999999999999999999999999999999\\ 9999999999999999999999999999$$

実装において，無限大をpより大きい値とし，楕円曲線Eのパラメータa，b，p，N，∞を大域変数とする．

４．２　点のデータ構造

楕円曲線上の点は，x成分とy成分があるので構造体にまとめる．関数の引数でこの構造体を渡す場合，実引数で&演算子を使用せずに記述できるように配列で表現する．

```
typedef struct ECPoint {
    mpz_tx;
    mpz_ty;
}ECPOINT;
typedef  ECPOINT ECP[1];
```

４．３　実行例

実行例ではメッセージ Message=1234567890 を暗号化し，Decrypted Message=1234567890 に復号する．

```
Parameters of Elliptic Curves
a = 10
b = 1348462411414361312611054113116931087580694918677422294274
p = 1569275433846670190958947355801916604025588861116008628353
N = 1569275433846670190958947355780287040305255540896946997883
Private Key:
s = 389001560119629233978738432295717387897423844232833642882
Public Keys:
P=(107263171103328529779176639115539783849390633718964567677 0,
   8765946074942521710319320370504278398314324114613067421 26)
B=(151700606918453533952105931382063647698646070646156728491 6,
   691856718019860628562552013491680548371100883858491989346)
Message = [1234567890]
Ciphertext:
M1=(151700606918453533952105931382063647698646070646156728491 6,
   691856718019860628562552013491680548371100883858491989346)
M2=(256331866128306448557420174558672715304034267451397875 29,
   152591414133756404334749653046130509977901704231865529374 7)
Decrypted Message = [1234567890]
```

５　まとめ

多倍長演算ライブラリ GNU MP を用いて，楕円 ElGamal 暗号を実用的な鍵長 190 ビットで実装した．楕円曲線暗号が安全であるためには，適切な楕円曲線を選択しなければならない．今回，文献[4]に記載されている楕円曲線を用いたが，今後は楕円曲線の選定から始めたい．

参考文献

1) The GNU Multiple Precision Arithmetic Library, https://gmplib.org/.
2) L. Washington. Elliptic Curves: Number Theory and Cryptography, Chapman and Hall/CRC, 2003.
3) D. Stinson. Cryptography: Theory and Practice Third Edition, Chapman and Hall/CRC, 2006.
4) Blake, G. Seroussi, and N. Smart. Elliptic Curves in Cryptography, Cambridge University Press, 1999.
5) 木田祐司, 初等整数論, 朝倉書店, 2001.

付録

本稿で作成したプログラムは，ファイル ec_elgamal.h，ec_elgamal.c である．アルゴリズムを分かりやすくするためエラー処理を除いている．

[ec_elgamal.h]

```
#define Str_a "10"
#define Str_b "134846241141436131261105411311693108758069
4918677422294274"
#define Str_p "156927543384667019095894735580191660402558
8861116008628353"
#define Str_N "156927543384667019095894735578028704030525
5540896946997883"
#define Str_Infinity "999999999999999999999999999999999999
99999999999999999999999999"

typedef struct ECPoint {
    mpz_t x;
    mpz_t y;
}ECPOINT;
typedef   ECPOINT ECP[1];

#define   RANDOM_LEN  191
#define   OK           1
#define   NG          -1

void SetParameters(void);
void PointAdd(ECP R, ECP P, ECP Q);
int  BitsLength(mpz_t num);
void Binary(ECP Q, mpz_t alpha, ECP P);
int  MsgToPnt(ECP M, mpz_t m);
int  PntToMsg(mpz_t m, ECP M);
int  ECPointToMessage(mpz_t message, ECP ecp);
int  ElGamalEnc(ECP M1, ECP M2, ECP P, ECP B, mpz_t m);
int  ElGamalDec(mpz_t m, ECP M1, ECP M2, ECP P, ECP B,
               mpz_t s);
void SqrtMod(mpz_t answer, mpz_t a, mpz_t p);
void SqrtModSub(mpz_t s, mpz_t p, mpz_t h, mpz_t b, int r);
int  GetQNR(mpz_t qnr, mpz_t p);
int  GetGenerator(ECP ecp);
void RandomNumber(mpz_t rn);
```

