Algorithme de Schoof

Gabriel Abauzit

Résumé

Ce rapport présente l'algorithme de Schoof, premier algorithme polynomial pour calculer le nombre de points sur une courbe elliptique définie sur un corps fini, ainsi que son implémentation en langage C, disponible à l'adresse `https://github.com/gabauzit/Algorithme-de-Schoof`**.**

Table des matières

$\mathbb{F}_q$ désignera un corps fini de caractéristique $p \notin \{2, 3\}$.

1 Introduction

1.1 Architecture du projet

Le répertoire du projet est composé des dossiers suivants :

- `bin/` : Contient l'exécutable des tests.
- `include/` : Contient les fichiers d'en-tête `.h`.
- `obj/` : Contient les fichiers objets `.o`.
- `results/` : Contient les résultats des tests automatisés.
- `src/` : Contient les fichiers sources `.c`.
- `tests/` : Contient les fichiers de tests.

Chacun des dossiers `include/`, `obj/` et `src/` contient leurs versions des fichiers suivants :

- `ell_curve` : Implémente la structure de courbe elliptique telle que détaillée dans la section *§2.1.*
- `ell_point` : Modélise les points d'une courbe elliptique et les opérations de groupes associées dans un anneau de torsion. Ceci est expliqué dans la section *§2.2.*
- `list` : Implémente la structure de liste, ceci est détaillé dans la section *§3.4.*
- `schoof` : Contient le calcul des polynômes de division et l'algorithme de Schoof expliqués aux sections *§3.1* et *§3.2.*
- `tors_ring` : Implémente les anneaux de torsion générique selon la section *§3.3.*

Le dossier `tests/` contient les fichiers `test_compare.h` et `test_compare.c` qui comparent notre implémentation de l'algorithme de Schoof avec une méthode naïve pour des courbes elliptiques aléatoires, ainsi que `test_perf.h` et `test_perf.c` qui mesure le temps d'exécution de notre algorithme pour des courbe elliptiques aléatoires.

Pour obtenir un inventaire des commandes, il suffit d'ouvrir un terminal dans le dossier du projet et d'entrer la commande `make help`.

1.2 Un mot sur la librairie FLINT

Tous les calculs algébriques ont été réalisés à l'aide de la librairie FLINT. Cette librairie respecte plusieurs conventions de programmation que nous avons empruntées. Elle possède ses propres versions des types `unsigned long` et `long`, respectivement `ulong` et `slong`, ce sont des entiers optimisés pour l'arithmétique bas niveau. FLINT fonctionne de paire avec la librairie des grands entiers GMP, elle a son propre type de grand entier `fmpz_t`. Comme FLINT est optimisé pour la performance, elle utilise abondamment les pointeurs pour définir ses types, ainsi tout nouveau type `type_t` est défini de la façon suivante :

```
typedef struct {
    ...
} type_struct;

typedef type_struct type_t[1];
```

Cela a plusieurs avantages :

- elle permet de modifier directement les arguments sans alourdir le code par l'usage systématique de l'opérateur `&`,

- la copie d'arguments sur la pile lors d'un appel de fonction est remplacée par une copie de pointeurs ce qui est nettement plus efficace, en particulier lorsqu'on manipule des structures volumineuses telles que les grands entiers,
- la déclaration d'une variable de type `type_t` entraîne automatiquement l'allocation de l'espace mémoire nécessaire, contrairement à un simple pointeur.

Les types de FLINT sont donc des pointeurs vers des structures intermédiaires. Ces structures stockent à la fois des métadonnées et un pointeur vers une zone mémoire distincte contenant les données effectives qui doivent être allouées dynamiquement sur le tas. Ainsi tout type `type_t` de ce genre possède les primitives suivantes :

`type_t_init()` : Alloue la mémoire effective d'une variable.

`type_t_set()` : Modifie le contenu effectif d'une variable.

`type_t_clear()` : Libère la mémoire effective allouée à la variable.

Il y a tout de même un gros piège avec cette pratique : l'aliasing. Considérons l'algorithme suivant :

Algorithme 1 : Exemple.

Données : `rop`, `op1`, `op2`.
Sorties : `rop` prend la valeur `res` + `op1` + `op2`.

```
1 rop ← rop + op1;
2 rop ← rop + op2;
```

Sous la convention de déclaration des variables de FLINT, si `rop` = `op2` en entrée alors `rop` prendra à l'issue de cet algorithme la valeur $2(\texttt{op1}+\texttt{op2})$ puisque la première ligne modifie `rop` et donc également `op2`. On peut résoudre le problème en copiant les opérandes avant de modifier l'un des arguments, mais cette approche implique une duplication coûteuse de la mémoire effective. Une stratégie plus efficace consiste à effectuer les calculs dans une variable temporaire `temp`, puis à transférer le résultat vers `rop`. Mieux encore, en échangeant simplement les pointeurs vers les mémoires effectives de `temp` et `rop`, on évite toute copie et on élimine complètement ce surcoût, ce qui donnerait ici :

Algorithme 2 : Exemple corrigé.

Données : `rop`, `op1`, `op2`.
Sorties : `rop` prend la valeur `rop` + `op1` + `op2`.

```
1 Déclarer une variable temp;
2 temp ← rop + op1;
3 temp ← temp + op2;
4 Échanger les pointeurs vers les mémoires effectives de rop et temp;
5 Libérer la variable temp;
```

Nous utiliserons abondamment le type `fq_t` qui modélise un élément d'un corps fini. Il est important de noter qu'une variable de type `fq_t` désigne un élément générique d'un corps fini et qui n'a aucune valeur bien définie indépendamment d'un contexte donné. En effet, le corps fini $\mathbb{F}_q$ est, mathématiquement parlant, un quotient $\mathbb{F}_p[x]/(f(x))$ avec $f(x)$ irréductible unitaire de degré d. Notant α le générateur canonique de ce quotient, tout élément de $\mathbb{F}_q$ s'écrit d'une unique façon sous la forme

$$a_0 + a_1\alpha + \cdots a_{d-1}\alpha^{d-1}$$

avec $a_i \in \mathbb{F}_p$ et on peut donc identifier un élément de $\mathbb{F}_q$ avec un polynôme de $\mathbb{F}_p[x]$. Or cette présentation dépend complètement de la représentation de $\mathbb{F}_q$ choisie *i.e* du choix de $f(x)$, une telle représentation est décrite par le type `fq_ctx_t`. En pratique, cela implique que le contexte doit toujours être fourni en argument et ne doit être fixé qu'au moment de l'appel.

Notons que FLINT prend en charge de manière automatique les problèmes d'aliasing dans ses fonctions arithmétiques pour les types `fq_t` et `fq_poly_t`, ce qui permet d'écrire sans problème des instructions telles que `fq_mul(a,a,a,ctx)` qui signifie $a \leftarrow a \times a$.

On termine cette section avec le fait suivant : dans $\mathbb{F}_q$, l'addition s'effectue en temps linéaire et la multiplication en temps quadratique, un élément de $\mathbb{F}_q$ étant représenté sur $O(\log q)$ bits.

2 Rappels sur les courbes elliptiques

2.1 Généralités

Par courbe elliptique sur $\mathbb{F}_q$, on entend une une variété projective définie une équation de Weierstrass

$$y^2 = x^3 + ax + b$$

où $a, b \in \mathbb{F}_q$ satisfont $4a^3 + 27b^2 \neq 0$. D'un point de vue plus géométrique, une courbe elliptique peut être définie comme une courbe projective lisse de genre 1. Le théorème de Riemann-Roch implique que toute courbe elliptique est isomorphe à une courbe de Weierstrass si $p \notin \{2, 3\}$, hypothèse faite ici. Il n'y a donc aucune perte de généralité à adopter cette forme comme définition. Les courbes elliptiques sont donc implémentées dans le fichier `ell_curve.h` par le type `ell_curve_t` par :

```
typedef struct {
    fq_t a;
    fq_t b;
} ell_curve_struct;

typedef ell_curve_struct ell_curve_t[1];
```

Il faudra vérifier la condition $4a^3 + 27b^2 \neq 0$ en amont de l'initialisation d'une courbe elliptique.

