Comptage de points d'une courbe elliptique sur des corps finis

Daniel Resende
supervisé par Luca De Feo

28 février 2017

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Pourquoi compter les points d'une courbe elliptique ?

Les courbes elliptiques définissent une loi de groupe sur les corps finis $\mathbf{F}_q$ qui est difficile pour le problème du logarithme discret. On retrouve par conséquent son utilisation dans plusieurs schémas cryptographiques comme Diffie-Hellman (avec ECDH) ou El-Gamal (avec ECDSA). Cependant, l'utilisation de schémas à l'aide de courbes elliptiques nécessite d'avoir un grand nombre premier qui divise l'ordre d'un sous-groupe cyclique de $E(\mathbf{F}_q)$. Nous avons donc besoin de connaitre le cardinal de $E(\mathbf{F}_q)$.

Figure – Portrait René Schoof

Il existe aujourd'hui de nombres algorithme de comptage de points d'une courbes elliptiques sur un corps finis $\mathbf{F}_q$:

- L'algorithme de Shank en 1971 basé sur Baby Step Giant Step et le théorème de Hasse,

Il existe aujourd'hui de nombres algorithme de comptage de points d'une courbes elliptiques sur un corps finis $\mathbf{F}_q$:

- L'algorithme de Shank en 1971 basé sur Baby Step Giant Step et le théorème de Hasse,
- L'algorithme de Schoof en 1985 que l'on va étudier dans ce mémoire,

Il existe aujourd'hui de nombres algorithme de comptage de points d'une courbes elliptiques sur un corps finis $\mathbf{F}_q$:

- L'algorithme de Shank en 1971 basé sur Baby Step Giant Step et le théorème de Hasse,
- L'algorithme de Schoof en 1985 que l'on va étudier dans ce mémoire,
- L'algorithme SEA [Schoof, Elkies, Atkin] en 1995 qui est une amélioration de l'algorithme de Schoof,

Il existe aujourd'hui de nombres algorithme de comptage de points d'une courbes elliptiques sur un corps finis $\mathbf{F}_q$:

- L'algorithme de Shank en 1971 basé sur Baby Step Giant Step et le théorème de Hasse,
- L'algorithme de Schoof en 1985 que l'on va étudier dans ce mémoire,
- L'algorithme SEA [Schoof, Elkies, Atkin] en 1995 qui est une amélioration de l'algorithme de Schoof,
- L'algorithme de Satoh en 2005 basé sur le relèvement canonique sur les $\mathbf{Z}$ q-adiques,

Il existe aujourd'hui de nombres algorithme de comptage de points d'une courbes elliptiques sur un corps finis $\mathbf{F}_q$:

- L'algorithme de Shank en 1971 basé sur Baby Step Giant Step et le théorème de Hasse,
- L'algorithme de Schoof en 1985 que l'on va étudier dans ce mémoire,
- L'algorithme SEA [Schoof, Elkies, Atkin] en 1995 qui est une amélioration de l'algorithme de Schoof,
- L'algorithme de Satoh en 2005 basé sur le relèvement canonique sur les $\mathbf{Z}$ q-adiques,
- L'algorithme AGM [Mestre] basé sur le calcul de suites arithmetico-géométriques.

Sommaire

Soit $\mathbf{F}_q$ un corps fini à $q = p^n$ éléments de caractéristiques $p \neq 2, 3$.

Définition

Soit E une courbe elliptique définie sur $\mathbf{F}_q$. On obtient l'équation affine de Weierstraß :

$$y^2 = x^3 + ax + b \tag{1}$$

avec $a, b \in \mathbf{F}_q$ et $\Delta = -16(4a^3 + 27b^2) \neq 0$.