[ec_elgamal.c]

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <gmp.h>
#include  "ec_elgamal.h"

/* Parameters of Elliptic Curves */
mpz_t     a;
mpz_t     b;
mpz_t     p;
mpz_t     N;
mpz_t Infinity;

void main(void)
{
    ECP P, B, M1, M2;
    mpz_t  s, message, decrypted_message;

     mpz_inits(P->x, P->y, B->x, B->y, M1->x, M1->y,
         M2->x, M2->y, s, message, decrypted_message, NULL);

    SetParameters();
    printf("Parameters of Elliptic Curves¥n");
    gmp_printf("a = %Zd¥n", a);
    gmp_printf("b = %Zd¥n", b);
    gmp_printf("p = %Zd¥n", p);
    gmp_printf("N = %Zd¥n", N);

    GetGenerator(P);
    RandomNumber(s);
    gmp_printf("Private Key: s = %Zd¥n", s);
    Binary(B, s, P);            // B <- sP
    printf("Public Keys:¥n");
    gmp_printf("P=(%Zd, %Zd)¥n", P->x, P->y);
    gmp_printf("B=(%Zd, %Zd)¥n", B->x, B->y);
    mpz_set_str(message, "1234567890", 10);
    gmp_printf("Message = [%Zd]¥n", message);

    ElGamalEnc(M1, M2, P, B, message);    // Encrypt
    printf("Ciphertext:¥n");
    gmp_printf("M1=(%Zd, %Zd)¥n", M1->x, M1->y);
    gmp_printf("M2=(%Zd, %Zd)¥n", M2->x, M2->y);

    // Decrypt
    ElGamalDec(decrypted_message, M1, M2, P, B, s);
    gmp_printf("Decrypted Message = [%Zd]¥n", d
               ecrypted_message);

    mpz_clears(P->x, P->y, B->x, B->y, M1->x, M1->y, M2->x,
               M2->y, s, message, decrypted_message, NULL);
}

int ElGamalEnc(ECP M1, ECP M2, ECP P, ECP B, mpz_t m)
{
    ECP   M, tmp;
    mpz_t k;

    mpz_inits(M->x, M->y, tmp->x, tmp->y, k, NULL);

    if ((MsgToPnt(M, m)) == NG) {
        printf("Encryption Error¥n");
        mpz_clears(M->x, M->y, tmp->x, tmp->y, k, NULL);
        return NG;
    }
    RandomNumber(k);
    Binary(M1, k, P);           // M1 <- kP
    Binary(tmp, k, B);          // M2 <- M+kB
    PointAdd(M2, tmp, M);

    mpz_clears(M->x, M->y, tmp->x, tmp->y, k, NULL);
    return OK;
}
```

```
int ElGamalDec(mpz_t m, ECP M1, ECP M2, ECP P, ECP B,
               mpz_t s)
{
    ECP   M;
    mpz_t zero;

    mpz_inits(M->x, M->y, NULL);
    mpz_init_set_ui(zero, 0L);

    Binary(M, s, M1);            // sM1
    mpz_sub(M->y, zero, M->y);   // -sM1=(x,-y)
    mpz_mod(M->y, M->y, p);

    PointAdd(M, M2, M);          // M <- M2-sM1
    mpz_mod(M->x, M->x, p);
    mpz_mod(M->y, M->y, p);

    PntToMsg(m, M);
    mpz_clears(M->x, M->y, NULL);
}

void SetParameters(void)
{
    mpz_inits(a, b, p, Infinity, NULL);
    mpz_set_str(a, Str_a, 10);
    mpz_set_str(b, Str_b, 10);
    mpz_set_str(p, Str_p, 10);
    mpz_set_str(N, Str_N, 10);
    mpz_set_str(Infinity, Str_Infinity, 10);
}

void Binary(ECP Q, mpz_t alpha, ECP P)
{
    ECP  R;
    Int  len, i;

    mpz_inits(R->x, R->y, NULL);

    mpz_set(Q->x, Infinity);
    mpz_set(Q->y, Infinity);
    len = BitsLength(alpha);
    for (i = len -1 ; i >= 0 ; i—) {
        PointAdd(R, Q, Q);               // R <- 2Q
        mpz_set(Q->x, R->x);
        mpz_set(Q->y, R->y);
        if (mpz_tstbit(alpha, i) == 1) {
            PointAdd(R, Q, P);           // R <- Q+P
            mpz_set(Q->x, R->x);
            mpz_set(Q->y, R->y);
        }
    }
    mpz_clears(R->x, R->y, NULL);
    return;
}
```

