Semaine d'Etude Maths-Entreprises 8 : Caractérisation de courbes elliptiques selon leurs capacités cryptographiques

Sylvain Arnt, Nadia Bernard, Djamal Foukrach, Loubna Ghammam, Alaa Jamal Eddine, Valentin Samoyeau, Christophe Tran

Caractérisation de courbes elliptiques selon leurs capacités cryptographiques.

Sylvain ARNT Nadia BERNARD Djamal FOUKRACH
Loubna GHAMMAM Alaa JAMAL EDDINE
Valentin SAMOYEAU Christophe TRAN

VIIIième SEMAINE D'ETUDE MATHS-ENTREPRISE

Contents

1 Résumé

Dans le cadre de la 8ième semaine d'étude maths-entreprise, SSL Europa nous a demandé de comparer les performances et les propriétés des protocoles cryptographiques sur le groupe multiplicatif des entiers modulo N d'une part, et le groupe des points d'une courbe elliptique d'autre part. Durant cette semaine, nous avons donc voulu implémenter nous mêmes les diffiérentes fonctionnalités cryptographiques. Nous avons ainsi pu comparer les temps de calcul pour chacune. Pour ce faire, nous avons choisi le logiciel libre de calcul formel *Sage*.

Dans une première section nous rappellerons les enjeux de la cryptographie moderne de manière à en dégager les critères de comparaison que nous utiliserons. Nous étudierons ensuite plus en détail la cryptographie RSA et la cryptographie sur courbes elliptiques.

2 Introduction

2.1 Qu'est-ce que la cryptographie?

Le but de la cryptographie est de développer les outils mathématiques permettant la communication confidentielle en présence d'adversaires malveillants. Ainsi, le schéma traditionnellement étudié en cryptographie est le suivant : Alice veut communiquer avec Bob, sachant qu'un adversaire, Charlie, écoute la ligne. Désormais, les protagonistes seront désignés par leurs initiales.

Exemples de situations concrètes :

- conversations téléphoniques,
- achats en ligne,
- identifications par badges électroniques...

Ainsi, les applications de la cryptographie sont extrêmement diverses, et les objectifs poursuivis seront nombreux. On peut néanmoins en présenter les principaux :

- *confidentialité* : C ne doit rien savoir du message de A;
- *authenticité* : C ne doit pas pouvoir se faire passer pour A auprès de B;
- *intégrité du message* : B doit pouvoir détecter toute altération du message effectuée par C.

Dans la suite, nous nous intéresserons donc à des protocoles de chiffrement/déchiffrement et de signatures.

2.2 Clef publique/clef privée

La plus ancienne forme de chiffrement est dite *symétrique* ou *à clef privée* (c'est par exemple le cas du chiffrement de César). Dans ce modèle, A et B se mettent au préalable d'accord sur une clef k qui doit rester secrète. Si A veut envoyer un message m à B, elle utilise le secret k et une fonction de chiffrement Enc pour calculer $c = Enc_k(m)$ qu'elle transmet à B. De son côté, B utilise le même secret k et la fonction de déchiffrement Dec pour retrouver $m = Dec_k(c)$.

Exemple : le chiffrement de Vigenère

La clef k est une suite de lettres. Chaque lettre de l'alphabet est reprśentée par un chiffre de 0 à 25. Le chiffrement se fait par un décallage des numéros

de chaque lettre du message vers la droite, le déchiffrement par un décallage équivalent vers la gauche. La première lettre de la clef indique le décallage à effectuer pour la première lettre du message, et ainsi de suite.
Par exemple, si la clef est le mot *clef*, il faudra décaller la première lettre du message à chiffrer de 2 crans, la seconde de 11, la troisième de 4, la quatrième de 5, puis la cinquième à nouveau de 2... Le message *salut* se chiffrerait donc en *ulpzv*.

L'avantage de tels protocoles est leur très grande efficacité. Par contre, un problème majeur posé par le chiffrement symétrique est la gestion des clefs. En effet, on a vu qu'A et B devaient se mettre d'accord au préalable sur une clef commune. Maintenant, si on veut mettre en place un réseau de N utilisateurs, on a besoin d'une clef secrète pour chacun des $N(N-1)/2$ couples d'utilisateurs. Il faut donc gérer l'échange et le stockage d'un grand nombre de clefs secrètes.
D'autre part, comme chaque clef secrète est partagée par deux entités différentes, il ne peut pas y avoir de protocoles de signature électronique basée sur la cryptographie symétrique.