En coordonnées projectives $[X : Y : Z]$ avec $x = X/Z$ et $y = Y/Z$, l'équation de Weierstrass s'écrit

$$Y^2Z = X^3 + aXZ^2 + bZ^3.$$

On note $E(\mathbb{F}_q)$ l'ensemble des points de E à coordonnées dans $\mathbb{F}_q$. La courbe E admet un unique point à l'infini noté $O = [0 : 1 : 0]$.

$E(\mathbb{F}_q)$ est muni d'une structure de groupe canonique que l'on peut définir de la façon suivante :

Définition 2.1. Soit $P_1 \in E(\mathbb{F}_q)$, on définit $P_1 + O = P_1$ et $O + P_1 = P_1$. Soit $P_2 \in E(\mathbb{F}_q)$, on suppose que $P \neq O$ et $Q \neq O$. Les coordonnées de $P_3 = P_1 + P_2$ peuvent être calculées comme suit :

— [•] Supposons que $P_1 \neq \pm P_2$, on pose $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$, alors

$$x_3 = \lambda^2 - x_1 - x_2 \text{ et } y_3 = \lambda(x_1 - x_3) - y_1$$

- Si $P_1 = -P_2$, on a $P_3 = O$.
- Supposons que $P_1 = P_2$, on pose $\lambda = \frac{3x_1^2 + a}{2y_1}$, alors

$$x_3 = \lambda^2 - 2x_1 \text{ et } y_3 = \lambda(x_1 - x_3) - y_1.$$

Remarque 2.2. Si $P \in E(\mathbb{F}_q)$ on a $-P = (x_P, -y_P)$.

En réalité, une courbe elliptique est un groupe algébrique et sa loi de groupe s'exprime par un morphisme de variétés algébriques

$$E \times E \longrightarrow E.$$

L'énorme avantage de cette approche est que la loi de groupe est purement géométrique. Plus généralement, pour toute $\mathbb{F}_q$-algèbre R, ce morphisme induit fonctoriellement une structure de groupe sur l'ensemble $E(R)$ des R-points de E :

$$E(R) \times E(R) \longrightarrow E(R).$$

héritée directement de la structure de groupe algébrique de E. Cette donnée caractérise la structure de E.

L'expression de la loi de groupe sur $E(\mathbb{F}_q)$ a le gros désavantage de nécessiter des divisions dans le calcul des pentes. Dans $\mathbb{F}_q$ la division est bien plus coûteuse que la multiplication car elle nécessite une relation de Bézout. De plus, cette formule ne s'applique pas à la loi de groupe sur les points à coefficients dans un anneau, or nous verrons qu'il est absolument central de savoir calculer algorithmiquement cette loi de groupe sur des anneaux qui ne sont même pas intègres. Avant d'aborder la résolution de ce problème dans la section suivante, nous rappelons quelques propriétés bien connues de la trace du Frobenius.

Définition 2.3. On note $\varphi_q : E \longrightarrow E$ le morphisme de Frobenius défini pour toute $\mathbb{F}_q$-algèbre R par

$$\begin{aligned} \varphi_q : E(R) &\longrightarrow E(R) \\ (x, y) &\longmapsto (x^q, y^q). \end{aligned}$$

Définition 2.4. Pour tout $m \in \mathbb{Z}$, on note $[m] : E \longrightarrow E$ l'opérateur défini par $[m]P = \underbrace{P + \cdots + P}_{m \text{ fois}}$ si $m \geqslant 0$ et $[m]P = -[-m](P)$ si $m < 0$.

Théorème 2.5. Soit $a_q = q + 1 - \#E(\mathbb{F}_q)$, on a les fait suivants :

(*i*) $|a_q| \leqslant 2\sqrt{q}$ (borne de Hasse).

(*ii*) φ_q vérifie l'identité

$$\varphi_q^2 - a_q \varphi_q + [q] = 0.$$

Démonstration. □

La borne de Hasse montre qu'une courbe elliptique générale sur $\mathbb{F}_q$ possède un nombre de points rationnels de l'ordre de $q + \mathrm{O}(\sqrt{q})$. Cette borne est optimale, il existe en effet des courbes elliptiques pour lesquelles cette borne est atteinte.

2.2 Le morphisme de Frobenius

3 L'algorithme de Schoof

3.1 Idée générale

La méthode la plus naïve pour calculer $\#E(\mathbb{F}_q)$ consiste à tester l'égalité $y^2 = x^3 + ax + b$ pour tous les q^2 couples (x, y) de $\mathbb{F}_q^2$, avec une complexité en $\mathrm{O}(q^2)$. On peut déjà faire mieux en remarquant qu'à $x \in \mathbb{F}_q$ fixé, il y a au plus deux possibilités pour y. Plus précisément, en remarquant que

$$\forall a \in \mathbb{F}_q, \#\{y \in \mathbb{F}_q, y^2 = a\} = 1 + \left(\frac{a}{q}\right)$$

où $\left(\frac{\cdot}{q}\right)$ désigne le symbole de Jacobi [1] on obtient, en prenant en compte le point à l'infini :

1. Dans ce cas précis où q est une puissance d'un nombre premier, le symbole de Jacobi caractérise les résidus quadratiques tout comme le symbole de Legendre.

Proposition 3.1.

$$\#E(\mathbb{F}_q) = q + 1 + \sum_{x \in \mathbb{F}_q} \left(\frac{x^3 + ax + b}{q} \right).$$

Le symbole de Jacobi peut être calculé en $\mathrm{O}(\log q)$ opération élémentaires dans $\mathbb{F}_q$ donc cette méthode permet d'obtenir $\#E(\mathbb{F}_q)$ avec une complexité $\mathrm{O}(q \log q)$. Cet algorithme est toujours exponentiel, mais il servira de base pour tester notre implémentation en comparant l'algorithme de Schoof avec cette méthode naïve pour des courbes elliptiques aléatoires. Le code se situe dans les fichiers `test_compare.h` et `test_compare.c`, et les résultats des tests dans le fichier `results_compare.csv`.

Remarquons que calculer $\#E(\mathbb{F}_q)$ est équivalent à calculer a_q, ce qui bien sûr très difficile. Une idée générale simple mais pourtant diablement efficace consiste à se ramener au calcul de a_q modulo des petits entiers.

Soient $P \in E(\overline{\mathbb{F}_q})[\ell]$ et t_ℓ le reste de la division euclidienne de a_q par ℓ, alors $[a_q]P = [t_\ell]P$ d'où

$$\varphi_q^2(P) + [t_\ell]P = [q]\varphi_q(P)$$

d'après la Proposition ? ? ?. t_ℓ est inconnu mais il ne peut prendre que ℓ valeurs qu'on peut toutes tester individuellement. Bien sûr, pour un point de ℓ-torsion donné, il peut y avoir des faux positifs, mais on a la résultat suivant :

Proposition 3.2. Soit $t \in \mathbb{F}_q$, alors sous l'hypothèse $\ell \nmid q$,

$$t \equiv a_q \pmod{l} \iff \forall P \in E(\overline{\mathbb{F}_q})[\ell] \setminus \{O\}, \varphi_q^2(P) - [t]\varphi_q(P) = [q]P.$$

Démonstration. Le sens $\Rightarrow$ est une conséquence directe de la Proposition ? ? ?. Supposons réciproquement que t satisfait la condition ci-dessus, l'égalité est également trivialement vraie pour $P = O$, cela signifie donc que φ_q est nul en tant qu'endomorphisme de $E(\overline{\mathbb{F}_q})[\ell]$. On a donc, d'après la Proposition ? ? ?

$$0 = (\varphi_q^2 - a_q\varphi_q + \mathrm{id}) - (\varphi_q^2 - t\varphi_q + \mathrm{id}) = (t - a_q)\varphi_q.$$

Comme $\ell \nmid q$, alors φ_q est un endomorphisme inversible de $E(\overline{\mathbb{F}_q})[\ell]$ (A JUSTIFIER ?) et ainsi $t \equiv a_q \pmod{\ell}$. □

On peut donc calculer a_q de la façon suivante : soient $\ell_1, \ldots, \ell_r$ des nombres premiers différents de p tels que $A = \ell_1 \cdots \ell_r > 4\sqrt{q}$ de sorte que a_q soit l'unique représentant de sa classe modulo A dans $]-A/2, A/2]$ d'après la borne de Hasse. On vérifie la condition de la Proposition ? ? ? dans chaque sous-groupe $E(\overline{\mathbb{F}_q})[\ell_i]$ pour obtenir $a_q \pmod{\ell_i}$, on en déduit alors $a_q \pmod{A}$ et donc a_q à l'aide du théorème des restes chinois.

