CRYPTOGRAPHIE SUR LES CORBES ELLIPTIQUES

Olayinka S. Folorunso

20 Juin, 2014

Génie Informatique, ENSA Marrakech

Encadré par M. S Labhalla.

Introduction

Selon le principe de Kerckhoffs : **La sécurité d'un protocole cryptographique doit être garantie par les clés de chiffrement et déchiffrement utilisées, et non par la méthode de chiffrement.** Ce principe a donné naissance à la cryptographie moderne. Celle-ci se caractérise par le fait que les algorithmes de chiffrement et méthode de déchiffrement sont connus de tous, mais les clés utilisées sont partiellement, ou entièrement secrètes.

Les protocoles cryptographiques se divisent principalement en deux catégories. Les systèmes symétriques pour lesquels la clé servant au chiffrement et au déchiffrement est secrète ; et les protocoles à clé publique pour lesquels la clé de chiffrement est publique, et la clé de déchiffrement privée.

En cryptographie asymétrique, la clé publique permet de chiffrer un message, elle est connue de tous. Le déchiffrement d'un message ne peut se faire sans la clé secrète associée à la clé publique. Le couple (clé privée, clé publique) est construit de sorte qu'un protocole donné soit difficile à casser, c'est à dire retrouver la clé secrète à partir de la clé publique.

Cryptographie à clé publique se base sur la grande complexité des certaines problèmes Mathématiques. Les premières systèmes à clé publique ont été sécurisé en supposant sue c'est difficile de factoriser un grand nombre entier composé de deux ou plusieurs grands facteurs premiers. Pour les crypto-courbe-elliptique, on a supposé que c'est *impossible* de trouver le logarithme discret d'un élément d'une courbe elliptique par rapport à un point de référence connu (*publiquement*. Celui-ci est le Problème de Logarithme Discrète d'une Courbe Elliptique, *The Elliptic Curve Discrete Logarithm Problem ECDLP*.

L'Idée de crypto-courbe-elliptique dépend de la capacité de calculer une multiplication à point et l'incapacité de calculer le multiplicande, étant donné les points d'origine et de produits. La taille de la courbe décrit la complexité du problème. Le principal avantage d'ECC est une taille de clé plus petite, une réduction de taille dans le stockage et la transmission requis. Un crypto-courbe-elliptique pourrait fournir le même niveau de sécurité offert par un système RSA avec un grand module et une grande clé.

Table des matières

Chapitre 1

Pourquoi les Courbes Elliptiques

1.1 Cryptographie à clé publique

L'histoire de la cryptographie peut être divisée en deux époques : l'époque classique et l'époque moderne. Le tournant a eu lieu entre les deux en 1977, lorsque les deux l'algorithme RSA et l'algorithme d'échange de clés Diffie-Hellman ont été introduits. Ces nouveaux algorithmes ont été révolutionnaires parce qu'ils représentaient les premiers schémas cryptographiques viables où la sécurité est fondée sur la théorie des nombres ; il a été le premier à permettre la communication sécurisée entre deux parties sans un secret partagé. Cryptographie est allé d'être sur le transport en toute sécurité dictionnaires secrètes à travers le monde à être en mesure d'avoir une communication sécurisée entre prouvablement deux partis sans se soucier de quelqu'un qui écoute dans l'échange de clé.

La cryptographie moderne est fondée sur l'idée que la clé que vous utilisez pour crypter les données peut être rendue publique alors que la clé qui est utilisée pour déchiffrer vos données est gardé secret. En tant que tels, ces systèmes sont connus comme des systèmes cryptographiques à clé publique. La première, et encore plus largement utilisé de ces systèmes, qui est connu comme RSA - nommé d'après les initiales des trois hommes qui, le premier décrit publiquement l'algorithme : Ron Rivets, Adi Shamir, et Leonard Aleman.

Qu'est-ce que vous avez besoin pour un système cryptographique à clé publique au travail est un ensemble d'algorithmes qui est facile à traiter dans un sens mais difficile à l'autre sens. Dans le cas de RSA, l'algorithme multiplie deux nombres premiers. Si la multiplication est l'algorithme simple, son algorithme de pair difficile est de factoriser le produit de la multiplication dans ses deux premiers composants. Trouver une bonne paire d'algorithmes est essentiel pour établir un système de cryptographie à clé publique sécurisée.

Pour créer une paire de clés RSA, d'abord choisir au hasard les deux nombres premiers (p et q) pour obtenir le maximum ($n = p \cdot q$). Ensuite, prenez un numéro e, la clé publique tel que $1 < e < \varphi(n)$ et $(e, \varphi(n)) = 1$. Tant que vous savez les deux nombres premiers,

vous pouvez calculer une clé privée d correspondante de cette clé publique. Factoriser n en p et q, ses deux premiers composants nous permet de calculer la clé privée ($d = e^{-1} \pmod{\varphi}(n)$) de quelqu'un à partir de la clé publique et décrypter leurs messages privés.

Un message m chiffré est $m_c = m^e \pmod{n}$ et pour déchiffrer on a $m = m_c^d \pmod{n}$. Ça implique que $m^{ed} \equiv m \pmod{n}$. Pour vérifie on applique le théorème des restes chinois et le petit théorème de Fermat.

Théorème 1.1 *Théorème des restes chinois*

Soient $n_1, \cdots, n_k$ des entiers deux à deux premiers entre eux càd $(n_i, n_j) = 1, i \neq j$. Alors pour tous entiers $a_1, \cdots, a_k$, il existe un entier x unique modulo $n = \prod_{i=1}^{k} n_i$ tel que

$$x \equiv a_i \pmod{n}_i$$

Théorème 1.2 *Petit théorème de Fermat*

Si p est un nombre premier et si a est un entier non divisible par p, alors $a^{p-1} - 1$ est un multiple de p. Autrement dit,

$$a^{p-1} - 1 \equiv 0 \pmod{p} \iff a^{p-1} \equiv 1 \pmod{p} \iff a^p \equiv a \pmod{p}$$

On a que $n = p \cdot q$, alors selon le théorème des restes chinois pour montrer que $m^{ed} \equiv m \pmod{n}$ il faut montrer que $m^{ed} \equiv m \pmod{p}$ et que $m^{ed} \equiv m \pmod{q}$.
On a choisi e et d tel que $ed - 1 \equiv 0 \pmod{\varphi}(n)$ avec $\varphi(n) = (p-1)(q-1)$ ce qui implique qu'il existe h tel que

$$m^{ed} = m \cdot m^{ed-1} = m \cdot m^{h(p-1)(q-1)}$$

Pour montrer que $m^{ed} \equiv m \pmod{p}$, puis que p est premier, il y a deux possibilités, soit $(m, p) = p$ ou $(m, p) = 1$. Si $(m, p) = p$, alors $m^{ed} \equiv 0 \equiv m \pmod{p}$ et si $(m, p) = 1$, alors selon le petit théorème de Fermat $m^{ed} = m \cdot m^{h(p-1)(q-1)} \equiv m \cdot 1^{h(q-1)} \equiv m \pmod{p}$. On a montré que $m^{ed} \equiv m \pmod{p}$, pour montrer que $m^{ed} \equiv m \pmod{q}$, on suit les même étapes et on peut conclure que $m^{ed} \equiv m \pmod{n}$.
On peut aussi montrer avec le théorème d'Euler

Théorème 1.3 *Théorème d'Euler*

Soit n un entier strictement positif et a un entier premier avec n, alors

$$a^{\varphi(n)} \equiv \quad \pmod{n}$$

où φ est la fonction indicatrice d'Euler.