Pour remédier à ces problèmes, Diffie, Hellman et Merkle inventèrrent en 1976 la notion de cryptographie *à clef publique*, ou cryptographie *asymétrique*. Dans ces schémas, chaque utilisateur génère un couple (e, d), où e est gardé secret tandis que d est rendu publique. Ainsi, il n'y a plus de problème d'échange de clefs, et chaque utilisateur n'a plus qu'une seule clef secrète à conserver. Si A veut envoyer un message à B, A utilise la clef publique d_B pour chiffrer son message, et B utilise son secret e_B pour déchiffrer. Si maintenant A veut de plus signer son message, elle utilisera sa clef secrète e_A. Bob vérifiera la validité de cette signature avec la clef publique d_A.

Les protocoles à clefs secrètes sont significativement plus lents que les protocoles à clefs publiques. C'est le prix à payer pour obtenir des fonctionnalités plus puissantes. En pratique, des systèmes hybrides bénéficiant des avantages des deux schémas sont donc souvent mis en place.
Dans ce rapport, les deux cryptosystèmes que nous mettons en concurrence (le RSA et la cryptographie sur courbes elliptiques) sont des cryptosystèmes à clef publique.

2.3 Sécurité, complexité et efficacité

En cryptographie moderne, diverses notions formellement bien définies de sécurité ont été développées. Elles se présentent sous forme d'un jeu pour l'attaquant (dans notre schéma traditionnel, C). Elles reposent sur le but fixé à l'attaquant (retrouver la clef secrète, déchiffrer un chiffré donné, ...) et les moyens qu'on lui autorise. Par exemple, dans certains jeux, l'attaquant

aura accès à un *oracle* qui pourra déchiffrer pour lui tout texte chiffré, hormis bien sûr celui soumis à l'attaquant.

Le domaine de la sécurité prouvée étant à la fois vaste et en dehors du sujet proposé, nous ne nous y sommes pas attardés. Dans la suite, nous ferons l'hypothèse que l'attaquant cherche à trouver la clef secrète. Nous avons étudié les niveaux standards de sécurité (128, 192, et 256 bits de sécurité). Autrement dit, nous avons donné à nos clefs une taille suffisante pour que les retrouver prenne 2^{128}, 2^{192} ou 2^{256} calculs à l'attaquant.

Une fois la taille de la clef fixée (et avec elle celle de toutes les données du système), nous voulons étudier la vitesse d'exécution de nos différrentes fonctions cryptographiques (génération de clefs, chiffrement, déchiffrement, signature, authentification). Pour cela, il est courant d'utiliser la notation "en grand Oh" :

Définition 1. *Soient f et g deux fonctions allant d'un même ensemble E vers $\mathbb{R}$, avec $g > 0$. On note $f \in O(g)$ ou $f = O(g)$ s'il existe une constante $C > 0$ telle que*

$$\forall x \in E,\ |f(x)| \leq Cg(x).$$

Exemples :

- $f = O(1)$ signifie que f est bornée;
- si $x > 0$, pour tout $a, b > 0$, $x^a + log(x)^b = O(x^a)$.

La *complexité* d'un algorithme se mesure par rapport à la taille des données. Un entier N se représentant sur un ordinateur par approximativement $log_2(N)$ bits, on dira que la taille de N est en $O(log_2(N))$. Dorénavant, on notera simplement log pour log_2.

On distingue trois grandes classes de complexité : polynômiale, sous-exponentielle et exponentielle.

Définition 2. *Pour $N \geq 1$, $\beta > 0$ et $0 \leq \alpha \leq 1$, on note*

$$L_{\alpha,\beta}(N) = exp\left(\beta(logN)^{\alpha}(loglogN)^{1-\alpha}\right).$$

Quand on ne veut pas préxciser β, on note plus simplement

$$L_{\alpha}(N) = exp\left(O(1)(logN)^{\alpha}(loglogN)^{1-\alpha}\right).$$

Ainsi,

- si $\alpha = 0$, $L_{0,\beta}(N) = (logN)^{\beta}$: c'est la classe polynômiale;
- si $\alpha = 1$, $L_{1,\beta}(N) = (N)^{\beta}$: c'est la classe exponentielle;
- si $0 < \alpha < 1$, on est entre les deux : c'est la classe sous-exponentielle.

2.4 Quelques algorithmes utilisés

Dans le cadre de notre travail, nous avons eu besoin de certaines techniques algorithmiques, qui, bien qu'utilisées par les cryptographes, ne relèvent pas à proprement parler de cryptographie. Nous les présentons ici.

