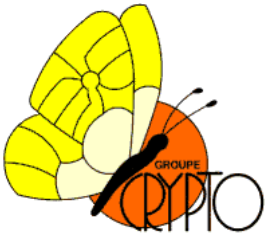
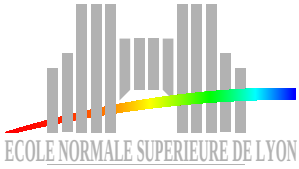


Université Catholique de Louvain
Laboratoire de micro-électronique
Groupe Crypto



Ecole Normale Supérieure de Lyon
Département de Mathématiques et d'Informatique
Magistère d'Informatique et Modélisation

Rapport de stage de seconde année de
Magistère d'Informatique et Modélisation (MIM)

Implémentation efficace du couplage de Weil en caractéristique trois et réalisation du protocole de Joux

Nicolas BERNARD
<n.bernard@lafraze.net>

Stage réalisé sous la direction de :

Jean-Jacques QUISQUATER
et
Benoît LIBERT

Septembre 2003
version recompilée en août 2010

Remerciements :

Dans tout rapport de ce type, c'est quasiment un devoir de remercier un tas de monde. Dans le cas présent c'est aussi un plaisir... Je remercie donc d'abord, bien sûr le Professeur Jean-Jacques Quisquater, qui m'a encadré durant ce stage, ainsi que Benoit Libert qui m'a guidé et donné de précieux conseils.
D'une manière plus générale, je remercie également tout le Groupe Crypto, qui m'a accueilli chaleureusement avec une pensée particulière pour mes deux autres coburéaux Gaël Rouvroy et François-Xavier Standaert, ainsi que pour Mathieu Ciet et Julien Cathalo qui ont toujours été disponibles pour répondre à mes questions, qu'elles soient mathématiques ou non.
D'une manière plus générale encore, j'aimerais remercier tout le personnel du laboratoire de micro-électronique, pour l'ambiance chaleureuse et amicale qui règne dans leurs locaux.

Table des matières

Introduction

La cryptographie est une branche à la frontière des mathématiques et de l'informatique qui prend une importance grandissante à notre époque où les transactions "immatérielles", via les réseaux informatiques, sont en passe de devenir une réalité tangible et inévitable dans la vie de l'individu lambda. En effet de telles transactions doivent offrir au moins les mêmes garanties que leurs pendants "classiques". La cryptographie permet de fournir ces garanties.

Néanmoins, la cryptographie, pour tenir son rôle, doit offrir des services fiables, rapides, et pour certains, peu gourmands en ressources de manière à pouvoir fonctionner sur des applications embarquées comme des cartes à puces. Pour atteindre ces objectifs, les cryptologues ont exploré [1] différentes méthodes ; l'une d'elles est la cryptographie basée sur les courbes elliptiques.

La cryptographie à courbes elliptiques, si elle est encore assez peu utilisée, promet beaucoup de choses : la plus connue est l'utilisation des algorithmes classiques avec des clefs plus courtes pour une sécurité identique, mais cela ne s'arrête pas là. Elle a également permis de réaliser l'un des premiers systèmes de chiffrement basés sur l'identité qui ne fasse pas d'hypothèses contraignantes sur ses conditions de mise en oeuvre ([6]), système maintenant commercialisé par une jeune société, **Voltage Security**, fondée par l'un des deux inventeurs de la méthode. Une des autres possibilités est la réalisation de signatures numériques courtes ([7]). On peut également réaliser un équivalent du protocole d'échange de clefs de Diffie-Hellman entre trois personnes en une seule passe.

Ces trois dernières possibilités reposent sur l'utilisation de *couplages*. Dans le présent rapport, nous expliquons comment réaliser une implémentation efficace du couplage de Weil dans le cas particulier des corps de caractéristique trois, en utilisant des résultats de recherche récents pour obtenir une efficacité maximale. Comme exemple d'utilisation d'une telle implémentation, nous décrivons ensuite comment réaliser l'échange de clefs auquel nous faisions allusion précédemment : le protocole de Joux.

[1] et explorent toujours !

Chapitre 1

Mathématiques, cryptologie et courbes elliptiques

Après avoir rappelé les définitions de base de l'arithmétique sur les corps finis, nous présenterons rapidement les problèmes qui fournissent la difficulté nécessaire aux outils cryptographiques que nous avons implémentés, puis les courbes elliptiques en indiquant les différents algorithmes élémentaires qui s'y rapportent, dans le cas de la caractéristique 3.