Supposons que m est premier avec n, alors selon le théorème d'Euler

$$m^{ed} = m^{1+h\varphi(n)} = m\left(m^{\varphi(n)}\right)^h \equiv m(1)^h \equiv m \pmod{n}$$

Par exemple, on prend deux nombres premiers $p = 13$ et $p = 7$. Leur produit donne $max = 91$. Prenons notre clé de chiffrement publique $e = 5$ et en appliquant l'algorithme d'Euclide étendu, on obtient que la clé privée est $d = 29$. Le triple $(n : 91, e : 5, d : 29)$ définit un système RSA complet.

1.2 Pourquoi l'ECC ?

La complexité de la cryptographie à clé publique RSA augmente selon la taille des clés. Comme les algorithmes pour la factorisation d'entiers sont devenus de plus en plus efficace, les méthodes basées sur RSA ont dû recourir à des clés plus longues. Cryptographie à courbe elliptique peut fournir le même niveau et le type de sécurité RSA (ou Diffie-Hellman), mais avec des clés beaucoup plus courtes.

Le tableau 1.1 compare les tailles de clé pour trois approches différentes de cryptage pour des niveaux comparables de sécurité contre les attaques par brute-force. Ce qui rend ce tableau d'autant plus significatif est que, pour des longueurs de clé comparables les charges de calcul de RSA et ECC sont comparables. Qu'est-ce que cela implique, c'est que, avec ECC, il faut un sixième l'effort de calcul pour fournir le même niveau de sécurité cryptographique que vous obtenez avec RSA 1024 bits.

Cryptographie symétrique clé (bits)	RSA et Diffie-Hellman symétrique clé (bits)	ECC clé (bits)
80	1024	60
112	2048	224
128	3072	256
192	7680	384
256	15360	512

TABLE 1.1 – Une comparaison des tailles de clé nécessaire pour atteindre le niveau de sécurité équivalent à trois méthodes différentes.

Tout cela signifie que RSA n'est pas le système idéal pour l'avenir de la cryptographie. Dans une paire d'algorithme idéal, l'algorithme facile et l'algorithme inverse (dur) deviennent plus difficiles à la même vitesse par rapport à la taille des clés. Nous avons donc besoin d'un système de clé publique fondée sur une meilleure paire d'algorithme.

Grace à des tailles de clé plus petites impliquées, l'algorithme ECC peut être mises en œuvre sur les cartes à puce sans coprocesseurs mathématiques. Cartes à puce sans

contact fonctionnent uniquement avec ECC parce que les autres systèmes nécessitent trop d'énergie d'induction. ECC est également de plus en plus important pour les communications sans fil car des clés plus courtes facilite le protocole *handshake*. ECC est également utilisé dans les algorithmes de gestion des droits numériques (DRM).

Chapitre 2

Cryptographie à Courbe Elliptique

Après l'introduction du RSA et Diffie-Hellman, les chercheurs ont exploré des solutions cryptographiques à base de mathématiques additionnelles à la recherche d'autres algorithmes au-delà de la factorisation qui serviraient comme une paire d'algorithme de meilleure complexité. En 1985, les algorithmes cryptographiques ont été proposées sur la base d'une branche ésotérique des mathématiques appelée « courbes elliptiques. »

Mais quelle est exactement une courbe elliptique et comment la paire d'algorithme marche ? Malheureusement, contrairement à factorisation, quelque chose que nous avons tous fait l'école primaire, la plupart des gens ne sont pas familiers avec les mathématiques autour des courbes elliptiques. Le calcul n'est pas aussi simple, ni l'explication.

2.1 L'idée principale d'ECC

Definition 2.1 *Groupe*

Un groupe est un ensemble G avec un opérateur binaire $\bullet$ appelle la loi de composition qui satisfait les conditions suivantes

- Loi de composition interne : $\forall\ a, b \in G\ ,\ c = a \bullet b \in G$
- Associativité : $(a \bullet b) \bullet c = a \bullet (b \bullet c) \quad \forall a, b, c \in G$
- Élément neutre : $\exists\ e$ tel que $\forall a \in G,\ e \bullet a = a \bullet e = a$
- Symétrique : $\forall a \in G,\ \exists\ b \in G \mid a \bullet b = b \bullet a = e$

Definition 2.2 *Anneau*

Un anneau est l'ensemble $(A, +, \cdot)$ qui satisfait les axiomes dits d'anneau suivants

- A est un groupe abélien
  - Loi associative $+$, $(a + b) + c = a + (b + c) \quad \forall a, b, c \in A$
  - Loi commutative $a + b = b + a$, $\forall a, b \in A$
  - Élément neutre noté $0_A \in A$ tel que $\forall a \in A$, alors $a + 0_A = a$

- Élément opposé note $-a \in A$, $\forall a \in A$ tel que $a + (-a) = 0_A$

- A est un monoïde
  - Loi associative $\cdot$, $(a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \forall a, b, c \in A$
  - Élément identité noté $1_A \in A$ tel que $\forall a \in A$, alors $a \cdot 1_A = 1_A \cdot a = a$
  - Loi disruptive à gauche, $a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad \forall a, b, c \in A$
  - Loi disruptive à droite, $(b + c) \cdot a = (b \cdot a) + (c \cdot a) \quad \forall a, b, c \in A$

Definition 2.3 *Caractéristique d'un anneau*

La caractéristique d'un anneau A est donc le plus petit entier naturel non nul n tel que

$$n \times 1_A \;=\; \underbrace{1_A + 1_A + \cdots + 1_A}_{ntermes} \;=\; 0_A$$

La caractéristique intégrale d'un anneau A est le plus petit entier naturel non nul n tel que $\forall a \in A$

$$n \times a \;=\; \underbrace{a + a + \cdots + a}_{ntermes} \;=\; 0_A$$

Supposons qu'on a un ensemble des points $E = (x_i, y_i)$ dans un plan donné. L'ensemble est finie mais vaste. On définit un opérateur (une loi de composition) $+$ (pas forcement l'opérateur addition) sur l'ensemble E tel que

$$\forall P, Q \in E, \quad \exists R \in E \quad | \quad P + Q = R$$

Soient $k \in \mathbb{N}$ et $P \in E$, on note $k \times P \in E$ (pas forcement l'opérateur multiplication) tel que

$$k \times P \quad = \quad \underbrace{P + P + \cdots + P}_{k \text{ fois}}$$

On choisit l'opérateur $+$ tel que après avoir calculé $k \times P$, c'est difficile de retrouver k à partir du résultat. Le problème de trouver k est appelle le problème du logarithme discret. Si cette condition est satisfait, alors l'opérateur choisit peut être utilisé dans le chiffrement comme Diffie-Hellman, RSA.