2.4.1 Exponentiation rapide

On se place sur un groupe G. On suppose que la loi de composition de G (que l'on notera multiplicativement) est connue et programmée (c'est une condition raisonnable si on veut faire de la cryptographie). On se donne un élément g de G, et on suppose que l'on veuille calculer efficacement g^k, pour k assez grand.

La méthode naïve consisterait à calculer d'abord $g^2 = g \cdot g$, puis $g^3 = g \cdot g^2$ et ainsi de suite jusqu'à $g^k = g^{k-1} \cdot g$. Mais cela nécessiterait $k - 1$ opérations dans G, et nous donnerait donc un algorithme exponentiel en log(k).

Imaginons par exemple que $k = 11$. L'écriture de 11 en base 2 donne 1101, donc

$$g^{11} = g^{2(2(2+0)+1)+1},$$

ce qui signifie qu'on peut calculer g^{11} en calculant successivement $g^2 = g \cdot g$, puis $g^5 = g^2 \cdot g^2 \cdot g$, et enfin $g^{11} = g^5 \cdot g^5 \cdot g$.

On obtient ainsi l'algorithme d'*exponentiation rapide*, qui calcule g^k en faisant un nombre linéraire en $log(k)$ opérations dans G. Nous avons implémené cette méthode sur le groupe des points d'une courbe elliptique (voir section 5.2).

2.4.2 Tri rapide

Une situation courante en cryptographie est la *recherche de collisions* : on dispose de deux listes L_1 et L_2 de même taille N, on cherche à retrouver des indices i et j tels que $L_1[i] = L_2[j]$.
Encore une fois, la méthode naturelle est inefficace : si on essaye de comparer successivement chaque élément de la liste L_1 avec chaque élément de la liste L_2, on obtient un algorithme en temps quadratique en N.
Par contre, si les éléments de nos listes appartiennent à un ensemble muni d'une relation d'ordre total (cela sera effectivement le cas dans les situations que nous allons étudier), la meilleure méthode consiste à d'abord *trier* intelligemment les deux listes L_1 et L_2.

Il existe plusieurs algorithmes de tri en $O(Nlog(N))$. Nous présentons ici le *tri fusion*. C'est un algorithme récursif.

Supposons que l'on dispose de deux petites listes triées $a_1 \leq \cdots \leq a_n$ et $b_1 \leq \cdots \leq b_m$. Il est alors facile de les fusionner pour obtenir une grande liste c triée.
En effet, on se donne trois pointeurs i, j et k pour les listes a, b et c respectivement. On compare a_i et b_j, c_k prend la valeur du plus petit des deux, puis on incrémente k et i ou j.
Ainsi, partant d'une liste $t_1, \ldots, t_N$ à trier, l'algorithme consiste à découper cette liste en deux sous listes $t_1, \ldots, t_{\lfloor N/2 \rfloor}$ et $t_{\lfloor N/2 \rfloor + 1}, \ldots, t_N$, à trier récursivement chacune de ces listes, et à fusionner le résultat selon la méthode expliquée ci-dessus.

3 Cryptosystème RSA

C'est le cryptosystème asymétrique le plus utilisé au monde. Pour plus de simplicité, nous présentons ici la versions "scolaire" du RSA (c'est-à-dire celle communément décrite dans les ouvrages). Dans la pratique, bien que les principes restent les mêmes, ce n'est pas ainsi que le RSA est implémenté.

3.1 Génération de clefs

L'utilisateur choisit aléatoirement deux grands nombres premiers p et q de même taille, et calcule $N = pq$. Soit ϕ l'indicatrice d'Euler, on rappelle que $\phi(N) = (p-1)(q-1)$ est le nombre d'entiers compris entre 1 et $N-1$ qui soient premiers à N. B choisit maintenant e qui soit premier à $\phi(N)$, et calcule d son inverse modulo $\phi(N)$:

$$ed \equiv 1 \pmod{\phi(N)}.$$

La clef publique est alors le couple (N, e), et la clef secrète est d.

3.2 Chiffrement

Dans le cryptosystème RSA, un message m est un entier modulo N. Pour chiffrer ce message, A utilise la clef publique de B pour calculer et envoyer :

$$c \equiv m^e \pmod{N}.$$

Comme les exposants e et d sont inverses l'un de l'autre modulo $\phi(N)$, on a pour tout x dans $\mathbb{Z}/N\mathbb{Z}$, $x^{de} \equiv x \pmod{N}$.
Pour dćhiffrer, B peut donc utiliser sa clef privée :

$$m \equiv c^d \pmod{N}.$$

La sécurité de ce protocole de chiffrement repose sur la difficulté de calculer des racines e-ième de c modulo N.