Définition

Soit ϕ l'endomorphisme de Frobénius d'une courbe elliptique E tel que

$$\begin{array}{rccc} \Phi : & E(\bar{\mathbf{F}}_q) & \longrightarrow & E(\bar{\mathbf{F}}_q) \\ & (x, y) & \longmapsto & (x^q, y^q). \end{array}$$

Définition

Soit ϕ l'endomorphisme de Frobénius d'une courbe elliptique E tel que

$$\begin{array}{rccc} \Phi : & E(\bar{\mathbf{F}}_q) & \longrightarrow & E(\bar{\mathbf{F}}_q) \\ & (x, y) & \longmapsto & (x^q, y^q). \end{array}$$

Définition (Trace)

Soit E une courbe elliptique sur $\mathbf{F}_q$. La trace de $E(\mathbf{F}_q)$ est l'entier $t \in \mathbf{Z}^*$ tel que

$$t = q + 1 - \#E(\mathbf{F}_q). \tag{2}$$

Définition

Soit ϕ l'endomorphisme de Frobénius d'une courbe elliptique E tel que

$$\begin{array}{cccc} \Phi : & E(\bar{\mathbf{F}}_q) & \longrightarrow & E(\bar{\mathbf{F}}_q) \\ & (x, y) & \longmapsto & (x^q, y^q). \end{array}$$

Définition (Trace)

Soit E une courbe elliptique sur $\mathbf{F}_q$. La trace de $E(\mathbf{F}_q)$ est l'entier $t \in \mathbf{Z}^*$ tel que

$$t = q + 1 - \#E(\mathbf{F}_q). \tag{2}$$

Proposition

Soit la trace t de $E(\mathbf{F}_q)$, on a alors

$$\phi^2 - t\phi + q = 0 \tag{3}$$

Théorème (de Hasse)

Soit E une courbe elliptique sur $\mathbf{F}_q$ et la trace t de $E(\mathbf{F}_q)$. On a

$$| \, t \, | \leq 2\sqrt{q}, \tag{4}$$

et par conséquent

$$| \, \#E(\mathbf{F}_q) - (q+1) \, | \leq 2\sqrt{q} \tag{5}$$

On va maintenant nous concentrer sur les sous-groupes de n-torsions $E[n]$ avec $n \in \mathbf{Z}_{\geq -1}$ tel que $p \nmid n$. Et on introduit la notion de polynôme de division.

Définition (Polynôme de division)

Soit $n \in \mathbb{Z}^*$, le polynôme de division ψ_n est la fonction polynôme de $K[E]$ de coefficient dominent n et de diviseur

$$div(\psi_n) = (E[n]) - n^2(\vartheta)$$

Proposition (Caractérisation du polynôme de division)

On construit le polynôme de division par récurrence sur $n \in \mathbf{Z}_{\geq 1}$:

a) $\psi_{-1}(X, Y) = -1,\ \psi_0(X, Y) = 0,\ \psi_1(X, Y) = 1,\ \psi_2(X, Y) = 2Y,$

b) $\psi_3(X, Y) = 3X^4 + 6aX^2 + 12bX - a^2,$

c) $\psi_4(X, Y) = 4Y(X^6 + 5aX^4 + 20bX^3 - 5a^2X^2 - 4bX - 8b^2 - a^3,$

d) $\psi_{2n}(X, Y) = \psi_n(\psi_{n+2}\psi_{n-1}^2 - \psi_{n-2}\psi_{n+1}^2)/2Y,$

e) $\psi_{2n+1}(X, Y) = \psi_{n+2}\psi_n^3 - \psi_{n+1}^3\psi_{n-1},$

f) $\psi_{-n} = \psi_n.$

Dans l'algorithme de Schoof, on utilisera une variante du polynôme de division.

Définition

Soit $n \in \mathbb{Z}^*$, le polynôme $f_n \in K[E]$ définie par la relation suivante :

$$f(n) = \begin{cases} \bar{\psi}_n(X, Y) & \text{si } n \text{ est pair} \\ \bar{\psi}_n(X, Y)/Y & \text{si } n \text{ est impair} \end{cases}$$

où $\bar{\psi}_n$ est la réduction de ψ_n par les termes en Y^2 par l'équation (E).