1.1 Corps finis

Un *corps fini* F est un corps qui contient un nombre fini d'éléments. L'*ordre* ou *cardinal* de F est le nombre d'éléments de F.

Si F est un corps fini, F contient p^m éléments, où p est premier et $m \geq 1$. Pour chacune de ces puissances p^m, il y a un unique corps (à un isomorphisme près) d'ordre p^m. Ce corps est noté $\mathbb{F}_{p^m}$[1]. p est la *caractéristique* de $\mathbb{F}_{p^m}$.

1.2 Les problèmes de Diffie-Hellman

1.2.1 Le problème de Diffie-Hellman Calculatoire (CDH)

Il existe différents « problèmes de Diffie-Hellman ». Le plus connu, « l'original » si l'on peut dire, est le *problème de Diffie-Hellman Calculatoire (CDH)* : si G est un groupe cyclique, g un générateur de G, a, b deux entiers aléatoires dans l'emsemble $\{1, \ldots, |G|\}$, le problème consiste à calculer g^{ab}

[1] On voit parfois la notation $GF(p^m)$ (pour *Galois Field* en anglais et non pas Groupe Fini !), en particulier lorsque peu de possibilités de mise en forme sont possibles

sans connaître ni a ni b mais uniquement g^a et g^b. S'il n'y a pas d'algorithme efficace pour ce faire dans G, on dit que G *vérifie l'hypothèse de Diffie-Hellman Calculatoire*. On remarque que résoudre le problème du logarithme discret dans G permet de résoudre CDH [2]. C'est sur celui-ci qu'est basé le protocole d'échange de clefs de Diffie-Hellman « classique ». Ce problème est réputé difficile.

1.2.2 Le problème de Décision de Diffie-Hellman (DDH)

Le problème de Décision de Diffie-Hellman peut, toujours informellement et avec les même notations que ci-dessus, s'énoncer ainsi : « a, b, c étant choisis aléatoirement dans $[1, |G|]$, comment distinguer les distributions $\langle g^a, g^b, g^{ab}\rangle$ et $\langle g^a, g^b, g^c\rangle$? ». De la même manière que précédemment, s'il n'existe pas d'algorithme efficace pour faire cela dans un groupe G, on dira qu'il satisfait l'*hypothèse de Décision de Diffie-Hellman*.

On note que cette hypothèse est plus forte que la précédente : il existe d'ailleurs des exemples où l'hypothèse CDH est toujours acceptée alors que l'hypothèse DDH ne tient pas, l'exemple le plus simple étant celui des groupes $\mathbb{Z}_p^*$ où l'on peut calculer à partir de g^a et g^b le symbole de Legendre de g^{ab} et ainsi le distinguer de g^c. (Voir [5] pour plus de détails sur DDH.)

1.2.3 Le problème de Diffie-Hellman Bilinéaire (BDH)

Soient G_1, G_2 deux groupes d'ordre premier q. Soit $e : G_1 \times G_1 \rightarrow G2$ une application bilinéaire [3] et P un générateur de G_1. Le problème de Diffie-Hellman Bilinéaire (BDH) dans $\langle G_1, G_2, e\rangle$ est : « étant donnés $\langle P, aP, bP, cP\rangle$, pour a, b, c quelconques dans $\mathbb{Z}_q^*$, calculer $W = e(P, P)^{abc}$ ».

C'est ce problème qui fournit la difficulté sous-jacente nécessaire à certains cryptosystèmes basés sur le couplage de Weil. Comme nous le verrons ultérieurement, le protocole de Joux en est une application directe.

Notons cependant que la difficulté exacte du problème est à l'heure actuelle encore mal connue. On sait qu'il n'est pas plus difficile que CDH, par contre on ignore s'il est strictement plus facile.

[2] Notons que l'on peut réduire le problème du logarithme discret sur les courbes elliptiques au problème du logarithme discret sur les corps finis, grâce au couplage de Weil. Cela sort néanmoins du cadre du présent rapport et le lecteur intéressé est invité à se référer au chapitre 5 de [19].