3.3 Signature

On suppose que A et B travaillent tous deux modulo le même entier N, et disposent chacun d'une clef publique et d'une clef secrète (respectivement (e_A, d_A) et (e_B, d_B)).
A veut signer le message qu'elle envoie à B. Après avoir calculé comme précédemment $c \equiv m^{e_B} \pmod{N}$, elle signe son chiffré en calculant

$$s \equiv c^{d_A} \pmod{N},$$

et envoie (s, c) à B.
Pour vérifier la validité de la signature, B utilise alors la clef publique de A : la signature est acceptée si et seulement si $s^{e_A} \equiv c \pmod{N}$. Si c'est bien le cas, B peut alors retrouver le message clair m comme précédemment.

Maintenant, l'attaquant C ne peut pas signer à la place d'A n'importe quel message c, à moins de savoir calculer des racines e_A-ième de c modulo N.

3.4 Sécurité

Du point de vue de l'attaquant, si la factorisation de N est connue, connaissant e, il est facile de trouver l'exposant secret d. Réciproquement, connaissant le secret d, on peut factoriser N en temps polynomial. Ainsi, trouver la clef secrète est aussi difficile que factoriser le module RSA N. Cela ne signifie pourtant pas que C soit obligé de factoriser N pour déchiffrer c. On supposera néanmoins que c'est le cas. On dira donc que la sécurité du RSA repose sur la difficulté du problème de factorisation de grands entiers.

Asymptotiquement, l'algorithme de factorisation le plus rapide connu est le *crible des corps de nombres* (en anglais number field sieve, ou NFS). Sa complexité est en $L_{1/3}(N)$, donc sous-exponentielle. La taille des nombres premiers p et q selon le niveau de sécurité exigé est donc :

Niveau de sécurité (en bits)	128	192	256
Taille des clés RSA	3072	7680	15360

3.5 Efficacité

La génération de clefs RSA est très rapide : en effet, il faut pour cela générer aléatoirement des grands nombres premiers, trouver e premier à $(p-1)(q-1)$ puis calculer son inverse d. Chacune de ces opérations est très rapide (le temps se compte en secondes).

Une fois le système mis en place, l'exécution des fonctions de chiffrement, déchiffrement et signature repose sur l'exponentiation rapide (voir section 2.4.1) et est très rapide également.

4 Cryptographie reposant sur le problème du logarithme discret

On énonce ici certains protocoles de cryptographie alternatifs au protocole RSA. On se place sur un groupe général $\mathbb{G}$ abélien d'ordre N que l'on note multiplicativement (dans la section 5, on se placera sur le groupe des points d'une courbe elliptique qui sera noté additivement).

Nous verrons dans la suite que la sécurité des protocoles présentés ici repose sur la difficulté du *problème du logarithme discret* (en anglais discrete logarithm problem ou DLP) sur G :

$$\text{Connaissant } g, h \in \mathbb{G}, \text{ trouver un entier } x \text{ tel que } h = g^x.$$

Nous désirons ainsi un groupe sur lequel l'arithmétique soit rapide (pour que les fonctions de chiffrement, déchiffrement,... soient rapides) mais le DLP difficile. Par exemple, le groupe $(\mathbb{Z}/N\mathbb{Z}, +)$ n'est absolument pas utilisable : s'il est vrai que l'arithmétique y est rapide, le DLP y est également trivial. Le groupe $((\mathbb{Z}/N\mathbb{Z})^*, \times)$ semble déjà plus convainquant.

On reporte la discussion sur le choix du groupe à la section 4.4, et on suppose pour le moment que l'on dispose d'un groupe convenable.

4.1 Echange de clefs Diffie-Hellman

A et B veulent se mettre d'accord sur une clef commune, qui pourra par exemple servir de clef pour un protocole de chiffrement symmétrique. Les données publiques sont le groupe G, l'élément $g \in \mathbb{G}$ et son ordre N. Le protocole d'échange se déroule alors comme suit :

1. A choisit aléatoirement un entier x_A entre 1 et $N-1$, et envoie g^{x_A} à B.
2. De même, B choisit aléatoirement un entier x_B entre 1 et $N-1$, et envoie g^{x_B} à A.
3. Les deux protagonistes peuvent alors calculer dans leur coin l'élément commun $g^{x_A x_B}$: A en faisant $(g^{x_B})^{x_A}$, B en faisant $(g^{x_A})^{x_B}$.