Proposition (Caractérisation de f_n)

On construit f_n par récurrence sur $n \in \mathbf{Z}_{\geq 1}$:

a) $f_{-1}(X) = -1,\ f_0(X) = 0,\ f_1(X) = 1,\ f_2(X) = 2,$

b) $f_3(X) = 3X^4 + 6aX^2 + 12bX - a^2,$

c) $f_4(X) = 4(X^6 + 5aX^4 + 20bX^3 - 5a^2X^2 - 4bX - 8b^2 - a^3,$

d) $f_{2n}(X) = f_n(f_{n+2}f_{n-1}^2 - f_{n-2}f_{n+1}^2),$

e) $f_{2n+1}(X) = \begin{cases} f_{n+2}f_n^3(x^2+ax+b) - f_{n+1}^3 f_{n-1} & \text{si } n \text{ est pair} \\ f_{n+2}f_n^3 - f_{n+1}^3 f_{n-1}(x^2+ax+b) & \text{si } n \text{ est impair} \end{cases}$

Proposition

Soit $P = (x, y) \in E(\bar{\mathbf{F}}_q)$ avec $P \notin E[2]$ et $n \in \mathbf{Z}_{\geq -1}$. Alors

$$nP = \vartheta \iff f_n = 0 \tag{6}$$

Proposition

Soit $P = (x, y) \in E(\bar{\mathbf{F}}_q)$ avec $P \notin E[2]$ et $n \in \mathbf{Z}_{\geq -1}$. Alors

$$nP = \vartheta \iff f_n = 0 \tag{6}$$

Proposition

Soit $P = (x, y) \in E(\bar{\mathbf{F}}_q)$ et $n \in \mathbf{Z}_{\geq 1}$. Alors

$$nP = \left(x - \frac{\psi_{n-1}\psi_{n+1}}{\psi_n^2}, \frac{\psi_{n+2}\psi_{n-1}^2 - \psi_{n-2}\psi_{n+1}^2}{4Y\psi_n^3} \right) \tag{7}$$

On définit l'application

$$\begin{array}{ccc} End_{\mathbf{F}_q}(E) & \longrightarrow & End_{Gal(\bar{\mathbf{F}_q}/\mathbf{F}_q)}(E[l]) \\ \phi & \longmapsto & \phi_l \end{array}$$

avec l premier.

On définit l'application

$$\begin{array}{ccc} End_{\mathbf{F}_q}(E) & \longrightarrow & End_{Gal(\bar{\mathbf{F}_q}/\mathbf{F}_q)}(E[l]) \\ \phi & \longmapsto & \phi_l \end{array}$$

avec l premier.
On obtient l'équation (3) par l'application précédente et l'équation :

$$\phi_l^2 - t\phi_l + q = 0 \tag{8}$$

Sommaire

Sommaire

Cet algorithme consiste à calculer la trace du frobénius modulo tous les $l < l_{max}$ tel que l_{max} soit le plus petit nombre premier vérifiant :

$$\prod_{l \text{ premier},\ p \nmid l}^{l_{max}} l > 4\sqrt{q}. \tag{9}$$

Une fois calculé la trace modulo toutes les l-torsions, on utilise le Théorème des Restes Chinois (CRT) pour obtenir la trace dans $\mathbf{F}_q$.

Algorithme 1 Algorithme de Shoof

Require: Une courbe elliptique E sur $\mathbf{F}_q$ un polynôme quelconque.
Ensure: Le cardinal de $E(\mathbf{F}_q)$.

```
M ← 2, l ← 3 ;
S ← {(t mod 2, 2)} ; {Cas pour l = 2}
while M < 4√q do
  k ← q mod l ;
  for τ = 0 to (l−1)/2 do
    if ∀P ∈ E[l], φ²(P) + [k]P = ±[τ]φ(P) then
      S ← S ∪ {(τ, l)} or S ← S ∪ {(−τ, l)} {Selon les cas}
      break ;
    end if
  end for
  M ← M ∗ l ;
  l ← nextprime(l) ; {Donne le prochain nombre premier après l}
end while
∀t ∈ S, trace ← CRT(t) ; {Effectue le théorème des restes chinois}
return q + 1 − trace.
```

On regarde cet algorithme plus en détails sur le calcul de la trace *mod l*.