Toutes les conditions et hypothèses que nous avons formulées ci-dessus sont satisfaites lorsque l'ensemble E de points (x_i, y_i) est tiré d'une courbe elliptique.

2.2 Qu'est-ce qu'une courbe elliptique ?

Les courbes elliptiques sont appelés elliptique en raison de leur relation à des intégrales elliptiques en mathématiques. Une intégrale elliptique peut être utilisée pour déterminer la longueur de l'arc d'une ellipse. Une courbe elliptique est cubique et dans sa forme la plus simple est décrit par

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

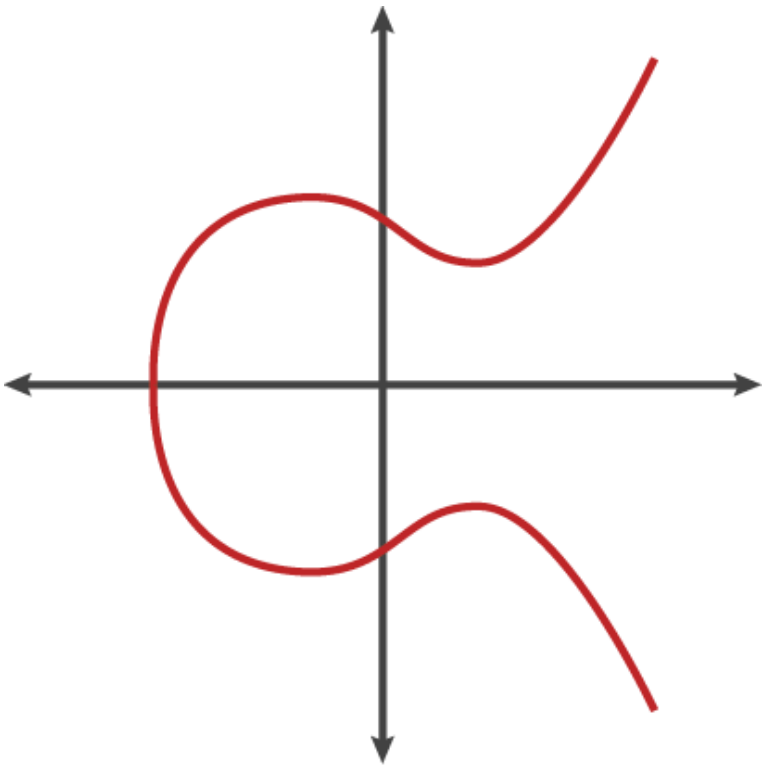

FIGURE 2.1 – Exemple d'une courbe elliptique

Les paramètres associés à l'équation ci-dessus (x, y, a et b) sont subits à des multiplications et additions. Ces valeurs doivent être tirées d'un anneau avec un élément identité de multiplication.

2.3 Operateur du groupe

Supposons que $f(x)$ a trois racines notées x_1, x_2 et x_3, on peut montrer facilement que le discriminant de $f(x)$ est

$$\Delta = -16(4a^3 + 27b^2)$$

On a $y = \pm\sqrt{x^3 + ax + b}$, cela signifie que une courbe elliptique est symétrique par rapport l'axe x.

Pour avoir trois racines différents, il faut que $4a^3 + 27b^2 \neq 0$, dans le cas contraires on appelle les courbes singuliers.. Les courbes singulières sont dangereuses dans le chiffrement.