L'attaquant C devra avec g, g^{x_A} et g^{x_B} trouver $g^{x_A x_B}$: c'est le problème Diffie-Hellman que l'on croit équivalent à DLP dans la plupart des groupes utilisés en cryptographie.

4.2 Chiffrement et signature ElGamal

On présente maintenant un protocole de chiffrement asymétrique. A veut envoyer un message à B. Le message se m présente sous la forme d'un

élément de G.
La clef publique de B est un couple $(g, h = g^x)$. Sa clef privée est x. Le protocole se déroule comme suit :

1. A choisit aléatoirement un entier k entre 1 et $N-1$. Elle calcule et envoie à B:
$$c_1 = g^k,\ c_2 = h^k \cdot m.$$
2. Pour déchiffrer, B calcule
$$c_2 \cdot c_1^{-x} = m.$$

Tout comme le chiffrement RSA, le chiffrement ElGamal se traduit naturellement en un protocole de signature.

4.3 Pourquoi les courbes elliptiques?

On s'intéresse maintenant au choix du groupe $\mathbb{G}$ sur lequel travailler. Cela revient à se demander sur quel groupe le DLP est difficile.
Sur un groupe générique (i.e. sur lequel on ne demande aucune information) $\mathbb{G}$ d'ordre N on dispose d'algorithmes dits quadratiques pour attaquer le DLP. Leurs coûts en temps est en $O(\sqrt{N})$ (en fait, en utilisant le théorème des restes chinois, en $O(\sqrt{p})$, où p est le plus grand nombre premier divisant N), c'est-à-dire en coût exponentiel. Nous présenterons un de ces algorithme, Baby-Step-Giant-Step, dans le cadre des courbes elliptiques, en 5.3.
Sur les groupes multiplicatifs des corps finis, il existe des algorithmes en temps sous-exponentiel pour traiter le DLP (ce sont les méthodes NSF déjà évoquées pour RSA). Par contre, sur les groupes des points d'une courbe elliptique (modulo certaines restrictions explicitées en 5.4), seules les attaques génériques peuvent être utilisés contre le DLP.

5 Cryptographie sur courbes elliptiques

Il existe de nombreuses références sur les courbes elliptiques. Ici, nous ne désirons pas passer trop de temps à définir formellement l'objet mathématique.

5.1 Définition, propriétés

Soit $\mathbb{K}$ un corps, $\overline{\mathbb{K}}$ une clôture algébrique. Une *courbe elliptique* $\mathcal{E}$ sur $\mathbb{K}$ est définie comme l'ensemble des solutions dans l'espace projectif $\mathbb{P}_2(\overline{\mathbb{K}})$ de l'*équation de Weierstrass* homogène :
$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3,$$

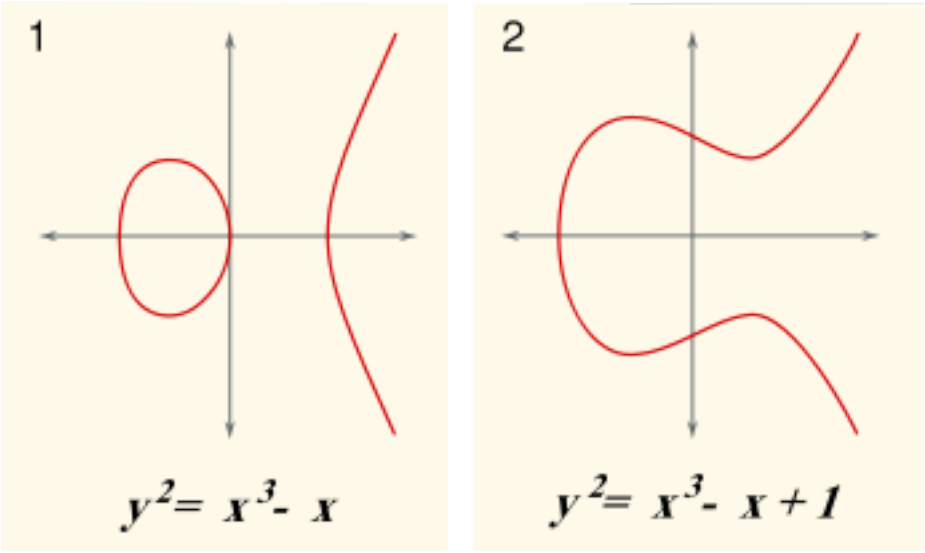
avec les coefficients a_i dans $\mathbb{K}$.
La courbe $\mathcal{E}$ doit être lisse : ceci signifie que si on écrit l'équation de $\mathcal{E}$ sous la forme $F(X, Y, Z) = 0$, en chaque point de $\mathcal{E}$ une au moins des dérivées de F doit être non nulle.
On voit facilement qu'il existe un seul point de $\mathcal{E}$ vérifiant $Z = 0$: c'est le point $\mathcal{O} = [0 : 1 : 0]$. On peut donc se placer dans la carte affine $Z \neq 0$ et effectuer le changement de variables $x = X/Z$ et $y = Y/Z$:

Définition 3. *Une* courbe elliptique $\mathcal{E}$ *sur* $\mathbb{K}$ *est l'ensemble des solutions* (x, y) *de*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

auquel on rajoute un point $\mathcal{O}$ *appelé point à l'infini.*

Voice deux exemples de courbes dessinées sur $\mathbb{R}$:



Ici et pour le reste du rapport, $\mathbb{F}_q$ sera un corps fini de caractéristique p, avec $q = p^\alpha$.

Proposition 1. *Si* $p \neq 2, 3$*, un changement affine permet de transformer l'ï¿?quation de* $\mathcal{E}$ *en*

$$y^2 = x^3 + Ax + B.$$

Une telle transformation laisse $\mathcal{O}$ *invariant.*

Remarque : Il existe également des modèles de courbes pour les caractéristiques 2 et 3. Nous ne les explicitons pas ici.

Il est temps de voir la principale propriété des courbes elliptiques qui nous intéresse : le fait que l'on puisse les munir d'une loi d'addition de faï¿?on à en faire un groupe.

5.2 Arithmétique

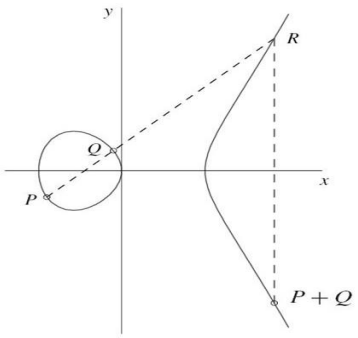
Proposition 2. *Soit $\mathcal{E} \subset \mathbb{P}_2$ une courbe elliptique. L'intersection de $\mathcal{E}$ avec toute droite projective $L \subset \mathbb{P}_2$ est composée d'exactement trois points comptés avec multiplicité.*

On définit donc sur $\mathcal{E}$ la loi de composition suivante :

Loi de composition interne : Soient $P, Q \in \mathcal{E}$, soit L la droite passant par P et Q (ou la tangente à $\mathcal{E}$ en P si $P = Q$), soit R le troisième point d'intersection de L et $\mathcal{E}$. Soit L' la droite passant par R et $\mathcal{O}$. Alors L' coupe $\mathcal{E}$ en un troisième point que l'on note $P + Q$.

Proposition 3. *$\mathcal{E}$ munie de la loi de composition ainsi définie est un groupe commutatif. L'élément neutre en est le point à l'infini $\mathcal{O}$.*

Voici un dessin représentant cette loi de groupe :



Addition

Remarques :

- Si on travaille avec une courbe dont l'équation a été simplifiée en $y^2 = x^3 + Ax + B$, l'opération $P \mapsto -P$ consiste simplement à prendre le symmétrique de P par rapport à la droite des abscisses.

- Pour calculer la loi de groupe sur $\mathcal{E}$, il suffit donc de calculer l'équation de $\mathcal{E}$, puis de trouver les coordonnées de R . Les coordonnées de $P+Q$ sont alors $(x_R, -y_R)$.

Nous avons programmé cette loi de groupe sur Sage :

```
def loidegroupe(F,a,b,x_A,y_A,x_B,y_B):                    # F corps; a,b paramètres de la courbe; A=(x_A,y_A) B=(x_B,y_B)

    if F(x_A)==F(0) and F(y_A)==F(0) :                     # Cas: A = élément neutre (0,0)
         x_C=x_B; y_C=y_B

    elif F(x_B)==F(0) and F(y_B)==F(0) :                   # Cas: B = élément neutre (0,0)
         x_C=x_A; y_C=y_A

    elif F(x_A)==F(x_B) and F(y_A)==F(-y_B):               # Cas: A = -B (pour la loi du groupe)
        x_C=F(0); y_C=F(0)

    elif F(x_A)!=F(x_B) and F(y_A)!=F(y_B):
        lambda_1=(y_B-y_A)/(x_B-x_A);mu_1=y_A-lambda_1*x_A;  # Cas: A et B différents
        x_C=lambda_1^2-x_A-x_B; y_C=-lambda_1*x_C-mu_1

    else:
        lambda_2=(3*x_A^2+a)/(2*y_A); mu_2=y_A-lambda_2*x_A; # Cas A = B
        x_C=lambda_2^2-x_A-x_B; y_C=-lambda_2*x_C-mu_2

    return([F(x_C),F(y_C)])                                # Renvoi A + B
```