On regarde cet algorithme plus en détails sur le calcul de la trace $mod\ l$.
Dans le cas $l = 2$, on cherche les points de 2-torsions,

$$t = 1\ mod\ 2 \Leftrightarrow pgcd(x^3 + ax + b, x^q - x) = 1 \tag{10}$$

On regarde cet algorithme plus en détails sur le calcul de la trace *mod l*.
Dans le cas $l = 2$, on cherche les points de 2-torsions,

$$t = 1 \; mod \; 2 \Leftrightarrow pgcd(x^3 + ax + b, x^q - x) = 1 \tag{10}$$

Dans le cas général, on pose $k = q \; mod \; l$ et on recherche un $\tau \; mod \; l$ qui vérifie (8), *i.e.*

$$\phi_l^2(P) - \tau\phi_l(P) + kP = \vartheta \Longleftrightarrow \phi_l^2(P) + kP = \tau\phi_l(P) \tag{11}$$

On regarde cet algorithme plus en détails sur le calcul de la trace *mod l*.
Dans le cas $l = 2$, on cherche les points de 2-torsions,

$$t = 1 \ mod \ 2 \Leftrightarrow pgcd(x^3 + ax + b, x^q - x) = 1 \tag{10}$$

Dans le cas général, on pose $k = q \ mod \ l$ et on recherche un $\tau \ mod \ l$ qui vérifie (8), *i.e.*

$$\phi_l^2(P) - \tau\phi_l(P) + kP = \vartheta \Longleftrightarrow \phi_l^2(P) + kP = \tau\phi_l(P) \tag{11}$$

On découpe ensuite l'équation (11) en deux et on applique (7) pour obtenir d'une part

$$\tau\phi_l(P) = \left(x^q - \left(\frac{\psi_{\tau-1}\psi_{\tau+1}}{\psi_\tau^2}\right)^q, \left(\frac{\psi_{\tau+2}\psi_{\tau-1}^2 - \psi_{\tau-2}\psi_{\tau+1}^2}{4Y\psi_\tau^3}\right)^q\right), \tag{12}$$

et d'autre part

$$\phi_l^2(P) + kP = \left(-x^{q^2} - x + \frac{\psi_{k-1}\psi_{k+1}}{\psi_k^2} + \lambda^2, \right.$$
$$\left. - y^{q^2} - \lambda\left(-x^{q^2} - x + \frac{\psi_{k-1}\psi_{k+1}}{\psi_k^2}\right)\right) \quad (13)$$

avec

$$\lambda = \frac{\psi_{k+2}\psi_{k-1}^2 - \psi_{k-2}\psi_{k+1}^2 - 4y^{q^2+1}\psi_k^3}{4\psi_k y((x - x^{q^2})\psi_k^2 - \psi_{k-1}\psi_{k+1})}. \quad (14)$$

On veux donc tester si les deux parties sont égale modulo l, et par conséquent tester que les abscisses puis les ordonnées sont égales.

Pour les abscisses, on doit vérifier que

$$((\psi_{k-1}\psi_{k+1}-\psi_k(x^{q^2}+x^q+x))\beta^2+\psi_k^2\alpha^2)\psi_\tau^{2q}+\psi_{\tau-1}^q\psi_{\tau+1}^q\beta^2\psi_k^2 = 0 \; mod \; \phi_l. \tag{15}$$

Si l'assertion est vrai, alors on vérifie aussi pour les ordonnées que

$$4y^q\psi_\tau^{3q}(\alpha((2x^{q^2}+x)\psi_k^2-\psi_{k-1}\psi_{k+1})-y^{q^2}\beta\psi_k^2)-\beta\psi_k^2(\psi_{\tau+2}\psi_{\tau-1}^2 - \psi_{\tau-2}\psi_{\tau+1}^2)^q = 0 \; mod \; \phi_l. \tag{16}$$

Où

$$\begin{cases} \alpha = \psi_{k+2}\psi_{k-1}^2 - \psi_{k-1}\psi_{k+1}^2 - 4y^{q^2+1}\psi_{k-1}^3 \\ \beta = ((x-x^{q^2})\psi_k^2 - \psi_{k-1}\psi_{k+1})4y\psi_k \end{cases}$$

Pour les abscisses, on doit vérifier que

$$((\psi_{k-1}\psi_{k+1}-\psi_k(x^{q^2}+x^q+x))\beta^2+\psi_k^2\alpha^2)\psi_\tau^{2q}+\psi_{\tau-1}^q\psi_{\tau+1}^q\beta^2\psi_k^2 = 0 \; mod \; \phi_l. \tag{15}$$