En pratique, cet algorithme est inutilisable, cela demande en effet la connaissance préalable de chacun des sous-groupes de torsion $E(\overline{\mathbb{F}_q})[\ell_i]$, et ces derniers sont très compliqués à calculer. Pire, ils sont généralement à coefficients dans de très grandes extensions de $\mathbb{F}_q$ ce qui alourdit considérablement les calculs.

L'astuce de Schoof est la suivante : l'égalité $\varphi_q^2(P) + [t_\ell]P = [q]\varphi_q(P)$ doit être vérifiée par tout point $P \in \mathbb{A}^2_{\mathbb{F}_q}$ vérifiant les conditions suivantes :

$$y_P^2 = x_P^3 + ax_P + b \text{ et } [\ell]P = O.$$

On peut donc s'affranchir de la connaissance de $E(\overline{\mathbb{F}_q})[\ell]$ en testant l'égalité ? ? ? pour un point générique ne vérifiant que les deux égalités ci-dessus. Plus précisément, soit $I_{E,\ell}$ le noyau du morphisme de $\mathbb{F}_q$-algèbres

$$\begin{aligned}\Phi_\ell : \mathbb{F}_q[x,y] &\longrightarrow \prod_{P\in E(\overline{\mathbb{F}_q})[\ell]\setminus\{O\}} \overline{\mathbb{F}_q} \\ f(x,y) &\longmapsto \big(f(x_P,y_P)\big)_{P\in E(\overline{\mathbb{F}_q})[\ell]\setminus\{O\}}\end{aligned},$$

c'est un idéal de $\mathbb{F}_q[x,y]$.

Définition 3.3. On définit

$$\mathcal{R}_{E,\ell} = \mathbb{F}_q[x,y]/I_{E,\ell}$$

l'anneau de ℓ-torsion générique de E.

Théorème 3.4. On a :

$$f = 0 \text{ dans } \mathcal{R}_{E,\ell} \iff \forall P \in E(\overline{\mathbb{F}_q})[\ell]\setminus\{O\}, f(x_P,y_P) = 0.$$

Démonstration. Il s'agit de montrer que l'idéal annulateur de $E(\overline{\mathbb{F}_q})[\ell]\setminus\{O\}$ est I_ℓ. Or $E(\overline{\mathbb{F}_q})[\ell]\setminus\{O\}$ est la sous variété fermée de $\mathbb{A}^2_{\overline{\mathbb{F}_q}}$ définie par l'idéal $I_{E,\ell}$ de $\mathbb{F}_q[x,y]$ par construction, il suffit donc de montrer que ce dernier est radiciel d'après le Nullstellensatz. Cela revient à montrer que l'anneau $\mathcal{R}_{E,\ell}$ n'a pas d'élément nilpotent. Φ_ℓ induit par passage au quotient une injection

$$\overline{\Phi_\ell} : \mathcal{R}_{E,\ell} \hookrightarrow \prod_{P\in E(\overline{\mathbb{F}_q})[\ell]\setminus\{O\}} \overline{\mathbb{F}_q},$$

or un produit de corps étant réduit, $\mathcal{R}_{E,\ell}$ l'est donc également, ce qu'on voulait démontrer. □

Comme la loi de groupe sur $E(\overline{\mathbb{F}_q})$ est rationnelle, la condition de Schoof de la Proposition 3.2 peut se réécrire comme suit :

$$t \equiv a_q \pmod{\ell} \iff (x^{q^2}, y^{q^2}) + [q](x,y) = [t](x^q, y^q) \text{ dans } \mathcal{R}_\ell^2.$$

Il subsiste la question de la description de l'idéal $I_{E,\ell}$ qui peut devenir très complexe, nécessitant l'utilisation de bases de Gröbner. Heureusement, cet idéal possède des générateurs qu l'on peut facilement en calculer à l'aide d'une formule récursive, c'est l'objet de la section suivante.

3.2 Polynômes de division d'une courbe elliptique

Dans toute cette section, on fixe une courbe elliptique E sur $\mathbb{F}_q$ définie par l'équation de Weierstrass $y^2 = x^3 + ax + b$. La suite que l'on définit ici s'appelle la suite des polynômes de divisions de E.

Définition 3.5. Soit $(\psi_n)_{n\geqslant 0}$ la suite à valeurs dans $\mathbb{F}_q[x,y]$ définie par

$$\begin{aligned}\psi_0 &= 0\\ \psi_1 &= 1\\ \psi_2 &= 2y\\ \psi_3 &= 3x^4 + 6ax^2 + 12bx - a^2\\ \psi_4 &= 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3)\end{aligned}$$

puis, par récurrence,

$$\begin{aligned}\psi_{2m} &= \frac{\psi_m}{2y}\left(\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2\right)\\ \psi_{2m+1} &= \psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3\end{aligned}$$

Calculer ψ_n se fait en $\mathrm{O}(\log n)$ multiplications dans $\mathbb{F}_q[x,y]$. S'il y a une ambiguïté sur la courbe elliptique à laquelle est rattaché les polynômes de division, on notera parfois $\psi_{E,n}$ au lieu de ψ_n.

Avant de démontrer la proposition suivante, remarquons que tout élément de $\mathbb{F}_q[x,y]/(y^2-x^3-ax-b)$ s'écrit d'une unique façon $A(x)+B(x)y$ avec $A(x), B(x) \in \mathbb{F}_q[x]$.

Proposition 3.6. Il existe une unique suite $(f_n)_{n\geqslant 0}$ à valeurs dans $\mathbb{F}_q[x]$ vérifiant

$$\psi_{2m}(x,y) = f_{2m}(x)y \text{ et } \psi_{2m+1}(x,y) = f_{2m+1}(x)$$

De plus, la suite $(f_n)_{n\geqslant 1}$ peut être calculée récursivement par la relation de récurrence suivante :

$$\begin{aligned}
f_0 &= 0\\
f_1 &= 1\\
f_2 &= 2\\
f_3 &= 3x^4+6ax^2+12bx-a^2\\
f_4 &= 4(x^6+5ax^4+20bx^3-5a^2x^2-4abx-8b^2-a^3)
\end{aligned}$$

puis,

$$\begin{aligned}
f_{2m} &= \frac{f_m}{2}\left(f_{m+2}f_{m-1}^2 - f_{m-2}f_{m+1}^2\right)\\
f_{2m+1} &= (x^3+ax+b)^2 f_{m+2}f_m^3 - f_{m-1}f_{m+1}^3 \text{ si } m \text{ est pair}\\
f_{2m+1} &= f_{m+2}f_m^3 - (x^3+ax+b)^2 f_{m-1}f_{m+1}^3 \text{ si } m \text{ est impair}
\end{aligned}$$

