TIPE – Courbes elliptiques et nombres premiers

Implémentation de la méthode de Lenstra

Samuel MIMRAM

2001–2002

1 Détails de la mise en œuvre

Pour les variables numériques, nous n'utiliserons pas le type `int` limité à 32 bits (soit au maximum 8589934591) mais le type `largeint` qui nous permet de manipuler des entiers de taille quelconque.

1.1 La classe `ec_point`

Nous allons travailler avec la courbe elliptique : $E : y^2 \equiv x^3 + ax + b \pmod{p} \cup \mathcal{O}$ avec $p \neq 2$ et $p \neq 3$. Les variables correspondantes `largeint a`, `b`, `p` sont définies globalement. Nous définissons alors la classe `ec_point` dans laquelle nous allons implémenter les opérations dans la courbe elliptique :

```
1  class ec_point
    {
    public:
        largeint x;
5       largeint y;
        bool Infty;

        ec_point() {Infty = false;};
        ec_point(largeint X, largeint Y) {x = X; y = Y; Infty = false;};
10      ec_point(bool is0) {Infty = is0;};
        ~ec_point() {};
        void Print();

        void operator = (ec_point P) {x = P.x; y = P.y; Infty = P.Infty;};
15      friend bool operator == (ec_point P1, ec_point P2);
        friend ec_point operator + (ec_point P1, ec_point P2);
        friend ec_point operator * (largeint m, ec_point P);
        friend ec_point operator * (int m, ec_point P);
    };
```

La variable booléenne `Infty` vaut `true` si le point est $\mathcal{O}$; sinon le point est défini par ses coordonnées `x` et `y`. L'élément neutre $\mathcal{O}$ est donc défini par :

```
#define O_neutre ec_point(true)
```

Nous implémentons ensuite l'addition de groupe par transcription directe de son expression analytique.

```

1  ec_point operator + (ec_point P1, ec_point P2)
   {
       ec_point P3;
       largeint lambda;
5
       if (P1.Infty)
           P3 = P2;
       else if (P2.Infty)
           P3 = P1;
10      else if (P1.x == P2.x && P1.y == mod (-P2.y, p))
           P3.Infty = true;
       else
       {
           if (P1 == P2)
15          {
               if (gcd(P1.y, p) != 1)
               {
                   //Le facteur n'est pas inversible !
                   error_factor = P1.y;
                   error = true;
20              }
               lambda = mod((3 * P1.x * P1.x + a) * invModN(2 * P1.y, p), p);
           }
           else
25          {
               if (gcd(P2.x - P1.x, p) != 1)
               {
                   //Le facteur n'est pas inversible !
                   error_factor = P2.x - P1.x;
                   error = true;
30              }
               lambda = mod((P2.y - P1.y) * invModN(P2.x - P1.x, p), p);
           }
           P3.x = mod((lambda * lambda - P1.x - P2.x), p);
           P3.y = mod((lambda * (P1.x - P3.x) - P1.y), p);
35      }
       return P3;
   }
}

```

1.2 Calcul de kP

L'algorithme utilisé est une adaptation de l'algorithme classique d'exponentiation rapide. Elle est fondée sur le fait que si $k = \overline{k_n k_{n-1} \dots k_1 k_0}^{(2)}$ est l'écriture en base 2 de k alors $kP = \sum_{i=0}^n k_i (2^i P)$. Il permet de calculer kP avec $O(\ln n)$ opérations au lieu de $O(n)$ pour l'addition naïve. Voici son implémentation pratique :

```

1  ec_point fast_scalarmult (largeint m, ec_point P)
   {
       ec_point tmpP, retP;
5
       retP = 0_neutre;
       tmpP = P;
       while (m != 0)

```

```

    {
        if(error)                //On a trouvé un facteur non inversible !
10         return 0_neutre;
        if(getbit(m, 0))         //Le bit de poids le plus faible est il non nul ?
            retP = retP + tmpP;
        m >> 1;                 //Decalage de 1 bit vers la gauche
15         tmpP = 2 * tmpP;
    }

    return retP;
}

```

1.3 Calcul des inverses

Lors du calcul de λ , l'inverse de $(x_2 - x_1)$ ou de $2y_1$ modulo n est calculé à l'aide de l'algorithme d'Euclide étendu qui permet de calculer les coefficients de Bezout. Il est basé sur l'identité fondamentale de l'algorithme d'Euclide suivante :

$$d = a \wedge b = b \wedge (a \bmod b) \quad (1)$$

Démonstration. Montrons $(bq + r) \wedge b = b \wedge r$. Soit $d = (bq + r) \wedge b$ et $d' = b \wedge r$. On a : $d|bq + r$ et $d|b$ donc $d|r$ d'où $d|r = (bq + r) - bq$ donc $d|d'$. De plus : $d'|b$ donc $d'|bq$ et $d'|r$ donc $d'|bq + r$ d'où $d'|d$. Finalement, on a bien $d = d'$. □

Ceci implique que si $d = bx + (a \bmod b)y'$ alors $d = ax + by$ avec $x = y'$ et $y = x' - \lfloor a/b \rfloor y'$.