Si l'assertion est vrai, alors on vérifie aussi pour les ordonnées que

$$4y^q\psi_\tau^{3q}(\alpha((2x^{q^2}+x)\psi_k^2-\psi_{k-1}\psi_{k+1})-y^{q^2}\beta\psi_k^2)-\beta\psi_k^2(\psi_{\tau+2}\psi_{\tau-1}^2 - \psi_{\tau-2}\psi_{\tau+1}^2)^q = 0 \; mod \; \phi_l. \tag{16}$$

Où

$$\begin{cases} \alpha = \psi_{k+2}\psi_{k-1}^2 - \psi_{k-1}\psi_{k+1}^2 - 4y^{q^2+1}\psi_{k-1}^3 \\ \beta = ((x-x^{q^2})\psi_k^2 - \psi_{k-1}\psi_{k+1})4y\psi_k \end{cases}$$

Remarque

- Pour résumer, si un τ vérifie que (15) alors $t = -\tau$ modulo l, sinon si un τ vérifie que (15) et (16), alors $t = \tau$ modulo l. Si aucun τ vérifie les deux relations, alors la trace est nulle modulo l.
- En pratique, dans les équations (15) et (16), on remplace les ψ_n par des f_n. Puis si nécessaire on réduira modulo (1) et on divisera par y.

Sommaire

Algorithme 2 Algorithme de Shoof amélioré

Require: Une courbe elliptique E sur $\mathbf{F}_q$ un polynôme quelconque.
Ensure: Le cardinal de $E(\mathbf{F}_p)$.

```
M ← 2, l ← 3 ;
S ← {(t mod 2, 2)} ; {Cas pour l = 2}
while M < 4√q do
    k ← q mod l ;
    if φ_l² P = ±kP then
        if (k/l) = −1 then
            S ← S ∪ {(0, l)}
        else
            on recherche w tel que k = w² mod l
            if ±w est une valeur propre de φ_l then
                S ← S ∪ {(w, l)} or S ← S ∪ {(−w, l)} {Selon les cas}
            else
                S ← S ∪ {(0, l)}
            end if
        end if
    else
        for τ = 1 to (l−1)/2 do
            if ∀P ∈ E[l], φ²(P) + [k]P = ±[τ]φ(P) then
                S ← S ∪ {(τ, l)} or S ← S ∪ {(−τ, l)} {Selon les cas}
                break ;
            end if
        end for
    end if
    M ← M ∗ l ;
    l ← nextprime(l) ; {Donne le prochain nombre premier après l}
end while
∀t ∈ S, trace ← CRT(t) ; {Effectue le théorème des restes chinois}
return q + 1 − trace.
```

On regarde si $\forall P$ nonzéro

$$\phi_l^2 P = \pm kP \tag{17}$$

avec $q \equiv k[l]$, sinon on fait le cas général.

Pour ce faire, on va donc transformer l'équation (17) en un calcul de *pgcd* pour pouvoir l'implémenter sur machine. On obtient le résultat suivant :

Pour ce faire, on va donc transformer l'équation (17) en un calcul de *pgcd* pour pouvoir l'implémenter sur machine. On obtient le résultat suivant :

$$\begin{cases} (x^{q^2} - x) f_k^2 (x^3 + ax + b) + f_{k-1} f_{k+1} = 0 & \text{si } k \text{ pair} \\ (x^{q^2} - x) f_k^2 + f_{k-1} f_{k+1} (x^3 + ax + b) = 0 & \text{si } k \text{ impair} \end{cases} \tag{18}$$

Pour ce faire, on va donc transformer l'équation (17) en un calcul de *pgcd* pour pouvoir l'implémenter sur machine. On obtient le résultat suivant :

$$\begin{cases} (x^{q^2} - x)f_k^2(x^3 + ax + b) + f_{k-1}f_{k+1} = 0 & \text{si } k \text{ pair} \\ (x^{q^2} - x)f_k^2 + f_{k-1}f_{k+1}(x^3 + ax + b) = 0 & \text{si } k \text{ impair} \end{cases} \tag{18}$$

Ce qui est équivalent à

$$\begin{cases} pgcd((x^{q^2} - x)f_k^2(x^3 + ax + b) + f_{k-1}f_{k+1}, f_l) & \text{si } k \text{ pair} \\ pgcd((x^{q^2} - x)f_k^2 + f_{k-1}f_{k+1}(x^3 + ax + b), f_l) & \text{si } k \text{ impair} \end{cases} \tag{19}$$