[3] e est bilinéaire si et seulement si $\forall P, Q \in G_1,\ \forall a, b \in \mathbb{Z},\ e(aP, bQ) = e(P, Q)^{ab}$.

Le couplage de Weil que nous utilisons par la suite comme application bilinéaire est supposé fournir un cas où G_2 satisfait l'hypothèse DDH.

1.3 Les courbes elliptiques et leurs utilisations en cryptologie

1.3.1 Brève présentation des courbes elliptiques

Nous indiquons ici quelques notions sur les courbes elliptiques, en nous intéressant plus particulièrement au cas des courbes de caractéristique 3. Pour de plus amples détails (mais dans le cas général) le lecteur se reportera à [17] et [19].

Définition

Une courbe elliptique est l'ensemble des solutions (x, y) d'une équation de la forme

$$E \ : \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a^6,$$

dite *équation de Weierstrass*, auquel on ajoute un point $\mathcal{O}$ dit *point à l'infini*.

Le *discriminant* Δ d'une telle courbe est défini comme :

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6$$

où $b_2 = a_1^2 + 4a_2,\ b_4 = 2a_4 + a_1a_3,\ b_6 = a_3^3 + 4a_6$ et
$b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$.

Une courbe elliptique est définie sur un corps $\mathbb{F}_q$, si les a_i sont dans $\mathbb{F}_q$ et si le discriminant est non nul.

Courbe elliptique sur un corps de caractéristique 3 Sur un corps de caractéristique différente de deux, donc plus particulièrement, en caractéristique 3, toute courbe elliptique peut être décrite par une équation de la forme

$$E' \ : \ y^2 = x^3 + b_2x^2 + b_4x + b_6$$

On obtient ce résultat en faisant dans l'équation de Weierstrass le changement de variables

$$(x, y) \longrightarrow \left(x, y - \frac{a_1}{2}x - \frac{a_3}{2}\right).$$

Les b_i sont donnés en fonction des a_i par les formules suivantes :

$$\left\{\begin{array}{rcl} b_2 & = & a_2 + \frac{a_1^2}{4} \\ b_4 & = & a_4 + \frac{a_1 a_3}{4} \\ b_6 & = & a_6 + \frac{a_3^2}{4} \end{array}\right.$$

Loi de groupe

Il existe une loi de groupe abélien sur E :

D'une manière plus formelle, on définit la loi suivante, que l'on note traditionnellement de manière additive : $\forall P, Q \in E$:

- $\mathcal{O} + P = P$ et $P + \mathcal{O} = P$;
- $-\mathcal{O} = \mathcal{O}$;
- si $P = (x_1, y_1) \neq \mathcal{O}$, alors $-P = (x_1, -y_1 - a_1 x_1 - a_3)$;
- si $Q = -P$, alors $P + Q = \mathcal{O}$.
- si $P \neq \mathcal{O}$, $Q \neq \mathcal{O}$, $Q \neq -P$, alors, si R est le troisième point d'intersection avec la courbe — soit de la ligne $\overline{PQ}$ si $P \neq Q$, soit de la tangente à la courbe en P si $P = Q$ — $P + Q = -R$.

Formules explicites Donnons maintenant explicitement quelques formules en caractéristique 3, dont on peut déduire les autres :

Inverse d'un point L'inverse d'un point P de coordonnées (x_P, y_P) est un point -P de coordonnées $(x_P, -y_P - a_1 p_1 - a_3)$ soit en caractéristique 3 :$(x_P, -y_P)$.

Addition En caractéristique 3, si on note $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ et $R = P + Q = (x_R, y_R)$, la loi d'addition est :

- Si $P = -Q$, $R = \mathcal{O}$.
- Si $P = Q$, $\lambda \equiv 1/y_P$, $\left\{\begin{array}{rcl} x_R & = & x_P + \lambda^2 \\ y_R & = & -(y_P + \lambda^3) \end{array}\right.$.
- Sinon, $\lambda \equiv \frac{y_Q - y_P}{x_Q - x_P}$, $\left\{\begin{array}{rcl} x_R & = & \lambda^2 - (x_P + x_Q) \\ y_R & = & y_P + y_Q - \lambda^3 \end{array}\right.$.