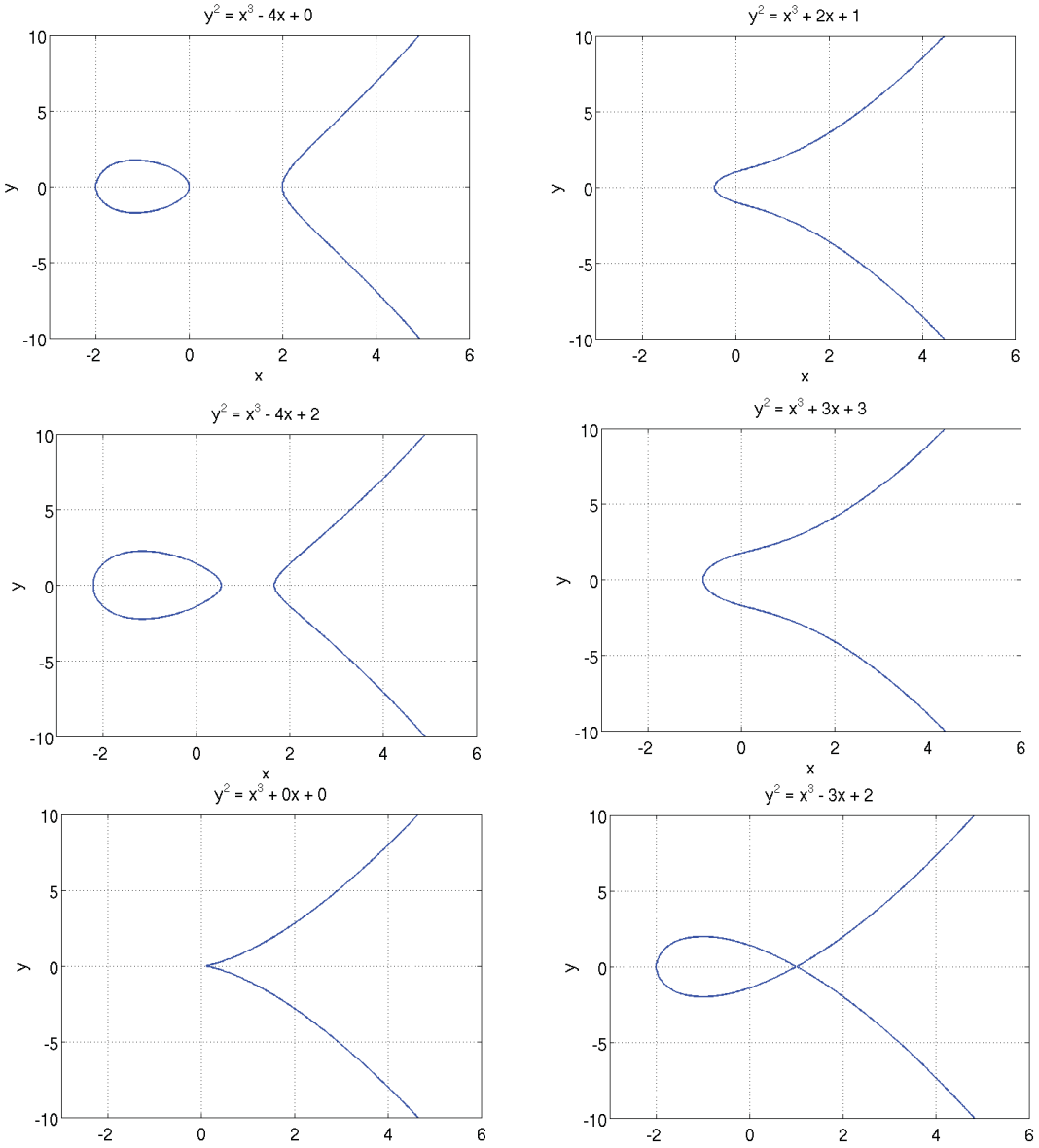

FIGURE 2.2 – Les courbes elliptiques pour des valeurs différentes de a et b

Les deux derniers exemples de la figure 2.2 montrent sont des courbes elliptiques pour lequel la condition sur le discriminant est violée. La courbe à gauche qui correspond à $f(x) = x^3$, les trois racines du polynôme cubique ont fusionné en un seul point et nous obtenons un point de rebroussement à ce point. Pour a courbe à droite qui correspond à $f(x) = x^3 - 3x^2$, deux des racines ont fusionné dans le point où la courbe se croise.

Ces deux courbes sont singulières. Comme mentionné précédemment, il est dangereux d'utiliser des courbes singulières pour la cryptographie.

Les points d'une courbe elliptique consistent un groupe. L'opérateur binaire (la loi de composition) d'une courbe elliptique est par convention appelle l'addition mais ça n'a rien avoir avec l'addition arithmétique. Pour ajouter un point P d'une courbe elliptique à une autre point Q de la même courbe, on suit la règle suivant :

— On joigne le premier point P avec Q avec une ligne droite. Un troisième point qui est l'intersection de cette droite avec la courbe, si une telle intersection existe, est notée R. L'image réflexion de ce point par rapport à l'axe x est le point $P+Q$. S'il n'existe pas un troisième point d'intersection, on dit qu'il est à l'**i**nfini.

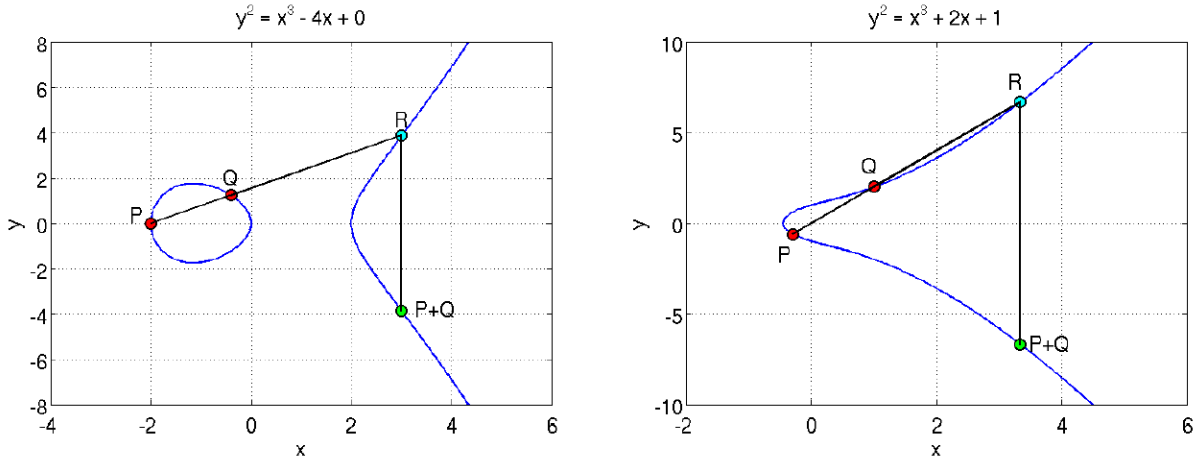

FIGURE 2.3 – La loi de composition d'une courbe elliptique.

Mais ce qui arrive quand l'intersection de P et Q est à l'infini ?

On note le point à l'infini par le symbole $\mathcal{O}$ et, à travers les définitions qui suivent, nous montrons que cela peut servir comme élément de l'identité additif pour l'opérateur de groupe. On montre alors que $P + \mathcal{O} = P$ pour tout point de la courbe.

On définit l'inverse additif d'un point P comme sa réflexion par rapport à l'axe x. Donc, si Q sur la courbe est la réflexion de P sur la courbe, alors $Q = -P$. Pour ces deux points, il serait que le troisième point de l'intersection avec la courbe d'une ligne passant par les deux points seront à l'infini. C'est à dire que le point d'intersection d'un point et son inverse additif sera le point distingué $\mathcal{O}$.

Aussi on définit que $\mathcal{O} + \mathcal{O} = \mathcal{O}$, ce qui implique que$-\mathcal{O} = \mathcal{O}$ et par conséquent, la réflexion du point à l'infini est le même point à l'infini.

À ce moment, en prenant $\mathcal{O}$ comme l'inverse additif, on peut dire que si l'intersection de P et Q est à l'infini, alors $P + Q = \mathcal{O}$ et que $Q = -P$, ce qui implique que $P + \mathcal{O} = P$.

Pour un point qui n'a pas de réflexion, c'est à dire un point qui a un tangent parallèle a l'axe y, on prendre sa réflexion comme le point lui-même $P = -P$ et on montre que pour tel point $P + P = \mathcal{O}$.

Mais si la réflexion de P existe, comment définit $P + P$. Prenons deux points P et Q et imaginons que le point Q approche P. La ligne qui connecte P et Q devient un tangent au point P si Q arrive a P. Alors pour définir $P + P$ on trace le tangent au point P et cherche la réflexion de son intersection avec la courbe pour trouver $P + P = 2P$.