Pour ce faire, on va donc transformer l'équation (17) en un calcul de *pgcd* pour pouvoir l'implémenter sur machine. On obtient le résultat suivant :

$$\begin{cases} (x^{q^2} - x)f_k^2(x^3 + ax + b) + f_{k-1}f_{k+1} = 0 & \text{si } k \text{ pair} \\ (x^{q^2} - x)f_k^2 + f_{k-1}f_{k+1}(x^3 + ax + b) = 0 & \text{si } k \text{ impair} \end{cases} \tag{18}$$

Ce qui est équivalent à

$$\begin{cases} pgcd((x^{q^2} - x)f_k^2(x^3 + ax + b) + f_{k-1}f_{k+1}, f_l) & \text{si } k \text{ pair} \\ pgcd((x^{q^2} - x)f_k^2 + f_{k-1}f_{k+1}(x^3 + ax + b), f_l) & \text{si } k \text{ impair} \end{cases} \tag{19}$$

Donc $pgcd \neq 1$ si et seulement si $\phi_l^2 P = \pm kP$.

On peux maintenant chercher le τ tel que $\tau^2 \equiv 4q[l]$. On a alors que si q n'est pas un carré modulo l alors $\tau = 0$ mod l, sinon on note w la racine carré de q modulo l. On regarde ensuite si $\pm w$ est une valeur propre de ϕ_l. On calcule le *pgcd* suivant :

On peux maintenant chercher le τ tel que $\tau^2 \equiv 4q[l]$. On a alors que si q n'est pas un carré modulo l alors $\tau = 0 \ mod \ l$, sinon on note w la racine carré de q modulo l. On regarde ensuite si $\pm w$ est une valeur propre de ϕ_l. On calcule le *pgcd* suivant :

$$\begin{cases} pgcd((x^q - x)f_w^2(x^3 + ax + b) + f_{w-1}f_{w+1}, f_l) & \text{si } w \text{ pair} \\ pgcd((x^q - x)f_w^2 + f_{w-1}f_{w+1}(x^3 + ax + b), f_l) & \text{si } w \text{ impair} \end{cases} \tag{20}$$

On peux maintenant chercher le τ tel que $\tau^2 \equiv 4q[l]$. On a alors que si q n'est pas un carré modulo l alors $\tau = 0 \ mod \ l$, sinon on note w la racine carré de q modulo l. On regarde ensuite si $\pm w$ est une valeur propre de ϕ_l. On calcule le *pgcd* suivant :

$$\begin{cases} pgcd((x^q - x)f_w^2(x^3 + ax + b) + f_{w-1}f_{w+1}, f_l) & \text{si } w \text{ pair} \\ pgcd((x^q - x)f_w^2 + f_{w-1}f_{w+1}(x^3 + ax + b), f_l) & \text{si } w \text{ impair} \end{cases} \tag{20}$$

Donc si $pgcd = 1$ alors $\tau = 0 \ mod \ l$, sinon on doit déterminer de signe de w.

On peux maintenant chercher le τ tel que $\tau^2 \equiv 4q[l]$. On a alors que si q n'est pas un carré modulo l alors $\tau = 0 \ mod \ l$, sinon on note w la racine carré de q modulo l. On regarde ensuite si $\pm w$ est une valeur propre de ϕ_l. On calcule le *pgcd* suivant :

$$\begin{cases} pgcd((x^q - x)f_w^2(x^3 + ax + b) + f_{w-1}f_{w+1}, f_l) & \text{si } w \text{ pair} \\ pgcd((x^q - x)f_w^2 + f_{w-1}f_{w+1}(x^3 + ax + b), f_l) & \text{si } w \text{ impair} \end{cases} \tag{20}$$

Donc si $pgcd = 1$ alors $\tau = 0 \ mod \ l$, sinon on doit déterminer de signe de w.

$$\begin{cases} pgcd(4(x^3 + ax + b)^{(q-1)/2} f_w^3 - f_{w+2}^2 f_{w-1} + f_{w-2}^2 f_{w-1}, f_l) & \text{si } w \text{ pair} \\ pgcd(4(x^3 + ax + b)^{(q-1)/2} f_w^3 - f_{w+2}^2 f_{w-1} + f_{w-2}^2 f_{w-1}, f_l) & \text{si } w \text{ impair} \end{cases} \tag{21}$$