Points de n-torsion

$E = E(\overline{\mathbb{F}_q})$ est un *groupe de torsion* si pour chaque point $P \in E$ il y a un entier positif n tel que $nP = \mathcal{O}$. Le plus petit entier satisfaisant cette condition est appelé l'*ordre* de P. On note $E[n]$ le sous-groupe des points de n-torsion dans $E(\overline{\mathbb{F}_q})$, $n \neq 0$.

Diviseurs

Soit $\mathbb{K} = \mathbb{F}_q$ et soit E / $\mathbb{K}$ une courbe elliptique. $\overline{\mathbb{K}}$ étant la clôture algébrique de $\mathbb{K}$, un *diviseur* D est une somme formelle de $\overline{\mathbb{K}}$-points

$$D = \sum_{P \in E} n_P(P)$$

où les $n_P \in \mathbb{Z}$ et sont tous nuls sauf en un nombre fini de points $P \in E$.

Le *degré d'un diviseur* $D = \sum n_P(P)$ est l'entier $\sum n_p$. On appelle D^0 l'ensemble des diviseurs de degré nul. Cet ensemble est un groupe.

Le *corps des fonctions* $\overline{\mathbb{K}}(E)$ est le corps des fractions de $\overline{\mathbb{K}}[E]$. Ses éléments sont appelées *fonctions rationnelles*.

Un diviseur $D \in D^0$ est *principal* s'il existe $f \in \overline{\mathbb{K}}(E)^*$ telle que $D = \mathrm{div}(f)$.

On a le théorème suivant : *Soit $D = \sum n_p(P)$ un diviseur. Alors D est principal si et seulement si $\sum n_p = 0$ et $\sum n_P P = \mathcal{O}$.*

Deux diviseurs $D_1, D_2 \in D^0$ sont dit *équivalents* (ce que l'on note $D_1 \sim D_2$) si et seulement s'il existe $f \in \overline{\mathbb{K}}(E)$ telle que $D_1 = D_2 + \mathrm{div}(f)$.

Divers

Equation de la tangente en un point Soit donc une courbe définie par son équation de Weierstrass :

$$E \ : \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Dérivons implicitement par rapport à x :

$$\begin{aligned} 2y\frac{dy}{dx} + a_1\frac{dy}{dx} + a_3\frac{dy}{dx} &= 3x^2 + 2a_2x + a_4 && (1.1)\\ (2y + a_1 + a_3)\frac{dy}{dx} &= 3x^2 + 2a_2x + a_4 && (1.2)\\ \frac{dy}{dx} &= \frac{\overbrace{3x^2 + 2a_2x + a_4}^{A(x)}}{\underbrace{2y + a_1 + a_3}_{B(y)}} && (1.3) \end{aligned}$$

L'équation de la tangente peut alors s'écrire :

$$B(y_0)(y - y_0) - A(x_0)(x - x_0) = 0$$

Sélection d'un point aléatoire sur une courbe elliptique Les algorithmes que nous décrirons par la suite demandent souvent de choisir un point au hasard sur une courbe elliptique (définie sur un corps fini).

Pour ce faire l'idée est la suivante : on commence par choisir une abscisse au hasard, et on regarde quelles sont les ordonnées qui conviennent pour celle-ci (il peut y en avoir zéro, une ou deux). S'il n'y a pas d'ordonnée qui convient, on répète le processus avec une abscisse aléatoire, sinon, on prend une des ordonnées.

Dans le cas où car($\mathbb{K}$) = 3, comme les x et les y dans l'équation de Weierstrass sont indépendants, cela revient à calculer la racine carrée de $x^3 + b_2x^2 + b_4x + b_6$ dans $\mathbb{K}$. Néanmoins, le calcul d'une racine carrée dans un corps fini n'est pas une chose aisée, et [4] indique qu'il est en réalité plus simple de prendre le problème en sens inverse et de résoudre l'équation du troisième degré. Nous suivons donc leur méthode. Il pourrait être intéressant cependant d'essayer avec différentes méthodes de résolution d'équation du troisième degré ou de recherche de la racine carrée.

Avertissement Pour des raisons de taille du rapport, nous n'avons pas inclu ici tous les algorithmes et formules utilisés. Nous nous sommes contentés ici de présenter soit des algorithmes simples mais essentiels, à titre d'exemple et pour leur clarté, soit des algorithmes qui ne sont que peu décrits dans la littérature. Nous renvoyons le lecteur aux différents articles cités dans la bibliographie pour plus de détails et pour les descriptions des autres algorithmes.