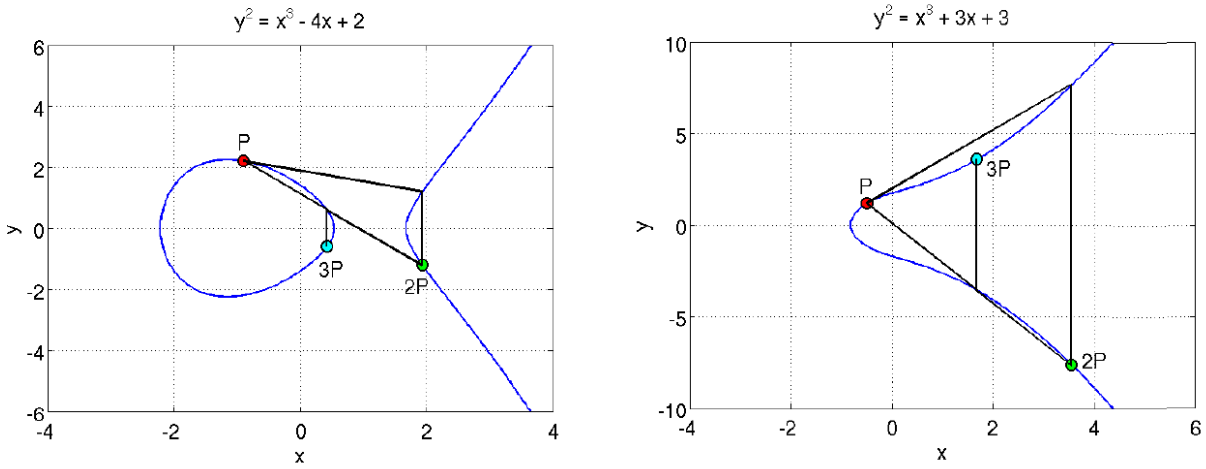

FIGURE 2.4 – Comment définir $P + P$?

Pour une courbe elliptique,

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

on note l'ensemble de points sur la courbe avec le point $\mathcal{O}$ par l'ensemble $E(a, b)$. $E(a, b)$ est un groupe d'une loi d'addition $+$ comme définit au-dessus. On a montré l'associativité, l'élément d'identité additif et la commutativité de la loi. On peut conclure que E est un groupe abélien.

On définit la "multiplication" comme l'addition répétitive

$$k \times P \quad = \quad \underbrace{P + P + \cdots + P}_{k \text{ fois}}$$

2.4 Une expression algébrique pour $P + Q$

D'après la section 2.3, l'intersection de la ligne qui joindre P et Q et la courbe et le point distinct R et $P + Q$ est la réflexion du point R, ce qui est $-R$. Alors, on a

$$P + Q = -R$$

ce qui implique que

$$P + Q + R = \mathcal{O}$$

Soient $P(x_P, y_P)$ et $Q(x_Q, y_Q)$. La ligne droite joignant P et Q est

$$y = \alpha x + \beta$$

ou $\alpha = \dfrac{y_Q - y_P}{x_Q - x_P}$ et $\beta = -\alpha x_P + y_P$.

Pour que $R(x_R, y_R)$ existe alors l'équation suivante doit être satisfaite

$$(\alpha x + \beta)^2 = x^3 + ax + b$$

et elle doit avoir trois racines distinctes x_P, x_Q et x_R. Puis que le coefficient de x^3 est égale à 1 alors on a $x_P + x_Q + x_R = \alpha^2$ ce qui implique que

$$x_R = \alpha^2 - x_P - x_Q$$

et que

$$y_R = \alpha x_R + \beta = \alpha(x_R - x_P) + y_P$$

Alors on peut définir la coordonnée de $P + Q$ qui est aussi la réflexion de point R autour de l'axe x comme suite ;

$$\begin{aligned} x_{P+Q} &= \alpha^2 - x_P - x_Q \\ y_{P+Q} &= \alpha(x_P - x_R) - y_P \end{aligned}$$

2.5 Une expression algébrique pour $P + P = 2P$

Soit le point $P \in E(a, b)$, pour trouver $2P$, on trace le tangent au point P et on cherche l'intersection de ce tangent avec la courbe. La réflexion de cette intersection nous donne $2P$

Le pont de ce tangent est

$$2y\frac{dy}{dx} = 3x^2 + a \implies \alpha = \frac{dy}{dx} = \frac{3x_P^2 + a}{2y_P}$$

et de même façon que l'analyse précédente on trouve que

$$\begin{aligned} x_{2P} &= \alpha^2 - 2x_P \\ y_{2P} &= \alpha(x_P - x_R) - y_P \end{aligned}$$

2.6 La courbe elliptique sur $\mathbb{Z}_p$

L'arithmétique de la courbe elliptique qu'on a décrite jusqu'ici était sur les nombres réels. Ces courbes ne peuvent pas être utilisés tels quels pour la cryptographie parce que les calculs avec des nombres réels sont sujettes à arrondir erreur. Cryptographie nécessite

une arithmétique sans erreur. En limitant les valeurs des paramètres a et b, la valeur de la variable indépendante x, et la valeur de la variable dépendante y à un certain corps fini $\mathbb{Z}_p$ d'un nombre premier p, on obtient les courbes elliptiques qui sont plus appropriés pour la cryptographie. Ces courbes seraient décrites par

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

On note $E_p(a, b) \subset E(a, b)$ l'ensemble des points qui satisfait l'équation décrit au-dessus.

Comme les points de $E_p(a, b)$ ne peuvent plus être reliés pour former une courbe lisse, on ne peut pas utiliser la construction géométrique pour illustrer l'action de l'opérateur de groupe. C'est à dire, étant donné un point P, maintenant on ne peut pas montrer géométriquement comment calculer $2P$, ou donné deux points P et Q, on ne peut pas montrer géométriquement comment déterminer $P+Q$. Cependant, les expressions algébriques que nous avons tirés de ces opérations continuent de tenir bon à condition que les calculs sont effectués modulo p.

2.7 Une implémentation en Java

Avec l'API BigInteger qui a toute les fonctions d'arithmétique modulaire, on peut facilement implémenter une courbe elliptique.

2.7.1 Point

```
import java.math.BigInteger;
import java.util.Objects;

public class Point {

    private final BigInteger x;
    private final BigInteger y;

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 59 * hash + Objects.hashCode(this.x);
        hash = 59 * hash + Objects.hashCode(this.y);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
```

```
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Point other = (Point) obj;
        if (!Objects.equals(this.x, other.x)) {
            return false;
        }
        return Objects.equals(this.y, other.y);
    }

    public Point(String x, String y) {
        this.x = new BigInteger(x);
        this.y = new BigInteger(y);
    }

    public Point(BigInteger x, BigInteger y) {
        this(x.toString(), y.toString());
    }

    public BigInteger getX() {
        return x;
    }

    public BigInteger getY() {
        return y;
    }

    Point getXReflection() {
        return new Point(x, y.negate());
    }

}
```