Démonstration. On raisonne par récurrence sur n, le cas $n \leqslant 4$ étant immédiat, on suppose le résultat vrai pour des indices $< n$. On traite uniquement le cas $n = 2m+1$ avec m pair, les autres se traitant de façon analogue. La relation de récurrence vérifiée par $(\psi_n)_{n\geqslant 0}$ implique, par hypothèse de récurrence :

$$\begin{aligned}
\psi_{2m+1}(x) &\equiv f_{m+2}(x)y \times f_m(x)^3y^3 - f_{m-1}(x) \times f_{m+1}(x) \pmod{y^2-x^3-ax-b}\\
&= (x^3+ax+b)^2 f_{m+2}(x)f_m(x) - f_{m-1}(x)f_{m+1}(x) \pmod{y^2-x^3-ax-b}
\end{aligned}$$

donc, par unicité de la représentation dans l'anneau $\mathbb{F}_q[x,y]/(y^2-x^3-ax-b)$, il existe un unique $f_{2m+1}(x) \in \mathbb{F}_q[x]$ tel que $\psi_{2m+1}(x) \equiv f_{2m+1}(x) \pmod{y^2-x^3-ax-b}$, défini par

$$f_{2m+1}(x) = (x^3+ax+b)^2 f_{m+2}(x)f_m^3(x) - f_{m-1}(x)f_{m+1}(x)^3.$$

□

En pratique, on calculera donc la suite de polynômes $(f_n)_{n\geqslant 1}$ en se restreignant à l'arithmétique de $\mathbb{F}_q[x]$.

Proposition 3.7. On a les faits suivants :

(*i*)
$$\deg f_n = \begin{cases} \dfrac{n^2-4}{2} \text{ si } n \text{ est pair,} \\ \dfrac{n^2-1}{2} \text{ si } n \text{ est impair.} \end{cases}$$

(*ii*) Si $p \nmid n$, le polynôme f_n est séparable.

Démonstration. □

Tout l'intérêt des polynômes de division, et qui justifie leur nom, est le théorème suivant :

Théorème 3.8. Supposons que $p \nmid n$ et soit $P = (x_P, y_P) \in E(\overline{\mathbb{F}_q}) \setminus \{O\}$, alors

$$[n]P = O \iff \psi_n(x_P) = 0.$$

On obtient donc une présentation complètement effective de l'idéal $I_{E,\ell}$ et donc de l'anneau de ℓ-torsion générique $\mathcal{R}_{E,\ell}$:

Corollaire 3.9. Supposons que $\ell \neq p$, alors

$$I_{E,\ell} = (\psi_\ell(x), y^2 - x^3 - ax - b).$$

Démonstration. Pour une démonstration du théorème, on renvoie vers [? ? ?]. Quant au corollaire, seule l'inclusion $\subseteq$ n'est pas évidente. Soit $f \in I_{E,\ell}$, la division euclidienne de f par $y^2 - x^3 - ax - b$ pour l'ordre lexicographique induit par $y > x$ s'écrit

$$f(x,y) = (y^2 - x^3 - ax - b)g(x,y) + A(x) + B(x)y.$$

Posons

$$E(\overline{\mathbb{F}_q})[\ell] \setminus \{O\} = \{P_1, \ldots, P_r\},$$

alors

$$\forall i \in [\![1, r]\!], A(x_{P_i}) + B(x_{P_i})y_{P_i} = 0 \text{ et } A(x_{P_i}) + B(x_{P_i})y_{P_i}$$

d'où, $A(x_{P_i}) = B(x_{P_i})y_{P_i} = 0$. Or chacun des P_i est affine donc $y_{P_i} \neq 0$, et on obtient donc $B(x_{P_i}) = 0$. Ainsi $A(x)$ et $B(x)$ s'annulent sur l'ensemble des racines de $\psi_\ell(x)$, donc sont dans l'idéal annulateur de la variété $V(\psi_\ell)$, donc $A(x)$ et $B(x)$ sont dans l'idéal $\sqrt{(\psi_\ell(x)}$ de $\overline{\mathbb{F}_q}[x]$ d'après le Nullstellensatz. Or $\ell \neq p$ par hypothèse donc $\psi_\ell(x)$ est séparable d'après la Proposition 3.7 et l'idéal $(\psi_\ell(x))$ est donc radiciel et $A(x)$ et $B(x)$ sont dans l'idéal $(\psi_\ell(x))$ de $\overline{\mathbb{F}_q}[x]$. Ceci implique par descente de Galois que $A(x)$ et $B(x)$ sont dans l'idéal de $(\psi_\ell(x))$ de $\mathbb{F}_q[x]$: écrivons par exemple $A(x) = \psi_\ell(x)h(x)$ avec $h(x) \in \overline{\mathbb{F}_q}[x]$, voire même $h(x) \in \mathbb{F}_{q^d}[x]$ avec $d \geqslant 1$ alors, notant σ_q le Frobenius sur $\overline{\mathbb{F}_q}$, on a

$$A(x) = \frac{1}{d}\sum_{j=0}^{d-1} \sigma_q^j \cdot A = \frac{1}{d}\sum_{j=0}^{d-1} (\sigma_q^j \cdot \psi_\ell(x))(\sigma_q^j \cdot h(x)) = \psi_\ell(x) \underbrace{\left(\frac{1}{d}\sum_{j=0}^{d-1} \sigma_q^j \cdot h(x)\right)}_{\in \mathbb{F}_{q^d}^{\sigma_q}[x] = \mathbb{F}_q[x]}.$$

Ceci permet de conclure que $f(x,y) \in (y^2 - x^3 - ax - b), \psi_\ell(x))$. □

3.3 L'utilisation de listes

Il reste un dernier détail d'implémentation, pourtant crucial : on ne connaît pas à l'avance le nombre de nombres premiers nécessaires pour reconstituer a_q. Cela pose donc un problème si l'on veut définir un tableau de valeurs répertoriant les différentes réductions de a_q modulo des nombres premiers. On pourrait procéder de la façon suivante : soit $g(q)$ le plus petit entier vérifiant $\ell_1 \cdots \ell_{g(q)+1} > 4\sqrt{q}$ où ℓ_j désigne le j-ème nombre premier impair (on en rajoute un au cas où un des ℓ_j est égal à p), on pourrait allouer dynamiquement de la mémoire pour un tableau de taille $g(q)$, ce qui serait exactement la taille nécessaire. Au moins quatre possibilités s'offrent à nous :

- Pré-calculer $g(q)$ avant chaque appel à l'algorithme principal mais cela serait assez coûteux, et d'autant plus si l'algorithme avait vocation à être très souvent utilisé.
- Pré-calculer une table répertoriant les bornes d'entiers pour lesquels $g(q)$ vaut une certaine valeur et les stocker dans un fichier binaire au moment de la compilation. Le désavantage est qu'il faudrait pour cela instaurer un nombre de bits maximal pour q, même si cela ferait parfaitement l'affaire pour des applications en cryptographie.

- Trouver une minoration effective de $g(q)$. En effet $g(q)$ est caractérisé par l'encadrement

$$\prod_{j=1}^{g(q)+1} \ell_j > 4\sqrt{q} \geqslant \prod_{j=1}^{g(q)} \ell_j$$

  c'est-à-dire

$$\vartheta(\ell_{g(q)}) > 2\log 2 + \frac{\log q}{2} \geqslant \vartheta(\ell_{g(q)-1})$$

  où [2] $\vartheta(x) = \sum_{\ell \leqslant x} \log \ell$ désigne la fonction de Tchebychev de première espèce. Or on sait que $\vartheta(x) \underset{x\to+\infty}{\sim} x$ (c'est équivalent au théorème des nombres premiers), et il existe des bornes effectives sur ϑ ce qui permet d'obtenir une telle minoration pour $g(q)$. Ceci est bien trop compliqué et pas vraiment dans l'esprit d'un algorithme de programmation, mais a l'avantage de la complexité algorithmique puisque tout le travail est d'ordre théorique.
- Ignorer $g(q)$ et rajouter à chaque fin de boucle les valeurs trouvées de $a_q \pmod{\ell}$ dans une liste.