Démonstration. On a en effet d'après le théorème de Bezout et l'égalité (1) :

$$\begin{aligned}
 d &= bx' + (a \bmod b)y' \\
 &= bx' + (a - \lfloor a/b \rfloor b)y' \\
 &= ay' + b(x' - \lfloor a/b \rfloor y')
 \end{aligned}$$
□

On a de plus la condition d'arrêt de l'algorithme en posant

$$d = a \wedge 0 = a \wedge 1 = a$$

```

1  struct couple
   {
       largeint x;
       largeint y;
5  };

   couple Bezout(largeint a, largeint n)
   {
10      couple cRet;
       largeint tmp;

       if (b == 0)

```

```

    {
        cRet.x = 1;
15      cRet.y = 0;
        return cRet;
    }
    cRet = Bezout(b, mod(a, b));
    tmp = cRet.y;
20    cRet.y = cRet.x - (a / b) * cRet.y;
    cRet.x = tmp;
    return cRet;
}

25 largeint invModN(largeint a, largeint n)
{
    return mod((Bezout(a, n)).x, n);
}

```

1.4 Le $\text{ppcm}(1, 2, \dots, B)$

Pour des raisons de rapidité, dans la pratique, on ne prend pas $k = \text{PPCM}(1, 2, \dots, B)$ mais $k = \text{PPCM}(B, B - 1)$:

```

1  largeint calc_k(largeint B)
  {
      return lcm(B, B - 1);
  }

```

1.5 La procédure principale

On va travailler sur les courbes elliptiques $E : y^2 \equiv x^3 + ax + b \pmod{n} \cup \mathcal{O}$. Le coefficient a étant choisis (c'est notre paramètre), on impose b tel que $P = (1, 1)$ appartienne à $E : b = 1^2 - 1^3 - a \times 1 = -a$. On calcule alors kP pour nb_essais valeurs de a , puis on incrémente B et on recommence jusqu'à ce que le facteur de λ ne soit pas inversible lors du calcul de kP ce qui est signalé par la valeur `true` de la variable booléenne `error`. La variable `error_factor` contient alors la valeur du nombre qui n'a pas été inversible lors du calcul de λ .

```

1  #define B_incr      1
   #define a_incr     30
   #define nb_essais  40
   #define x_init     1
5  #define y_init     1

   largeint TrouveFacteur(largeint n, largeint B)
   {
       ec_point P, Q;
10      largeint k, tmp;
       int a_essais;
       bool nouvelle_courbe=false;

       error = false;
15      if ((n % 2) == 0)
       {

```

```

//Le facteur 2 a ete trouve
//...
return largeint(2);
20 }
if ((n % 3) == 0)
{
//Le facteur 3 a ete trouve
//...
25 return largeint(3);
}

p = n;
k = calc_k(B);
30 b = 1;
a = 1; a_essais = 0;

P.x = x_init; P.y = y_init;
while (1)
35 {
b = (P.y * P.y) - (P.x * P.x * P.x) - (a * P.x);

//PGCD(Delta, n) == 1 ?
tmp = gcd(4 * a * a * a + 27 * b * b, n);
40 while (tmp != 1)
{
if (tmp == n)
{
45 a += a_incr;
b = (P.y * P.y) - (P.x * P.x * P.x) - (a * P.x);
tmp = gcd(4 * a * a * a + 27 * b * b, n);
}
else
return tmp;
50 }
if (a_essais > nb_essais)
nouvelle_courbe = true;
Q = k * P;
//Lambda n'etait-il pas inversible ?
55 if (error)
{
largeint g = gcd(error_factor, n);

if(1 < g && g < n)
60 {
//Impression des resultats
//...
return g;
}
65 else
nouvelle_courbe = true;
}
//On change de courbe
if (nouvelle_courbe)
70 {

```

```

75         a = 1;
           a_essais = 0;
           B += B_incr;
           while (k == calc_k(B)) {B += B_incr;};
           k = calc_k(B);
           }
           else
           //On incremente a
           {
80             a += a_incr;
               a_essais++;
           }
       }
75
85     //Cette commande n'est jamais executee
       return largeint(1);
   }

```

2 Quelques résultats

2.1 $109849677793909 = 239633 \cdot 11131 \cdot 41183$

Nombre a factoriser : 109849677793909
 B-lissite : 10487395

17:58:58

17:58:58

facteur : 11131
 a : 151
 b : -151
 p : 109849677793909
 B : 10487395
 k : 109985443398630

17:58:59

facteur : 41183
 a : 571
 b : -571
 p : 9868805839
 B : 10487395
 k : 109985443398630

2.2 $134755010254579987971511 = 61494437 \cdot 42398497 \cdot 51684299$

Nombre a factoriser : 134755010254579987971511
 B-lissite : 367091132971

18:53:46

19:03:32

facteur : 42398497
 a : 1

b : -1
p : 134755010254579987971511
B : 367091143560
k : 134755907679821438330040