2.7.2 EllipticCurve

```
import java.math.BigInteger;

public class EllipticCurve {

    public static final BigInteger ZERO = new BigInteger("0");
    public static final BigInteger ONE = new BigInteger("1");
    public static final BigInteger TWO = new BigInteger("2");
```

```
public static final BigInteger THREE = new BigInteger("3");
public static final Point IDENTITY = new Point(ZERO, ZERO);
private final BigInteger prime;
private final BigInteger a;
private final BigInteger b;

public EllipticCurve(String prime, String a, String b) {
    this.prime = new BigInteger(prime);
    this.a = new BigInteger(a).mod(this.prime);
    this.b = new BigInteger(b).mod(this.prime);
}

public Point add(Point p1, Point p2) {
    if (p1 == IDENTITY) {
        return p2;
    }
    if (p2 == IDENTITY) {
        return p1;
    }
    if (p1.equals(p2.getXReflection())) {
        return IDENTITY;
    }

    BigInteger alphaN, alphaD, alpha;

    if (p1 == p2 || p1.equals(p2)) {
        alphaN = THREE.multiply(p1.getX().modPow(TWO, prime))
                .add(a).mod(prime);
        alphaD = TWO.multiply(p1.getY()).mod(prime);
    } else {
        alphaD = p1.getX().subtract(p2.getX());
        alphaN = p1.getY().subtract(p2.getY());
    }

    alpha = alphaN.multiply(alphaD.modInverse(prime))
            .mod(prime);
    BigInteger rX = alpha
            .modPow(TWO, prime)
            .subtract(p1.getX()).mod(prime)
            .subtract(p2.getX()).mod(prime);
    BigInteger rY = alpha
            .multiply(p1.getX().subtract(rX)).mod(prime)
            .subtract(p1.getY()).mod(prime);
```

```
        return new Point(rX, rY);
    }

    public Point pow(Point p, String k) throws Exception {
        return pow(p, new BigInteger(k));
    }

    private Point pow(Point p, BigInteger k) throws Exception {
        if (k.compareTo(ZERO) <= 0) {
            throw new Exception("Illegal value for k");
        }
        if (p == IDENTITY) {
            return IDENTITY;
        }
        if (k.equals(ONE)) {
            return p;
        }
        if (k.equals(TWO)) {
            return add(p, p);
        }
        if (k.mod(TWO).equals(ZERO)) {
            return pow(pow(p, k.divide(TWO)), TWO);
        } else {
            return add(pow(pow(p, k.divide(TWO)), TWO), p);
        }
    }

    private String imageOf(String x) {
        return imageOf(new BigInteger(x)).toString();
    }

    public BigInteger imageOf(BigInteger x) {
        return x.modPow(THREE, prime)
                .add(a.multiply(x)).mod(prime)
                .add(b).mod(prime);
    }

    boolean isOnCurve(Point p) {
        return p.getY().modPow(TWO, prime)
                .equals(imageOf(p.getX()));
    }
}
```

2.7.3 Un exemple facile $a = 1$, $b = 4$, $p = 23$

```
import java.util.HashSet;

public class Main {

    public static void main(String[] args) {
        HashSet<Point> set = new HashSet<>();
        EllipticCurve ecc = new EllipticCurve("23", "1", "4");
        for (Integer i = 0; i < 23; i++) {
            for (Integer j = 0; j < 23; j++) {
                Point p = new Point(i.toString(), j.toString());
                if (ecc.isOnCurve(p)) {
                    set.add(p);
                }
            }
        }
        for (Point p : set) {
            System.out.println(p.getX().toString() + ", " + p.getY());
        }
    }
}
```

Pour une courbe elliptique avec $a = 1$, $b = 4$ et $p = 23$ on trouve les 29 points suivants. Une telle courbe est cyclique, chaque point est génératrice de tous les autres points.

```
[ IDENTITY, [8, 15], [15, 6], [7, 3], [14, 5], [0, 21], [22, 5],
[17, 9], [10, 18], [18, 14], [22, 18], [11, 9], [13, 11],
[17, 14], [8, 8], [13, 12], [9, 11], [15, 17], [9, 12], [14, 18],
[10, 5], [11, 14], [4, 7], [7, 20], [18, 9], [4, 16], [1, 11],
[0, 2], [1, 12] ]
```

La courbe utilisée par Google a pour valeurs

$p = 115792089210356248762697446949407573530086143415290314195533631308867097853951$

$a = 115792089210356248762697446949407573530086143415290314195533631308867097853948$

$b = 41058363725152142129326129780047268409114441015993725554835256314039467401291$

2.8 ECC - Échange de clés Diffie-Hellman

Une communauté d'utilisateurs qui souhaitent s'engager dans des communications sécurisées avec ECC choisit les paramètres q, a et b pour un groupe basé à courbe elliptique $E_q(a, b)$, et un point de base $G \in E_q(a, b)$.

A sélectionne un nombre entier X_A pour servir de sa clé privée. A génère alors $Y_A = X_A \times G$ pour servir de sa clé publique. A met à la disposition du public sa clé publique YA.

B désigne un entier X_B pour servir de sa clé privée. Comme cela a été fait par A, B calcule également sa clé publique par $Y_B = X_B \times G$.

Afin de créer une clé secrète partagée (qui pourrait par la suite être utilisé pour, par exemple, un lien de communication basée à clé symétrique), A et B maintenant effectuer les opérations suivantes :

— A calcule la clé de session partagée par $K_A = X_A \times Y_B$

— B calcule la clé de session partagée par $K_B = X_B \times Y_A$

On a

$$\begin{aligned} K_A = &= X_A \times Y_B \\ &= X_A \times (X_B \times G) \\ &= (X_A \times X_B) \times G \\ &= (X_B \times X_A) \times G \\ &= X_B \times (X_A \times G) \\ &= X_B \times Y_A \\ &= K_B \end{aligned}$$

Alors $K = K_A = K_B$ est la clé partagée.

Pour découvrir la clé secrète de session, un attaquant pourrait tenter de découvrir X_A du point de base disponible publiquement G et la disposition du public Y_A. Rappel, $Y_A = X_A \times G$. Mais, comme déjà expliqué, ça nécessite de résoudre le problème du logarithme discret qui, pour un ensemble de paramètres de la courbe et G bien choisi, est extrêmement difficile.

Pour augmenter le niveau de difficulté à résoudre le problème du logarithme discret, nous sélectionnons pour G un point de base dont l'ordre est très grande. L'ordre d'un point sur la courbe elliptique est le plus petit nombre de fois où G doit être ajoutée à elle-même de sorte que l'on obtient un élément d'identité $\mathcal{O}$ du groupe $E_q(a, b)$. On peut également associer la notion d'ordre avec une courbe elliptique sur un corps fini : L'ordre d'une courbe elliptique est le nombre total de points dans l'ensemble $E_q(a, b)$.

Puis que les entiers X_A, Y_A, X_B, Y_B doivent tous être inférieur à l'ordre du point de base G, la valeur de l'ordre du point de base doit également être mis à la disposition du public.