Il a été décidé d'opter pour la dernière solution, elle permet d'éviter la restriction sur une borne maximale du nombre de bits de q tout en restant assez simple. Si notre implémentation était vouée à être utilisée dans un cadre cryptographique, nous aurions opté pour la seconde solution puisque les tableaux sont plus efficaces que les listes à longueur connue.

Une liste est composée de cellules qui chacune contient des informations (dont ici l'indice de la cellule dans la liste, bien que ça ne soit pas nécessaire) et un pointeur vers la prochaine cellule. Une cellule est donc une structure auto-référentielle. Le type des cellules contenant des éléments de type `type_t` est alors défini ainsi :

```
typedef struct cell_type_struct {
    int index;
    type_t var;
    cell_type_struct* next;
} cell_type_t;
```

Une fois ceci fait, une liste est définie par un couple de pointeurs vers des cellules : sa tête et sa queue. On pourrait en théorie se passer du pointeur vers la queue de la liste, puisque la dernière cellule correspond à celle dont le pointeur vers la prochaine cellule prend la valeur `NULL`, mais cela à l'avantage d'éviter de devoir parcourir toute la liste à chaque ajout d'une nouvelle cellule. On définit une liste stockant des variables de type `type_t` comme suit :

```
typedef struct {
    cell_type_t* head;
    cell_type_t* tail;
} list_type_t;
```

Dans notre cas, nous aurons besoin de trois listes : une liste répertoriant les différentes réductions $a_q \pmod{\ell}$ et une autre stockant ces mêmes nombres premiers ℓ, toutes deux contiennent des entiers non signés de type `ulong`, et enfin une liste contenant les polynômes de division $\psi_\ell(x)$ qui sont eux de type `fq_poly_t`. On pourrait créer une structure de liste générique qui englobe ces deux possibilités, mais du fait que le type `fq_poly_t` nécessite un traitement spécial d'allocation et de libération de la mémoire qui ne concerne pas les types d'entiers usuels, il a été décidé de créer deux types distincts : `list_ulong_t` et `list_fq_poly_t`.

Les usages de ces deux types de listes sont fondamentalement différents. Les listes de type `list_ulong_t` servent à regrouper les données calculées à chaque itération de la boucle alors que celle de type

2. La somme est indexée sur l'ensemble des nombres premiers inférieurs ou égaux à x.

`list_fq_poly_t` n'existe que par nécessité de garder la trace des polynômes de division déjà calculés afin de pouvoir évaluer $\psi_\ell(x)$ à l'aide de la relation de récurrence, ce n'est qu'un intermédiaire de calcul, elle contient donc des polynômes de division d'indices non-premiers.

4 Modélisation de l'anneau de ℓ-torsion générique $\mathcal{R}_{E,\ell}$

4.1 Structure d'un anneau de torsion

L'implémentation décrite ci-après se trouve dans les fichiers `tors_ring.h` et `tors_ring.c`.

Un anneau de torsion sera la donnée d'une courbe elliptique E, et d'un polynôme $\psi \in \mathbb{F}_q[x]$, le couple (E, ψ) représentera alors l'anneau $\mathbb{F}_q[x,y]/(y^2 - x^3 - ax - b, \psi(x))$ si E est d'équation $y^2 = x^3 + ax + b$:

```
typedef struct {
    ell_curve_t E;
    fq_poly_t psi;
} tors_ring_struct;

typedef tors_ring_struct tors_ring_t[1];
```

Ainsi l'anneau de ℓ-torsion générique $\mathcal{R}_{E,\ell}$ est représenté par le couple (E, ψ_ℓ).

Proposition 4.1. Supposons $\psi \neq 0$, alors tout élément de $\mathbb{F}_q[x,y]/(y^2 - x^3 - ax - b, \psi(x))$ s'écrit de façon unique $A(x) + B(x)y$ avec $\deg A, \deg B < \deg \psi$. De plus,

$$(A_1(x) + B_1(x)y) + (A_1(x) + B_2(x)y) = (A_1(x) + A_2(x)) + (B_1(x) + B_2(x))y,$$

$$-(A(x) + B(x)y) = (-A(x)) + (-B(x))y,$$

$$(A_1(x) + B_1(x)y) \times (A_1(x) + B_2(x)y) = A_3(x) + B_3(x)y$$

où $A_3(x)$ est le reste de la division euclidienne de $A_1(x)A_2(x) + B_1(x)B_2(x)(x^3 + ax + b)$ par $\psi(x)$, et $B_3(x)$ est le reste de la division euclidienne de $A_1(x)B_2(x) + B_1(x)A_2(x)$ par $\psi(x)$.

L'addition et la soustraction dans l'anneau de torsion représenté par le couple (E, ψ) nécessitent $\mathrm{O}(\deg \psi)$ opérations élémentaires dans $\mathbb{F}_q$. La multiplication revient à multiplier puis réduire des polynômes de degré au plus $2(\deg \psi - 1)$, ce qui nécessite $\mathrm{O}\left((\deg \psi)^2\right)$ opérations élémentaires dans $\mathbb{F}_q$.

Remarque 4.2. Lorsque $\psi = 0$ la proposition reste valide en supprimant les divisions euclidiennes et la condition $\deg A, \deg B < \deg \psi$. Ce cas ne sera jamais utilisé, mais par soucis d'exhaustivité il a été tout de même été traité à part dans le code.

Un élément d'un anneau de torsion peut donc être représenté par un unique couple de polynômes à coefficients dans $\mathbb{F}_q[x]$:

```
typedef struct {
    fq_poly_t A;
    fq_poly_t B;
} tors_elem_struct;

typedef tors_elem_struct tors_elem_t[1];
```

Suivant la philosophie de FLINT, un élément d'un anneau de torsion est implémenté de façon générique tant qu'on ne lui a pas associé un contexte. En l'occurrence, par contexte on entend ici une variable de type `fq_ctx_t` pour les variables de type `fq_poly_t`, mais également un anneau de torsion qui devient indispensable si l'on souhaite multiplier des éléments d'un anneau de torsion entre eux.

4.2 Loi de groupe d'une courbe elliptique

On a vu à la section *§2.1* qu'une courbe elliptique était un groupe algébrique, et que la loi de groupe induite sur $E(\mathbb{F}_q)$ avait une expression effective. Or cette expression a le vilain défaut de faire apparaître une division dans le calcul de pente. Cela est dommageable à au moins deux égards :

- la division est générale beaucoup plus coûteuse que la multiplication, par exemple dans $\mathbb{F}_q$ la division nécessite une relation de Bézout,
- Nous avons besoin d'une expression de la loi de groupe sur $E(\mathcal{R}_{E,\ell})$ et $\mathcal{R}_{E,\ell}$ n'est qu'un simple anneau, il n'est même pas intègre, donc la division est à proscrire et cette formule de la loi de groupe n'est absolument pas applicable ici.

Une façon de naturelle de résoudre ce problème est de rendre la loi de groupe polynomiale, on peut procéder à l'aide des coordonnées projectives en normalisant dans le but de supprimer les pôles.

Proposition 4.3. Soient R une $\mathbb{F}_q$-algèbre, $P_1 = [X_1 : Y_1 : Z_1]$ et $P_2 = [X_2 : Y_2 : Z_2]$ des points de $E(R)$ en coordonnées projectives. Les coordonnées projectives de $P_3 = P_1 + P_2$ dans $E(R)$ peuvent être calculées comme suit :

- Supposons que $P_1 \neq \pm P_2$, on pose $U = Y_2Z_1 - Y_1Z_2, V = X_2Z_1 - X_1Z_2$, alors

$$\begin{aligned} X_3 &= U^2Z_1Z_2 - V^3 - 2V^2X_1Z_2, \\ Y_3 &= U(V^2X_1Z_2 - X_3) - V^3Y_1Z_2, \\ Z_3 &= V^3Z_1Z_2. \end{aligned}$$

- Si $P_1 = -P_2$, on a $P_3 = O$,
- Supposons que $P_1 = P_2$, on pose $W = 3X_1^2 + aZ_1^4$, alors

$$\begin{aligned} X_3 &= W^2Z_1 - 8X_1^2Y_1^2, \\ Y_3 &= W(4X_1^2Y_1^2 - X_3) - 8Y_1^4Z_1, \\ Z_3 &= 2Y_1Z_1. \end{aligned}$$

Un œil aguerri remarquera que dans ce système de coordonnées, le doublement d'un point peut s'effectuer en 7 multiplications et 5 passages au carré dans R, et la somme en 12 multiplications et 2 passages au carrés dans R.