```
void PointAdd(ECP R, ECP P, ECP Q)
{
    mpz_t lambda, numerator, denominator, tmp;

    if ((mpz_cmp(P->x, Infinity) == 0) &&  // Q+O=Q
        (mpz_cmp(P->y, Infinity) == 0)) {
        mpz_set(R->x, Q->x);
        mpz_set(R->y, Q->y);
        return;
    }
    if ((mpz_cmp(Q->x, Infinity) == 0) &&  // P+O=P
        (mpz_cmp(Q->y, Infinity) == 0)) {
        mpz_set(R->x, P->x);
        mpz_set(R->y, P->y);
        return;
    }

    mpz_inits(lambda, numerator, denominator, tmp, NULL);
    if (mpz_cmp(P->x, Q->x) == 0) {
        mpz_neg(tmp, Q->y);
        mpz_mod(tmp, tmp, p);
        if (mpz_cmp(P->y, tmp) == 0) {   // Q=-P
            mpz_set(R->x, Infinity);
            mpz_set(R->y, Infinity);
            mpz_clears(lambda, numerator, denominator,
                       tmp, NULL);
            return;
        } else {                         // P=Q
            mpz_powm_ui(tmp, P->x, 2, p);
            mpz_mul_ui(tmp, tmp, 3);
            mpz_add(numerator, tmp, a);
            mpz_mod(numerator, numerator, p);
            mpz_mul_ui(tmp, P->y, 2);
            mpz_invert(denominator, tmp, p);
            mpz_mul(lambda, numerator, denominator);
            mpz_mod(lambda, lambda, p);
        }
    } else {                             // P!=Q
        mpz_sub(numerator, Q->y, P->y);
        mpz_sub(tmp, Q->x, P->x);
        mpz_invert(denominator, tmp, p);
        mpz_mul(lambda, numerator, denominator);
        mpz_mod(lambda, lambda, p);
    }
    mpz_powm_ui(tmp, lambda, 2, p); // x3 <- lambda^2-x1-x2
    mpz_sub(tmp, tmp, P->x);
    mpz_sub(R->x, tmp, Q->x);
    mpz_mod(R->x, R->x, p);

    mpz_sub(tmp, P->x, R->x);    // y3 <- lambda(x1-x3)-y1
    mpz_mul(tmp, lambda, tmp);
    mpz_sub(R->y, tmp, P->y);
    mpz_mod(R->y, R->y, p);

    mpz_clears(lambda, numerator, denominator, tmp, NULL);
    return;
}
```

```
int BitsLength(mpz_t alpha)
{
     int hw,pos,i;

     hw = mpz_popcount(alpha);
     if (hw == 0)
         return 0;
     pos = mpz_scan1(alpha, 0);
     for (i = 1; i <= hw - 1; i++)
         pos = mpz_scan1(alpha, pos + 1);
     return pos + 1;
}

void SqrtMod(mpz_t result, mpz_t q, mpz_t p)
{
    mpz_t g, t, b, h, z, tmp, rem;
    int r;

    mpz_inits(g, t, b, h, z, tmp, rem, NULL);