On peux maintenant chercher le τ tel que $\tau^2 \equiv 4q[l]$. On a alors que si q n'est pas un carré modulo l alors $\tau = 0 \ mod \ l$, sinon on note w la racine carré de q modulo l. On regarde ensuite si $\pm w$ est une valeur propre de ϕ_l. On calcule le *pgcd* suivant :

$$\begin{cases} pgcd((x^q - x)f_w^2(x^3 + ax + b) + f_{w-1}f_{w+1}, f_l) & \text{si } w \text{ pair} \\ pgcd((x^q - x)f_w^2 + f_{w-1}f_{w+1}(x^3 + ax + b), f_l) & \text{si } w \text{ impair} \end{cases} \tag{20}$$

Donc si $pgcd = 1$ alors $\tau = 0 \ mod \ l$, sinon on doit déterminer de signe de w.

$$\begin{cases} pgcd(4(x^3 + ax + b)^{(q-1)/2} f_w^3 - f_{w+2}^2 f_{w-1} + f_{w-2}^2 f_{w-1}, f_l) & \text{si } w \text{ pair} \\ pgcd(4(x^3 + ax + b)^{(q-1)/2} f_w^3 - f_{w+2}^2 f_{w-1} + f_{w-2}^2 f_{w-1}, f_l) & \text{si } w \text{ impair} \end{cases} \tag{21}$$

Et par conséquent, si $pgcd = 1$ alors $\tau = -2w \ mod \ l$, sinon $\tau = 2w \ mod \ l$.

Sommaire

Sommaire

Théorème (des nombres premiers)

Soit $\pi(x)$ le nombre de premier plus petit que x. On a alors que

$$\pi(x) \rightsquigarrow \frac{x}{log(x)}$$
$$x \rightsquigarrow +\infty$$

Théorème (des nombres premiers)

Soit $\pi(x)$ le nombre de premier plus petit que x. On a alors que

$$\pi(x) \rightsquigarrow \frac{x}{log(x)}$$
$$x \rightsquigarrow +\infty$$

Remarque

Le théorème précédent donne aussi la relation suivante :

$$\prod_{l \ premier, \ p \nmid l}^{l_{max}} l \rightsquigarrow e^{lmax}$$

On en déduit que $e^{lmax} > 4\sqrt{q} \Rightarrow lmax = O(log(q))$. Et par conséquent, on a $O\left(\frac{log(q)}{log(log(q))}\right)$, *i.e.*$O(log(q))$ torsion à calculer.

On en déduit que $e^{lmax} > 4\sqrt{q} \Rightarrow lmax = O(log(q))$. Et par conséquent, on a $O\left(\frac{log(q)}{log(log(q))}\right)$, *i.e.*$O(log(q))$ torsion à calculer.

Ensuite on calcule la complexité de la création du tableau de polynôme de division. On doit pour ce faire calculer $lmax = O(log(q))$ polynômes avec 9 (cas pair) ou 11 (cas impair) opérations élémentaires à chaque tour. Le coût total de la création du tableau est de $O(log(q))$.

On en déduit que $e^{lmax} > 4\sqrt{q} \Rightarrow lmax = O(log(q))$. Et par conséquent, on a $O\left(\frac{log(q)}{log(log(q))}\right)$, *i.e.*$O(log(q))$ torsion à calculer.

Ensuite on calcule la complexité de la création du tableau de polynôme de division. On doit pour ce faire calculer $lmax = O(log(q))$ polynômes avec 9 (cas pair) ou 11 (cas impair) opérations élémentaires à chaque tour. Le coût total de la création du tableau est de $O(log(q))$.

On aussi que $deg(f_l) = O(l^2) = O(log^2(q))$ et que le *pgcd* à une complexité de $O(deg(f_l)^2 log^3(q))$. D'où une complexité pour chaque tour $O(log^7(q))$.

On en déduit que $e^{lmax} > 4\sqrt{q} \Rightarrow lmax = O(log(q))$. Et par conséquent, on a $O\left(\frac{log(q)}{log(log(q))}\right)$, *i.e.*$O(log(q))$ torsion à calculer.