On peut réduire ce nombre d'opération en changeant de système de coordonnées. En effet le but étant de supprimer les pôles, une fois avoir remarqué que x a un unique pôle en O d'ordre 2 et y a un unique pôle en O d'ordre 3, on serait tenter d'adopter le système de coordonnées suivant :

Définition 4.4. Les coordonnées jacobiennes $[X : Y : Z]$ sont définies par $x = X/Z^2$ et $y = Y/Z^3$. Dans ce système de coordonnés, l'équation de Weierstrass devient

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

et le point à l'infini est $O = [1 : 1 : 0]$.

Remarque 4.5. Soit R une $\mathbb{F}_q$-algèbre.

(*i*) Soit $P \in \mathbb{P}^2_R$ de coordonnées jacobiennes $[X : Y : Z]$, alors P est affine si et seulement si $Z \in R^\times$.

(*ii*) Pour des points $P_1, P_2 \in \mathbb{P}^2_R$ en coordonnées jacobiennes, on a

$$P_1 = P_2 \iff \exists \lambda \in R^\times, \begin{cases} X_2 = \lambda^2 X_1 \\ Y_2 = \lambda^3 Y_1 \\ Z_2 = \lambda Z_1 \end{cases}.$$

Proposition 4.6. Soient R une $\mathbb{F}_q$-algèbre, $P_1 = [X_1 : Y_1 : Z_1]$ et $P_2 = [X_2 : Y_2 : Z_2]$ des points de $E(R)$ en coordonnées jacobiennes. Les coordonnées jacobiennes de $P_3 = P_1 + P_2$ dans $E(R)$ peuvent être calculées comme suit :

- Supposons que $P_1 \neq \pm P_2$, on pose

$$A = Z_1^2, B = Z_2^2, C = X_1 B, D = X_2 A, E = Z_1 Z_2, F = Z_1 E, G = Z_2 E,$$
$$H = Y_1 G, I = Y_2 F, J = D - C, K = I - H, L = J^2, M = JL, N = CL,$$

  alors

$$X_3 = K^2 - M - 2N,$$
$$Y_3 = K(N - X_3) - HM,$$
$$Z_3 = JE.$$

- Si $P_1 = -P_2$, on a $P_3 = O$.
- Supposons que $P_1 = P_2$, on pose

$$A = Y_1^2, B = X_1 A, C = 8A^2, D = X_1^2, E = Z_1^4, F = 3D - aF,$$

  alors

$$X_3 = F^2 - 2B,$$
$$Y_3 = BFX_3 - C,$$
$$Z_3 = 2Y_1 Z_1.$$

On voit ainsi qu'en coordonnées jacobiennes, le doublement d'un point se fait en 4 multiplications et 6 passage au carré, et l'addition en 12 multiplication et 4 passages au carré.

Il est à noter qu'en général le passage au carré est plus rapide à calculer qu'une multiplication. En effet, le carré correspond à la restriction de la multiplication à la diagonale, ce qui introduit des symétries absentes dans le cas bilinéaire général. Ces symétries permettent d'éliminer les termes redondants, ce qui réduit typiquement d'un facteur 2 le nombre d'opérations nécessaires [3]

De par les avantages qu'offrent ce système de coordonnées, un point d'une courbe elliptique sera à partir de maintenant toujours décrit à l'aide de dernier. Nous avons besoin d'implémenter la loi de groupe d'une courbe elliptique uniquement pour des points à coordonnées dans des anneaux de torsion, les éléments de cette sorte d'anneau seront de type `tors_elem_t`. Les points d'une courbe elliptique sont modélisés par le type `ell_point_t` dans les fichiers `ell_point.h` et `ell_point.c` par :

```
typedef struct {
    tors_elem_t X;
    tors_elem_t Y;
    tors_elem_t Z;
} ell_point_struct;

typedef ell_point_struct ell_point_t[1];
```

Les lois de groupes sont donc à interpréter au sens de l'arithmétique dans un anneau de torsion. Notons que la courbe elliptique à laquelle appartient un point est complètement implicite.

Il reste une dernière optimisation offerte par les coordonnés jacobiennes. Supposons que l'on veuille calculer la somme $P_1 + P_2$ avec P_1 affine, cette situation arrive très souvent lorsqu'on veut calculer les itérés $[m]P$ d'un point affine avec la méthode *double & square*, alors on peut réduire le nombre d'opérations à 8 multiplications et 3 passages au carré.

3. Il est bon de remarquer que le calcul d'un carré et d'une multiplication sont asymptotiquement équivalents, comme l'illustre l'identité

$$4xy = (x + y)^2 - (x - y)^2.$$

Ainsi, toute amélioration observée pour le calcul du carré relève d'optimisations constantes et ne peut apparaître d'un point de vue asymptotique.

Proposition 4.7. Soient R une $\mathbb{F}_q$-algèbre, $P_1 = [x_1 : y_1 : 1]$ et $P_2 = [X_2 : Y_2 : Z_2]$ des points de $E(R)$ en coordonnées jacobiennes tels que $P_1 \neq \pm P_2$. Les coordonnées jacobiennes de $P_3 = P_1 + P_2$ dans $E(R)$ peuvent être calculées comme suit : on pose

$$A = Z_2^2, B = Z_2 A, C = x_1 A, D = y_1 B,$$
$$E = C - X_1, F = D - Y_2, G = E^2, H = GE, I = X_1 G,$$

alors

$$\begin{aligned} X_3 &= F^2 - 2I - H, \\ Y_3 &= F(I - X_3) - Y_2 H, \\ Z_3 &= EZ_2. \end{aligned}$$

L'addition mixte n'est utile que lorsque $Z_1 = 1$ ce qui, dans un anneau, n'est pas équivalent à être affine. En pratique, lorsqu'on devra définir explicitement un point affine d'une courbe elliptique, on le mettra sous cette forme-ci. Dans notre algorithme, toute addition entre deux points d'une courbe elliptique fera apparaître au moins un point affine défini par nos soins, et qui vérifiera donc cette condition.

Pour additionner deux variables `op1` et `op2` de type `ell_point_t` on fera successivement les vérifications suivantes :

- (i) Si `op1` $= O$, on renvoie `op2`.
- (ii) Si `op2` $= O$, on renvoie `op1`.
- (iii) Si `op1` $=$ `op2`, on renvoie 2`op1` en utilisant les formules de la Proposition ? ? ?.
- (iv) Si `op1` $= -$`op2`, on renvoie O.
- (v) Si `op1->Z` $= 1$, on effectue une addition mixte avec les formules de la Proposition ? ? ? en utilisant `op1` comme point affine.
- (vi) Si `op2->Z` $= 1$, on effectue une addition mixte avec les formules de la Proposition ? ? ? en utilisant `op2` comme point affine.
- (vi) Dans les cas restants, on additionne selon les formules de la Proposition ? ? ?.

Reste enfin à savoir tester une égalité dans $E(\mathcal{R}_{E,\ell})$ avec des coordonnées jacobiennes pour pouvoir, entre autre, vérifier la condition de Schoof.