Pour les cryptos de ce projet , on utilise les valeurs suivantes définit par NIST (National Institute of Standards and Technology)
http ://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf

$$\begin{aligned} p &= 11579208921035624876269744694940757353008614341529031419553363130886709785395 1 \\ a &= -3 \\ b &= 5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b_{16} \\ \text{ordre} &= 11579208921035624876269744694940757352999695522413576034242225906106851204436 9 \\ G_x &= 6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296_{16} \\ G_y &= 4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5_{16} \end{aligned}$$

On vérifie aussi les paramètres avec notre implémentation en Java.

```
public static void main(String[] args) throws Exception {
    EllipticCurve ecc = new EllipticCurve(DEF_PRIME, DEF_A, DEF_B);
    Point base = new Point(DEF_GX, DEF_GY);

    //verifier que l'ordre est correct
    System.out.println("ORDRE * PRIME =: " + DEF_ORDER
            .multiply(DEF_PRIME).mod(DEF_PRIME));

    //verifier que le point de base est bien un point de la courbe
    System.out.println("Image de G_X =: " + ecc
            .imageOf(DEF_GX).toString(16));
    System.out.println("Valeur de G_Y^2 =: " + DEF_GY
            .modPow(TWO, DEF_PRIME).toString(16));

}
```

```
ORDRE * PRIME =: 0
Image de G_X =:
55df5d5850f47bad82149139979369fe498a9022a412b5e0bedd2cfc21c3ed91
Valeur de G_Y^2 =:
55df5d5850f47bad82149139979369fe498a9022a412b5e0bedd2cfc21c3ed91
```

2.9 ECDSA - Elliptic curve digital signature algorithm

Ceci est la version ECC de l'algorithme de signature numérique. Cet algorithme, plus communément connu par son acronyme ECDSA, a été beaucoup dans les nouvelles dernièrement en raison de son utilisation pour l'authentification de code dans les consoles de jeux PlayStation3. Des moyens d'authentification de code que la signature numérique d'un fichier binaire est contrôlé et vérifié avant d'être autorisé à être exécuté sur un processeur. Parallèlement à notre description plus tôt, les différentes étapes d'ECDSA sont :

— Pour une signature numérique basée sur une courbe elliptique définie sur un corps fini premier Z_p, sélectionner un grand nombre premier p et choisissez les paramètres a et b de la courbe, et le point de base G de haute ordre n, ce qui signifie que $n \times G = \mathcal{O}$ pour un grand n. Maintenant, sélectionnez au hasard X tel que, $1 \leq X \leq n-1$, pour servir de votre clé privée.

— Ensuite, vous calculez votre clé publique par $Y = X \times G$ où l'opération "multiplication" est selon la loi de groupe de la courbe elliptique. À noter que la clé publique est constituée d'une paire de nombres qui sont les coordonnées du point Y sur la courbe elliptique.

— Vous ferez p, a, b, g, n et Y à la disposition du public et vous traiter X comme votre clé privée.

— Générer un nombre aléatoire unique K tel que $0 < K < n - 1$. Par unique, c'est à dire qu'il faut jeter K après Chaque utilisation. C'est à dire, chaque signature numérique que vous créez sera avec un autre K. Vous devez jeter K après chaque utilisation. En utilisant le même K pour deux signatures différentes est une violation majeure de la sécurité dans l'utilisation de cet algorithme, comme il sera expliqué plus tard.

— Maintenant, vous êtes prêt à construire une signature numérique d'un document. Soit M un entier qui représente un hachage du document que vous souhaitez signer.

— La signature numérique que vous construisez pour M se composera de deux parties que nous noterons S_1 et S_2. Vous construisez S_2 par calculant d'abord le point de la courbe elliptique $K \times G$ et retenant seulement sa coordonnée-x modulo n :

$$S_1 = (K \times G)_x \pmod{n}$$

Si l'opération modulo produit une valeur zéro pour S_1, vous essayez une valeur différente pour K. Vous construisez S_2 par

$$S_2 = K^{-1} \cdot (M + X \cdot S_1) \pmod{n}$$

ou K^{-1} est l'inverse multiplicatif de K modulo n qui peut être obtenu avec l'algorithme de Euclide étendue.

Supposons que vous avez envoyé votre document avec sa signature (S_1, S_2) à un certain destinataire et le destinataire souhaite s'assurer qu'il ne reçoit pas un message modifié. Le destinataire peut vérifier l'authenticité du document en ; premièrement il calcule le hachage M du document (en utilisant le même algorithme que vous avez fait) et par suite calculer le nombre $w = S_2^{-1} \pmod{n}$, $u_1 = M \cdot w \pmod{n}$, et $u_2 = S_1 \cdot w \pmod{n}$. Avec ces chiffres il calcule le point $(x, y) = u_1 \times G + u_2 \times Y$ sur la courbe et enfin, l'authentification de la signature en vérifiant si $S_1 \equiv x \pmod{n}$ est vérifié.

On va maintenant aborder le danger d'utiliser le même K pour deux documents différents - danger en ce sens est que l'adversaire peut trouver la clé privée, puis passez à la contrefaçon de votre signature. Supposons que les hachages de deux documents différents que vous signez avec la même valeur K sont M et M'. Les deux signatures pour ces deux documents vont ressembler :

$$\begin{aligned} S_1 &= (K \times G)_x \pmod{n}, \quad S_2 = K^{-1} \cdot (M - X \cdot S_1) \pmod{n} \\ S_1' &= (K \times G)_x \pmod{n}, \quad S_2' = K^{-1} \cdot (M' - X \cdot S_1') \pmod{n} \end{aligned}$$

On remarque que S_1 et S_2 restent les mêmes parce qu'ils sont indépendants du document. Par conséquent, si un adversaire devait calculer la différence $S_2 - S_1$, il obtiendrait

$$S_2 - S_1 = K^{-1}(M - M')$$

De là, l'adversaire peut immédiatement calculer la valeur de K vous avez utilisé pour votre signature numérique. Et, en utilisant l'équation $S_2 = K^{-1} \cdot (M - X \cdot S_1) \pmod{n}$, l'adversaire peut procéder pour calculer la clé privée X.

Chapitre 3

L'interface graphique

L'interface se compose de deux parties, la première est l'échange de clés Diffie-Hellman et la deuxième est la signature et vérification DSA.

3.1 ECDH - Elliptic curve Diffie–Hellman

Si on clique sur le bouton Diffie-Hellman, une vue s'affiche. Cette vue se compose en deux Alice et Bob. Dans cette vue, avec les boutons "GEN", on peut générer les clés privées pour Alice et Bob, les clés publiques sont calculer automatiquement. Une fois que les deux clés ont été bien générées, la clé secrète partagée est aussi générée pour Alice et Bob.

On remarque que cette clé est la même pour Alice et Bob ce qui vérifie bien l'algorithme qu'on a décrit.