Puis l'exponentiation rapide (voir section 2.4.1) pour calculer le multiple d'un point :

```
def exporapide(F,a,b,x_A,y_A,n):                    # Exponentielle rapide: calcul de nA pour A=(x_A,y_A)

    if n==0:                                        # Cas n = 0, renvoi l'élément neutre (0,0)
        return(0,0)

    x_T = F(x_A); y_T = F(y_A);
    L = basedeux(n);
    k = len(L);

    for i in range(1,k):                            # Calcul de nA suivant la décomposition binaire de n

        [x_T,y_T]=loidegroupe(F,a,b,x_T,y_T,x_T,y_T)

        if L[k-i-1]==1:
            [x_T,y_T]=loidegroupe(F,a,b,x_T,y_T,x_A,y_A)

    return([x_T,y_T])                               # Renvoi nA (= A+A+...+A n fois)
```

Un mot sur le cardinal du groupe que nous avons obtenu :

Proposition 4. *Si $\mathbb{E}$ est une courbe elliptique définie sur $\mathbb{F}_q$, alors*

$$|\sharp\mathbb{E}(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}.$$

Remarque : Encore une fois, les algorithmes présentés ici sont des algorithmes scolaires : ainsi, la loi de groupe que nous avons programmée demande une inversion dans le calcul de λ, le coefficient directeur de la droite L. Sachant que le coût d'une inversion sur corps fini est environ 10 celui d'une multiplication, il faut absolument les éviter. Une solution est d'utiliser une représentation projective des points. Il existe également d'autres modèles des courbes elliptiques que le modèle de Weierstrass : ainsi, le modèle d'Edwards donne une arithmétique plus rapide.

5.3 Baby-Step-Giant-Step sur courbes elliptiques

Nous avons vu (section 4.3) que les meilleurs algorithmes pour casser le DLP sur une courbe elliptique sont les algorithme génériques :

Soient $P, Q \in \mathbb{E}(\mathbb{F}_q)$, avec P d'ordre N. On peut trouver k tel que $Q = kP$ en $O(\sqrt{N})$ opérations sur $\mathbb{E}$.

Nous présentons ici l'algorithme Baby Step/Giant Step.
Si nous voulons trouver k, la solution naïve est de tester tous les entiers entre 1 et $N-1$. On peut diviser le travail en deux de la manière suivante :

on pose $m = \lceil \sqrt{N} \rceil$, et on décompose k en $k = k_1 + k_2 m$, avec maintenant k_i plus petits que m. L'équation $Q = kP$ devient maintenant

$$Q + k_1(-P) = k_2(mP).$$

Trouver k est alors équivalent à trouver k_1 et k_2. Pour ce faire, on calcule tous les $Q + i(-P)$ et les $j(mP)$, on les range dans deux listes, puis on recherche une collision entre elles (voir section 2.4.2). Voici le code Sage produit :

```
def bsgs(F,a,b,x_A,y_A,x_B,y_B):                    # Baby step - Giant step: détermine k tel que kA = B

    E=EllipticCurve(F,[a,b]);
    n=E(x_A,y_A).order();                           # n = ordre de A dans le groupe

    m=ceil(sqrt(n));                                # on recherche k sous la forme k=k1+k2*m
                                                    # k1: baby step; k2: giant step

    mA = exporapide(F,a,b,x_A,y_A,m);

    L1=[]; L2=[];

    for i in range(0,m+1):                          # calcul de k1(-A)+B et k2(mA)
        iA = exporapide(F,a,b,x_A,-y_A,i);
        L1 = L1+[[loidegroupe(F,a,b,iA[0],iA[1],x_B,y_B),i]];
        L2 = L2+[[exporapide(F,a,b,mA[0],mA[1],i),i]];

    L1.sort(); L2.sort();

    j1=0; j2=0;
    while L1[j1][0][0]!=L2[j2][0][0]:               # recherche de k1 et k2 tels que k1(-A)+B =k2(mA)
        if L1[j1][0][0]<L2[j2][0][0]:
            j1 = j1+1;
        else:
            j2 = j2+1;

    k1=L1[j1][1];
    if L1[j1][0][1]==L2[j2][0][1]:
        k2=L2[j2][1];
    else:
        k2=-L2[j2][1];

    return((k1+k2*m)%n)                             # renvoi k
```