Proposition 4.8. Soient R une $\mathbb{F}_q$-algèbre et $P_1, P_2 \in \mathbb{A}^2_R$, on a en coordonnées jacobiennes :

$$P_1 = P_2 \iff \begin{cases} X_1 Z_2^2 = X_2 Z_1^2 \\ Y_1 Z_2^3 = Y_2 Z_1^3 \end{cases}.$$

Démonstration. Supposons que $P_1 = P_2$, il existe donc $\lambda \in R^\times$ tel que $X_1 = \lambda^2 X_2, Y_1 = \lambda^3 Y_2, Z_1 = \lambda Z_2$ d'où

$$X_1 Z_2^2 = (\lambda^2 X_2)(\lambda^{-1} Z_1)^2 = X_2 Z_1^2,$$
$$Y_1 Z_2^3 = (\lambda^3 Y_2)(\lambda^{-1} Z_1)^3 = Y_2 Z_1^3.$$

On suppose réciproquement ces égalités vérifiées, comme P_1 et P_2 sont affines, Z_1 et Z_2 sont inversibles donc on peut définir $\lambda = Z_2^{-1} Z_1 \in R^\times$. On a alors

$$\lambda^2 X_2 = Z_2^{-2} Z_1^2 X_2 = Z_2^{-2} Z_2^2 X_1 = X_1,$$
$$\lambda^3 X_2 = Z_2^{-3} Z_1^3 Y_2 = Z_2^{-3} Z_2^3 Y_1 = Y_1,$$
$$\lambda Z_2 = Z_2^{-1} Z_1 Z_2 = Z_1.$$

donc $P_1 = P_2$. □

Comme le seul point à l'infini d'une courbe elliptique est O, cette proposition fournit une méthode effective pour tester l'égalité entre deux points d'une courbe elliptique à coefficients dans un anneau.

5 Algorithme final

Nous sommes à présent en mesure d'écrire l'algorithme de Schoof sous sa version finale :

Algorithme 3 : Algorithme de Schoof.

Données : Une courbe elliptique E sur $\mathbb{F}_q$.
Sorties : $\#E(\mathbb{F}_q)$.

$A \leftarrow 1$;
$\ell \leftarrow 3$;
$p \leftarrow \mathrm{car}(\mathbb{F}_q)$;
Initialiser une liste vide `list_psi` de type `list_ulong_t`;
Initialiser une liste vide `list_primes` de type `list_ulong_t`;
Initialiser une liste vide `list_ts` de type `list_fq_poly_t`;
while $A \leqslant 4\sqrt{q}$ **do**
    if $\ell \neq p$ **then**
        Actualiser `list_psi` jusqu'à $\ell + 1$ éléments;
        $\psi \leftarrow$ `list_psi`$[\ell]$;
        Initialiser l'anneau de torsion $\mathcal{R}$ avec les paramètres (E, ψ);
        `success` $\leftarrow 0$;
        $t \leftarrow 0$;
        while (`success` $= 0$ & $t < \ell$) **do**
            $P \leftarrow (x^{q^2}, y^{q^2}) + [q](x, y)$ dans $E(\mathcal{R})$;
            if $t = 0$ **then**
                $Q \leftarrow O$;
            else
                $Q \leftarrow Q + (x^p, y^p)$ dans $E(\mathcal{R})$;
            end
            if $P = Q$ **then**
                `success` $\leftarrow 1$;
            else
                $t \leftarrow t + 1$;
            end
        end
        Rajouter ℓ à `list_primes`;
        Rajouter t à `list_ts`;
        $A \leftarrow \ell A$;
    end
    $\ell \leftarrow$ `next_prime`(ℓ);
end
$a_q \leftarrow$ `CRT`$^{-1}$(`list_ts`, `list_primes`);
Libérer les listes `list_psi`, `list_primes` et `list_ts`;
Renvoyer $q + 1 - a_q$;

Comme q peut être très grand, la condition $A \leqslant 4\sqrt{q}$ peut devenir assez coûteuse à vérifier, on peut la contourner en comparant le nombre de bits nécessaires pour encoder A et q qui peut être obtenu en temps constant. Il est à noter que la fonction `fmpz_bits()` renvoie le nombre de bits nécessaires pour représenter la valeur absolue de son entrée de type `fmpz_t`, il n'y a donc pas de bit de signe.

Proposition 5.1. Soient $\text{bits}(A)$ et $\text{bits}(q)$ le nombre de bits de A et q respectivement, si

$$\text{bits}(A) \geqslant \left\lfloor \frac{8 + \text{bits}(q)}{2} \right\rfloor$$

alors $A > 4\sqrt{q}$.

Démonstration. Remarquons que les entrées étant non signées, on a $\text{bits}(A) = 1+\lfloor \log_2(A) \rfloor$ et $\text{bits}(q) = 1 + \lfloor \log_2(q) \rfloor$. Ainsi, si la condition $\text{bits}(A) \geqslant \left\lfloor \frac{8+\text{bits}(q)}{2} \right\rfloor$ est vérifiée, cela implique que

$$\log_2(A) \geqslant \text{bits}(A) - 1 \geqslant \left\lfloor \frac{8 + \text{bits}(q)}{2} \right\rfloor - 1 > \frac{8 + \text{bits}(q)}{2} - 2 = 2 + \frac{\text{bits}(q)}{2} \geqslant 2 + \frac{\log_2(q)}{2}$$

d'où $A > 4\sqrt{q}$. □

Estimons la complexité de cet algorithme. Pour commencer, la condition $A \leqslant 4\sqrt{q}$ peut être vérifiée en $\mathrm{O}(1)$ opérations d'après ce qui précède. Ensuite, chaque boucle se décompose en plusieurs étapes :

- Le calcul de $\psi_\ell(x)$ se fait en $\mathrm{O}(\log \ell)$ multiplications de polynômes de $\mathbb{F}_q[x]$ d'après ? ? ?, qui ont un degré en $\mathrm{O}(\ell^2)$, soit en tout $\mathrm{O}\left(\ell^2 (\log \ell)(\log q)^2\right)$ opérations élémentaires.
- La multiplication dans $\mathcal{R}_{E,\ell}$ nécessite $\mathrm{O}(\ell^4)$ multiplications dans $\mathbb{F}_q$ et que la multiplication dans $\mathbb{F}_q$ a une complexité en $\mathrm{O}\left((\log q)^2\right)$, cette opération nécessite donc $\mathrm{O}\left(\ell^4 (\log q)^2\right)$ opérations élémentaires.
- Le calcul de $x^{q^2}, y^{q^2}, x^q, y^q$ se fait en $\mathrm{O}(\log q)$ multiplications dans $\mathcal{R}_{E,\ell}$ par exponentiation rapide.
- L'addition dans $E(\mathcal{R}_{E,\ell})$ se fait en $\mathrm{O}(1)$ multiplications dans $\mathcal{R}_{E,\ell}$.

Ainsi chaque boucle correspondant au nombre premier ℓ nécessite $\mathrm{O}\left(\ell^4 (\log q)^3\right)$ opérations élémentaires. Reprenant les notations de la section *§3.3*, toutes les boucles réunies nécessitent

$$\sum_{j=1}^{g(q)+1} \mathrm{O}\left(\ell^4 (\log q)^3\right) = \mathrm{O}\left((\log q)^3 g(q)^5\right)$$

opérations élémentaires. Reste à estimer $g(q)$, on a $\vartheta(x) \underset{x \to +\infty}{\sim} x$ d'après le théorème des nombres premiers donc

$$q \approx \prod_{j=1}^{g(q)} \ell_j = e^{\vartheta(\ell_{g(q)})} \approx e^{\ell_{g(q)}} \approx e^{g(q) \log g(q)}$$

d'où $g(q) \approx \frac{\log(q)}{\log \log(q)}$. On obtient une complexité totale de $\mathrm{O}\left(\frac{(\log q)^8}{(\log \log q)^3}\right)$ pour l'ensemble des boucles, ce qu'on majore généralement par $\mathrm{O}\left((\log q)^8\right)$. Comme le théorème des restes chinois nécessite $g(q)$ modulos qui valent au plus $\ell_{g(q)} \approx g(q) \log g(q)$ donc de taille en binaire de l'ordre de $\log g(q)$, l'appel final du théorème des restes chinois a une complexité en $\mathrm{O}\left(g(q)(\log g(q))^2\right)$ avec une implémentation naïve, qui est bien inférieure à un $\mathrm{O}\left((\log q)^8\right)$. La complexité totale de l'algorithme de Schoof est donc $\mathrm{O}\left((\log q)^8\right)$, ce qui est polynomiale.