Ensuite on calcule la complexité de la création du tableau de polynôme de division. On doit pour ce faire calculer $lmax = O(log(q))$ polynômes avec 9 (cas pair) ou 11 (cas impair) opérations élémentaires à chaque tour. Le coût total de la création du tableau est de $O(log(q))$.

On aussi que $deg(f_l) = O(l^2) = O(log^2(q))$ et que le *pgcd* à une complexité de $O(deg(f_l)^2 log^3(q))$. D'où une complexité pour chaque tour $O(log^7(q))$.

De plus, comme on fait $O(l) = O(log(q))$ tour, on arrive à une complexité à $O(log^8(q))$. Donc on arrive à une complexité de $O(log^9(q))$ Enfin la complexité de CRT est négligeable par rapport à $O(log^9(q))$

Sommaire

Voici une petite comparaison entre les différents algorithmes :

- L'algorithme Schoof est en $O(log^9(q))$,

Voici une petite comparaison entre les différents algorithmes :

- L'algorithme Schoof est en $O(log^9(q))$,
- L'algorithme SEA est en $O(log^4(q))$,

Voici une petite comparaison entre les différents algorithmes :

- L'algorithme Schoof est en $O(log^9(q))$,
- L'algorithme SEA est en $O(log^4(q))$,
- L'algorithme de Satoh est en $O(n^3)$.

Sommaire

Pour rappel, ce programme est écrit en langage `C` et j'utilise la librairie `FLINT` afin de manipuler des polynômes dans $\mathbf{F}_q$. J'ai choisi de décomposer mon programme en trois parties :

La fonction `main`

La fonction `main` qui récupère les arguments auprès de l'utilisateur, et qui vérifie que les arguments donne une courbe elliptique sur $\mathbf{F}_q$. Elle se finie en affichant le cardinal de la courbe elliptique sur $\mathbf{F}_q$.

La fonction division_polynomial

La fonction division_polynomial remplie un tableau avec tous les polynômes de division.

```
void division_polynomial(fq_poly_t *tab, fq_t a,
fq_t b, fq_poly_t ecc, ulong k, fq_ctx_t fq)
ENTRÉE :
     Tableau de Fq-polynôme tab et k la taille du tableau
     Entiers a,b tels que E: y² = x³ + ax + b une courbe
     elliptique sur Fq
     Fq-polynôme eec représentant la courbe elliptique
     Corps fini fq à q éléments
SORTIE :
     Tableau de Fq-polynôme tab rempli de k polynôme de
     division
```

La fonction schoof

La fonction `schoof` crée un tableau de *lmax* polynômes de division (remplie par la fonction `division_polynomial`), puis elle exécute l'algorithme de `schoof`. Et renvoie le cardinal de la courbe elliptique.

```
void schoof(fmpz_t card, fq_t a, fq_t b, fmpz_t q,fq_ctx_t fq)
ENTRÉE :
     Entier q premier tel que Fq un corps fini à q éléments
     Entiers a,b tels que E: y² = x³ + ax + b une courbe
     elliptique sur Fq
SORTIE :
     Entier card tel que card = #E(Fq)
```

La fonction fmpz_nextprime

Par ailleurs, j'ai implémenté une fonction `fmpz_nextprime` qui renvoi le prochain nombre premier. En effet, cette fonction n'est pas inclut dans la librairie `FLINT`. Il est possible d'optimiser cette fonction, cependant dans notre cas les nombres premiers tester sont de l'ordre de $O(log(q))$.

```
ENTRÉE :
    Entier op tel que op > 2
SORTIE :
     Entier rop

void fmpz_nextprime(fmpz_t rop, fmpz_t op)
{
    fmpz_add_ui(rop, op, 2);
    while(!fmpz_is_prime(rop)) fmpz_add_ui(rop, rop, 2);
}
```

Sommaire

Exemple (Soit $p = 101$ et $E : y^2 = x^3 + 3x + 4$)

On a $l_{max} = 7$ et $l = 2, 3, 5, 7$.
On obtient $t = 0[2], t = 1[3], t = 0[5], t = 3[7]$ à chaque étape.
D'où $\#E(\mathbf{F}_{101}) = 92$.

Test du programme

Questions ?

Merci.