5.4 Choisir sa courbe

Nous ne parlerons pas couplages dans ce rapport. Il s'agit de méthodes cryptographiques, utiles autant pour le construction de cryptosystèmes avec des propriétés inédites et intéressantes (tels les schémas de signature basés

sur l'identité) que pour l'attaque du DLP. Très grossièrement décrit, un couplage permet de transporter le DLP sur la courbe $\mathbb{E}(\mathbb{F}_q)$ vers une extension de corps $\mathbb{F}_{q^k}$. Sans plus d'explication, les couplages sont à l'origine du second critère de sélection de courbes que nous donnons :

Critères pour choisir une courbe $\mathbb{E}$ définie sur un corps $\mathbb{F}_q$ avec $\sharp\mathbb{E}(\mathbb{F}_q) = n$:

- Le groupe $\mathbb{E}(\mathbb{F}_q)$ doit posséder un point P d'ordre r premier assez grand pour rendre l'attaque Baby Step/Giant Step trop coûteuse.
- Le plus petit entier k tel que $q^k \equiv 1 \pmod{n}$ doit être assez grand pour que le NFS sur $\mathbb{F}_{q^k}$ soit trop coûteuse.

On obtient alors le tableau suivant :

Niveau de sécurité (en bits)	128	192	256
Taille de r	256	384	512
Taille de q^k	3072	7680	15360

5.5 Efficacité

La génération de courbes adaptées à la cryptographie peut se faire de deux façons différentes :

- Choisir q, générer une courbe $\mathcal{E}$ sur $\mathbb{F}_q$ aléatoire, calculer $n = \sharp E$, et recommencer jusqu'à obtenir une courbe vérifiant les deux critères énoncés.
- Choisir les paramètres q, n et k, et construire une courbe $\mathbb{E}(\mathbb{F}_q)$ avec cardinal n et k comme dans le critère 2 par une méthode CM (Complex Multiplication).

Il existe des algorithmes pour calculer l'ordre d'une courbe, et pour construire une courbe avec un ordre prescrit. Sans rentrer dans les détails, l'établissement d'une courbe adaptée prend significativement plus de temps que la génération de clefs RSA. Ainsi, sur Sage, il nous a fallu environ 30 minutes pour générer 10 courbes admettant un sous groupe d'ordre premier plus grand que 2^{200}.

Plus important, le temps d'exécution des fonctions cryptographiques repose sur l'efficacité de l'arithmétique sur la courbe. Comme nous l'avons souligné en 5.2, cette efficacité dépend de la représentation choisie de la courbe.
Ainsi, en prenant compte des résultats obtenus par Oumar DIAO et Emmanuel FOUOTSA sur l'utilisation des coordonnées thêta sur les courbes d'Edwards, un calcul approximatif au tableau nous a montré que les temps d'exécution des fonctions cryptographiques sur courbes elliptiques étaient comparables avec ceux des fonctions cryptographiques pour le RSA.

6 Conclusion

En terme de coûts, l'utilisation des courbes elliptiques est préférable à l'utilisation des modules RSA : ainsi que nous l'avons vu, les coûts en temps sont essentiellement les mêmes alors que la taille des données est 12 fois inférieure pour la cryptographie sur courbes elliptiques que pour la cryptographie RSA en 128 bits de sécurité (et ce ratio augmente avec le niveau de sécurité).
D'autre part, bien que cela n'ait pas été abordé dans ce travail, l'utilisation des courbes et des couplages sur ces courbes permettent d'obtenir des fonctionnalités cryptographiquement intéressantes et par ailleurs inédites.

Dans ce rapport, nous avons donné les critères permettant de choisir la courbe que l'on veut utiliser. Plus que les données chiffrées brutes, nous avons données l'explication de ces critères. En cas de nouvelles percées dans la recherche cryptographique, l'entreprise SSL Europa pourra ainsi faire évoluer ces chiffres.

Dans ce rapport, nous avons souligné l'importance du choix de la bonne représentation de la courbe, et de l'implémentation intelligente de son arithmétique. Nous pouvons rappeler dans cette conclusion que l'arithmétique sur les corps finis mérite également une bonne implémentation. Il faudra regarder à l'impact de méthodes à la Karatsuba ou Toom-Cook, sur ordinateur et sur carte à puce.