6 Amélioration du temps de calcul

6.1 Polynômes de division universels

Une partie non-négligeable de notre algorithme principal consiste à calculer les polynômes de division $\psi_\ell(x)$ à chaque nouveau nombre premier. Du fait qu'il ne s'agisse que d'un intermédiaire de calcul et que les nombres premiers restent petits, on pourrait étudier la possibilité de les pré-calculer. Cependant, les polynômes de division dépendent complètement de la courbe elliptique à laquelle ils sont associés et ne peuvent donc pas être pré-calculés en toute généralité. Heureusement, leurs coefficients sont polynomiaux en les coefficients a et b de l'équation de Weierstrass de leur courbe elliptique.

Définition 6.1. Soit $(\Psi_n)_{n\geqslant 0}$ la suite à valeurs dans $\mathbb{Z}[x, y, a, b]$ définie par

$$\begin{aligned}
\Psi_0 &= 0\\
\Psi_1 &= 1\\
\Psi_2 &= 2y\\
\Psi_3 &= 3x^4 + 6ax^2 + 12bx - a^2\\
\Psi_4 &= 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3)
\end{aligned}$$

puis, par récurrence,

$$\begin{aligned}
\Psi_{2m} &= \frac{\Psi_m}{2y}\left(\Psi_{m+2}\Psi_{m-1}^2 - \Psi_{m-2}\Psi_{m+1}^2\right)\\
\Psi_{2m+1} &= \Psi_{m+2}\Psi_m^3 - \Psi_{m-1}\Psi_{m+1}^3
\end{aligned}$$

On obtient immédiatement par récurrence la proposition suivante :

Proposition 6.2. La suite $(\Psi_n)_{n\geqslant 0}$ est bien définie et, pour toute courbe elliptique E sur $\mathbb{F}_q$ d'équation de Weierstrass $y^2 = x^3 + a_0x + b_0$, on a

$$\psi_{E,n}(x, y) = \overline{\Psi_n}(x, y, a_0, b_0) \text{ dans } \mathbb{F}_q[x, y]$$

où $f \mapsto \overline{f}$ désigne l'unique morphisme d'anneaux $\mathbb{Z}[x, y, a, b] \longrightarrow \mathbb{F}_q[x, y, a, b]$ *i.e* la réduction modulo p de tous les coefficients.

En supposant que q s'écrive sur au plus 512 bits, on pourrait pré-calculer les polynômes

$$\Psi_\ell(x, y, a, b) \pmod{y^2 - x^3 - ax - b}$$

pour les 45 premiers nombres premiers et stocker le résultat dans un fichier binaire. À chaque itération de la boucle, on pourrait récupérer le polynôme d'indice le nombre premier correspondant puis réduire ses coefficients et enfin l'évaluer en les paramètres de la courbe elliptique en entrée.

Estimons la taille qu'aurait un tel fichier binaire. Supposons que les coefficients de Ψ_ℓ peuvent être encodés sur b bits pour $\ell \leqslant 45$, alors notant N_ℓ le nombre de coefficients de $\Psi_\ell(x, y, a, b) \pmod{y^2 - x^3 - ax - b}$, le nombre de bits contenus dans ce fichier binaire est de l'ordre de

$$\sum_{j=1}^{45} bN_{\ell_j}$$

où ℓ_j désigne le j-ème nombre premier impair.

Proposition 6.3.

$$N_\ell \approx \frac{\ell^2}{48}.$$

Démonstration. On pondère le degré sur $\mathbb{Z}[x,y,a,b]/(y^2-x^3-ax-b)$ par

$$\deg(x)=2, \deg(y)=3, \deg(a)=4, \deg(b)=6.$$

Ceci implique de par la relation de récurrence vérifiée par $(\Psi_n)_{n\geqslant 0}$ que $\Psi_\ell(x,y,a,b) \pmod{y^2-x^3-ax-b}$ est un polynôme homogène de $\mathbb{Z}[x,a,b]$ de degré total ℓ^2-1 pour ℓ impair, ainsi

$$N_\ell = \#\{(i,j,k)\in\mathbb{N}^3, 2i+4j+6k=\ell^2-1\}.$$

Comme ℓ est impair, à chaque couple $(j,k)\in\mathbb{N}^2$ vérifiant $4j+6k\leqslant \ell^2-1$ est associé un unique entier $i\in\mathbb{N}$ tel que $2i+4j+6k=\ell^2-1$. On obtient donc

$$N_\ell = \sum_{i=0}^{\mathrm{O}(\ell^2)} \#\{(j,k)\in\mathbb{N}^2, 4j+6k\leqslant \ell^2-1\}$$

et ceci peut être approché par l'aire du domaine

$$\{(u,v)\in(\mathbb{R}^+)^2, 4u+6v\leqslant \ell^2-1\}$$

qui n'est autre que le triangle de sommets $(0,0)$, $\left(\frac{\ell^2-1}{4},0\right)$, $\left(\frac{\ell^2-1}{6}\right)$. Ceci donne

$$N_\ell \approx \frac{\frac{\ell^2}{4}\times\frac{\ell^2}{6}}{2} = \frac{\ell^2}{48}$$

□

On a donc

$$\sum_{j=1}^{45} N_{\ell_j} \approx \sum_{j=1}^{45} \frac{\ell_j^2}{48} \approx \frac{1}{48}\sum_{j=1}^{45} j^2(\log j)^2 \approx \frac{1}{48}\int_1^{45} t^2(\log t)^2 \mathrm{d}t \approx 8\cdot 10^3.$$

Si les coefficients de $\mathbb{Z}[x,y,a,b]$ avaient une taille contrôlable, cette méthode serait envisageable. Malheureusement, la taille des coefficients de Ψ_n explosent complètement, par exemple les coefficients de Ψ_{23}, 23 étant le 8-ème nombre premier impair, doivent être stockés sur 191 bits !

Cette idée de pré-calcul doit donc être abandonnée en toute généralité. Remarquons tout de même qu'en cryptographie, on a tendance à fixer une courbe elliptique une bonne fois pour toute, auquel cas on peut simplement pré-calculer les polynômes de division universels usuels, le fait qu'ils soient à coefficients dans $\mathbb{F}_q$ assure un contrôle sur la taille des coefficients.

6.2 Identification du goulot d'étranglement, l'astuce de Atkin & Elkies

Au vu du calcul de la complexité de l'algorithme de Schoof effectué à la section *§5*, on voit que le goulot d'étranglement se situe au niveau de la multiplication dans les anneaux de ℓ-torsion générique $\mathcal{R}_{E,\ell}$. En examinant le partage du temps de calcul de l'algorithme avec `gprof` on se rend compte que la multiplication dans $\mathcal{R}_{E,\ell}$ occupe un tiers du temps de calcul.

La cause de ce phénomène est le fait que $\deg\psi_\ell = \mathrm{O}(\ell^2)$. L'idée de Atkin & Elkies est de n'utiliser que des nombres premiers ℓ pour lesquels ψ_ℓ se factorise, on ne travaille donc jamais explicitement dans l'anneau $\mathcal{R}_{E,\ell}$ mais dans des anneaux plus petits grâce au théorème des restes chinois. Cela a donnée naissance à l'algorithme SEA qui a une complexité en $\mathrm{O}\left((\log q)^4\right)$. Nous n'avons pas implémenté cet algorithme.

Références