Chapitre 2

Le couplage de Weil

Le couplage de Weil a de nombreuses utilisations en cryptologie : il est par exemple utilisé pour ramener le problème du logarithme discret sur une courbe elliptique à celui du logarithme discret dans un corps fini, ce qui donne une attaque potentiellement dangereuse[1] contre les systèmes basés sur les courbes elliptiques supersingulières. Plus récemment, il a été utilisé ([6]) pour réaliser l'un des premiers cryptosystèmes basés sur l'identité viable [2].

Nous l'avons implémenté sur les courbes de caractéristique 3 et l'utilisons pour réaliser un protocole de Diffie-Hellman tripartite en un seul tour dû à Joux ([1]).

2.1 Définition

Soit m un entier positif premier avec p. Soit $\mu_m \subset \overline{K}^*$ le groupe des racines $k^{\text{ème}}$ de l'unité.

Soit $P, Q \in E[m]$. Soit A et B des diviseurs de degré 0 tels que $A \sim (P) - (\mathcal{O})$, $B \sim (Q) - (\mathcal{O})$ ayant un support disjoint. Soit $f_A, f_B \in \overline{K}^*(E)$ telles que

$$\operatorname{div}(f_A) = mA \quad \text{et} \quad \operatorname{div}(f_B) = mB.$$

Le *couplage de Weil* e_m est une application

$$\begin{array}{rccc} e_m & : & E[m] \times E[m] & \longrightarrow & \mu_m \\ & & P\ ,\ Q & \longrightarrow & f_A(B)/f_B(A) \end{array}$$

[1] si les paramètres sont mal choisis.

[2] Il s'agit en fait d'une version légèrement modifiée du couplage de Weil. Elle est néanmoins toujours appelée ainsi. Nous notons le couplage de Weil e, et le couplage modifié $\hat{e}$ dans le présent document.

Notons que la valeur $e_m(P,Q)$ est indépendante du choix de A, B, f_A et f_B.

2.1.1 Quelques propriétés

1. identité : $\forall P \in E[m],\ e_m(P,P) = 1$;
2. alternance : $\forall P,Q \in E[m],\ e_m(P,Q) = e_m(Q,P)^{-1}$;
3. bilinéarité : $\forall P,Q,R \in E[m],\ e_m(P+Q,R) = e_m(P,R)e_m(Q,R)$ et $e_m(P,Q+R) = e_m(P,Q)e_m(P,R)$;
4. non-dégénérescence : $\forall P \in E[m], e_m(P,\mathcal{O}) = 1$ et si $e_m(P,Q) = 1$ pour tout $Q \in E[m]$, alors $P = \mathcal{O}$;
5. si $E[m] \subseteq E(K)$ alors $\forall P,Q \in E[m],\ e_m(P,Q) \in K$;
6. compatibilité : si $P \in E[m]$ et $Q \in E[mm']$, alors $e_{mm'}(P,Q) = e_m(P,m'Q)$;

2.1.2 Une modification du couplage de Weil

Pour certaines applications, on doit utiliser une forme bilinéaire non dégénérée définie sur un groupe cyclique, c'est-à-dire où la première propriété n'est pas vérifiée. C'est pourquoi nous définissons maintenant un couplage de Weil modifié, qui n'a pas cette propriété (voir [6]).

Soit donc $\hat{e}$ notre couplage de Weil modifié. On le définit de la manière suivante :

$$\hat{e}_m(P,Q) = e_m(P,\phi(Q))$$

où ϕ est un automorphisme sur le groupe des points de E.

Plusieurs automorphismes peuvent ici convenir. Ce choix peut avoir des répercussions sur la rapidité des calculs et sur la sécurité. Nous avons laissé cette question de côté et préféré considérer l'automorphisme comme un paramètre. Un exemple d'un tel automorphisme est donné dans [6].

2.2 Calcul

Nous nous intéressons maintenant au calcul effectif du couplage de Weil. L'algorithme décrit ici est dû à Miller.

2.2.1 Calcul de la fonction d'un diviseur principal

Un diviseur de degré nul peut être écrit $D = (P) - (\mathcal{O}) + \mathrm{div}(f)$ pour un $P \in E$ unique et $f \in \overline{K}(E)$. C'est la *forme canonique* de D.