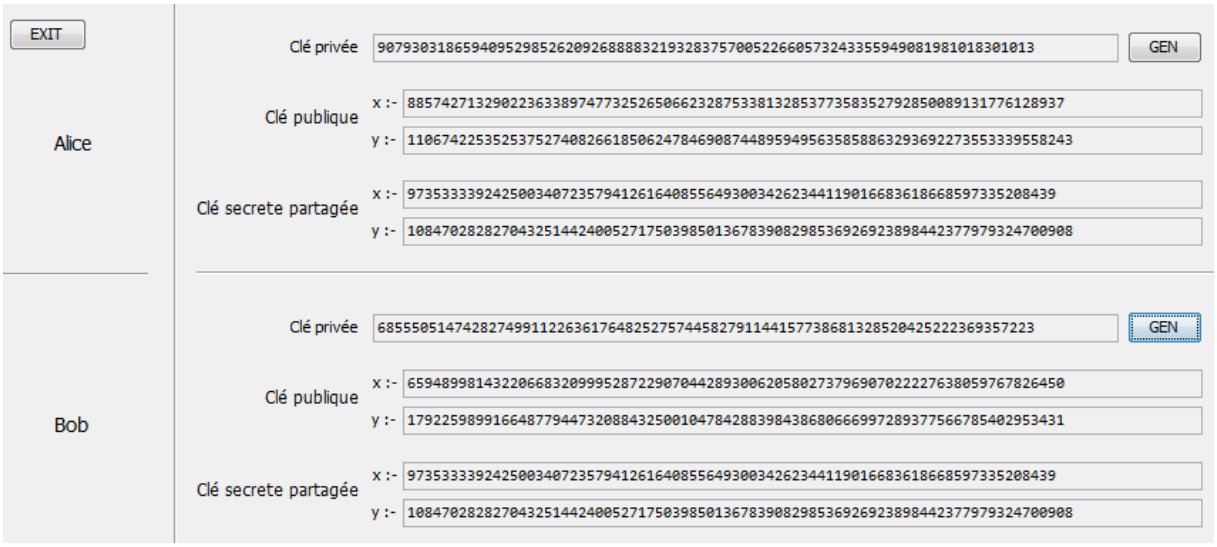

FIGURE 3.1 – Démonstration ; Échange de clés Diffie Hellman

3.2 ECDSA - Elliptic curve digital signature algorithm

Le bouton "ECC DSA" nous permet de lancer la vue d'ECDSA. Pour démontrer, on choisir un fichier et le hachage MD5 est calculé immédiatement. Assi on génère une clé privé et la clé publique est calculer automatiquement. La valeur de nombre unique K est aussi génère. Une fois que tous ces paramètres sont avalables, la signature $S(S_1, S_2)$ est calculé automatiquement est cette signature est valider en utilisant l'algorithme qu'on a décrit dans le chapitre précèdent.

On observe que la valeur de S est bien la valeur de S_1 de la signature calculer, ce qui montre bien que l'algorithme est valide.

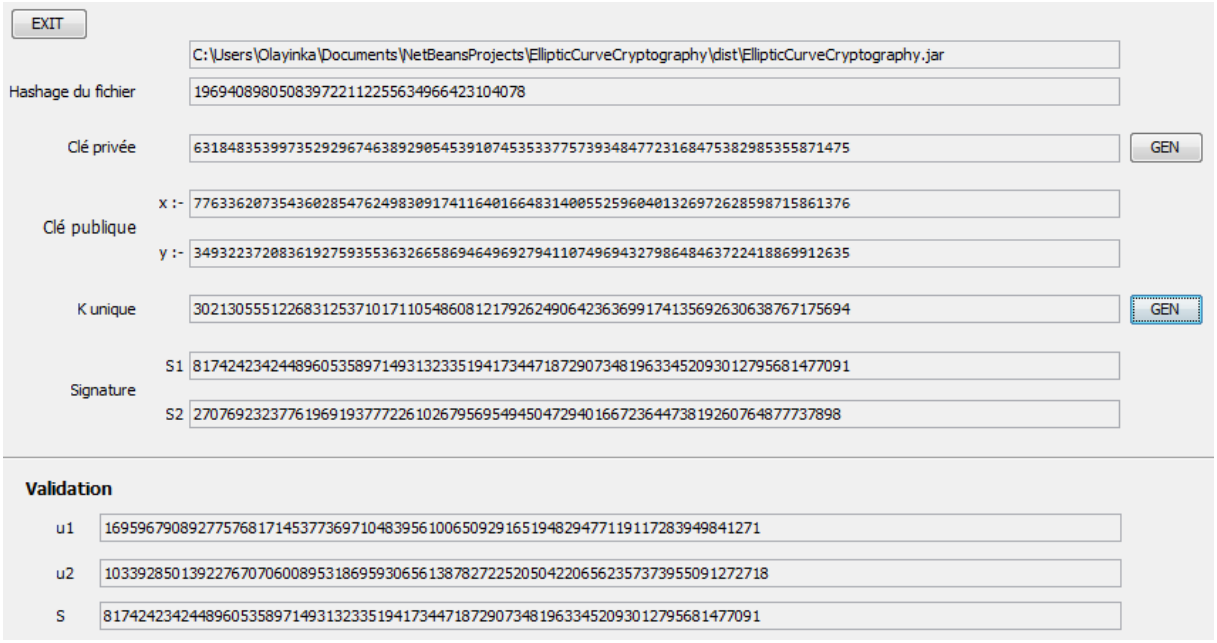

FIGURE 3.2 – Démonstration ; Signature DSA

3.3 La Sécurité d'ECC

Tout comme RSA dépend de la difficulté de factorisation d'un grand nombre pour sa sécurité, ECC dépend de la difficulté du calcul du logarithme discret d'un grand nombre. C'est ce qu'on appelle aussi le problème du logarithme discret d'une courbe elliptique (ECDLP).

Il a été montré par Menezes, Okamoto, et Vanstone (MOV) en 1993 (pour les courbes elliptiques super singuliers) le problème de la résolution du problème de ECDLP (dont le domaine est le groupe $E_q(a, b)$) peut être réduite au problème beaucoup plus facile de trouver des logarithmes dans un corps fini. Il a eu beaucoup de travail récemment sur l'extension de la réduction aux courbes elliptiques généraux.

Afin de ne pas tomber en proie à la crise MOV, la courbe elliptique sous et le point de base choisi doit satisfaire ce qui est connu comme la condition MOV.

L'état de MOV est énoncé en termes de l'ordre du point G. L'ordre m du point de base G est la valeur de m tel que $m \times G = \mathcal{O}$, où $\mathcal{O}$ est l'élément de l'identité l'additif comme définit précédemment.

La condition MOV indique que l'ordre m du point de base ne doit pas diviser $Q^B - 1$ pour les petits B, par exemple pour $B < 20$ Notez que q est le premier p lorsque le corps fini est Z_p.

Les courbes elliptiques pour lesquels le nombre total de points de la courbe est égal au nombre d'éléments dans le champ fini sont également considérés comme cryptographiquement faibles.

MERCI