    GetQNR(g, p);    // QNR
    mpz_sub_ui(t, p, 1);
    mpz_tdiv_r_ui(rem, t, 2);
    for(r = 0 ; mpz_get_ui(rem) == 0 ; r++) {
        mpz_tdiv_q_ui(t, t, 2);
        mpz_tdiv_r_ui(rem, t, 2);
    }
    mpz_powm(h, g, t, p);       // h <- g^t modp
    mpz_powm(b, q, t, p);       // b <- q^t modp
    SqrtModSub(z, p, h, b, r);
    mpz_add_ui(tmp, t, 1);
    mpz_tdiv_q_ui(tmp, tmp, 2);
    mpz_powm(tmp, q, tmp, p);
    mpz_mul(result, tmp, z);
    // result <- q^{(t+1)/2}z modp
    mpz_mod(result, result, p);
    mpz_clears(g, t, b, h, z, tmp, rem, NULL);
}

int GetQNR(mpz_t qnr, mpz_t p)
{
    mpz_t q;
    int   result = NG;

    mpz_init_set_ui(q, 2);
    while(mpz_cmp(q, p) < 0) {
        if(mpz_legendre(q, p) == -1) {   // QNR
            mpz_set(qnr, q);
            result = OK;
            break;
        }
        mpz_add_ui(q, q,1);
    }
    mpz_clear(q);
    return result;
}
```

```
void SqrtModSub(mpz_t z, mpz_t p, mpz_t h, mpz_t b, int r)
{
    mpz_t minus_one, tmp, h2;
    int  i;

    mpz_inits(minus_one, tmp, h2, NULL);

    mpz_sub_ui(minus_one, p, 1);         // -1
    mpz_set_ui(z, 1);
    mpz_invert(h, h, p);
    for(i = r-2 ; i >= 0 ; i—) {
        mpz_ui_pow_ui(tmp, 2, i);
        mpz_powm(tmp, b, tmp, p);        // b^{2^{i}}
            if(mpz_cmp(tmp, minus_one) == 0) {
                mpz_mul(z, z, h);
                mpz_mod(z, z, p);        // z <- zh modp
                mpz_powm_ui(h2, h, 2, p);
                mpz_mul(b, h2, b);
                mpz_mod(b, b, p);        // b <- bh^2 modp
            }
            mpz_powm_ui(h, h, 2, p);
    }
    mpz_clears(minus_one, tmp, h2, NULL);
}

int MsgToPnt(ECP M, mpz_t m)
{
    mpz_t x, y, z, tmp;
    int   i;

    mpz_inits(x, y, z, tmp, NULL);

    for (i = 0 ; i < 100 ; i++) {
        mpz_mul_si(x, m, 100);  // x <- 100*m+i
        mpz_add_ui(x, x, i);
        mpz_pow_ui(z, x, 3);   // z <- x^{3}+ax+b modp
        mpz_mul(tmp, a, x);
        mpz_add(z, z, tmp);
        mpz_add(z, z, b);
        mpz_mod(z, z, p);
        if (mpz_legendre(z, p) == 1){    // QR
            SqrtMod(y, z, p);
            mpz_set(M->x, x);
            mpz_set(M->y, y);
            mpz_clears(x, y, z, tmp, NULL);
            return OK;
        }
    }
    mpz_clears(x, y, z, tmp, NULL);
    return NG;
}

int PntToMsg(mpz_t m, ECP M)
{
    mpz_fdiv_q_ui(m, M->x, 100);
}
```

```
int GetGenerator(ECP M)
{
    mpz_t x, t, z, y;
    gmp_randstate_t state;
    int   i, result = NG;

    mpz_inits(x, y, z, t, NULL);
    gmp_randinit_default(state);
    gmp_randseed_ui(state, 1L);
    for (i = 0; i < 100 ; i++) {
        mpz_urandomb(x, state, RANDOM_LEN);
        mpz_pow_ui(z, x, 3);   // z <- x^3+ax+b modp
        mpz_mul(t, a, x);
        mpz_add(z, z, t);
        mpz_add(z, z, b);
        mpz_mod(z, z, p);
        if (mpz_legendre(z, p) == 1){    // QR
            SqrtMod(y, z, p);
            mpz_set(M->x, x);
            mpz_set(M->y, y);
            result = OK;
            break;
        }
    }
    mpz_clears(x, y, z, t, NULL);
    return result;
}

void RandomNumber(mpz_t rn)
{
    gmp_randstate_t state;

    gmp_randinit_default(state);
    gmp_randseed_ui(state, (unsigned long)time(NULL) );
    mpz_urandomb(rn, state, RANDOM_LEN);
    mpz_mod(rn, rn, N);
}
--------------------------------------------------------------------------------
```