Addition de deux diviseurs sous forme canonique

Soient D_1 et D_2 deux diviseurs de degré 0. $D_1 = (P_1) - (\mathcal{O}) + \mathrm{div}(f_1)$ et $D_2 = (P_2) - (\mathcal{O}) + \mathrm{div}(f_2),\ P_1, P_2 \in E,\ f_1, f_2 \in \mathbb{K}(E)$. On suppose que $P_1 \neq \mathcal{O}$ et $P_2 \neq \mathcal{O}$. Alors

$$D_1 + D_2 = (P_3) - (\mathcal{O}) + \mathrm{div}(f_1 f_2 f_3)$$

où $P_3 = P_1 + P_2$ et $f_3 = \frac{l}{v}$, l étant l'équation de la droite passant par P_1 et P_2 ; v l'équation de la verticale passant par P_3 (si $P_3 = \mathcal{O}$, $v = 1$).

Déterminer la fonction d'un diviseur principal

Soit $D = \sum_{i=1}^{n} a_i(P_i)$ un diviseur principal. Nous cherchons $f \in \overline{\mathbb{K}}(E)$ telle que $D = \mathrm{div}(f)$. La méthode est la suivante :

1. Ecrire D sous la forme $\sum_{i=1}^{n} a_i((P_i) - (\mathcal{O}))$. (Ce qui est possible car D est de degré nul).
2. pour chaque i entre un et n, calculer $P_i' \in E$ et $f_i \in \overline{K(E)}$ tels que

   $$a_i((P_i) - (\mathcal{O})) = (P_i') - (\mathcal{O}) + \mathrm{div}(f_i)$$

   Cela se fait en prenant une chaine additive $1 = d_1, d_2, \ldots, d_t = a_i$ (c'est-à-dire que pour chaque $d_j,\ j \geq 2$ est une somme $d_j = d_k + d_l,\ k < j,\ l < j$). On utilise alors la méthode décrite ci-dessus pour calculer les $d_j((P) - (\mathcal{O}))$, $j = 1, 2, \ldots, t$.
3. On fait finalement l'addition des $(P_i') - (\mathcal{O}) + \mathrm{div}(f_i),\ 1 \leq i \leq n$.

2.2.2 Calcul du couplage de Weil

Nous avons maintenant les briques nécessaires pour calculer le couplage de Weil :

Prenons des points $T, U \in E$ tels que $P + T \neq U,\ P + T \neq Q + U,\ T \neq U,\ T \neq Q + U$.

Soit $A = (P + T) - (T)$. Comme

$$A - (P) + (\mathcal{O}) = (P + T) - (T) - (P) + (\mathcal{O}) \in D_l,$$

$A \sim (P) - (\mathcal{O})$. De la même manière soit $B = (Q + U) - (U)$. Donc $B \sim (Q) - (\mathcal{O})$.

Soient $f_A,\ f_B \in \overline{K}^*(E)$, déterminées par la méthode ci-dessus, telles que

$$\mathrm{div}\,(f_A) = mA \quad \text{et} \quad \mathrm{div}\,(f_B) = mB.$$

Alors,

$$e_m(P,Q) = \frac{f_A(B)}{f_B(A)} = \frac{f_A((Q+U)-(U))}{f_B\left((P+T)-(T)\right)} = \frac{f_A(Q+U)f_B(T)}{f_A(U)f_B(P+T)}$$

2.3 Implémentation en caractéristique 3

Nous avons réalisé une implémentation en caractéristique 3 qui peut se diviser en deux parties : une partie de bas niveau constitue en fait une implémentation des corps finis de caractéristique 3 tandis qu'une autre partie vient par dessus et implémente les fonctions qui nous sont nécessaires sur les courbes elliptiques de caractéristique 3.

2.3.1 Implémentation des corps

Les corps sont représentés en base polynomiale, c'est-à-dire qu'un élément de $\mathbb{F}_p^m$ est un vecteur de m éléments de $\mathbb{F}_p$. Traditionnellement, les éléments de cette liste étaient placés dans un tableau d'entiers, ce qui est une idée naturelle. Néanmoins, si l'on pense que les entiers sont stockés, sur un ordinateur grand-public actuel, sur 32 bits (mêmes les entiers plus courts, pour des raisons d'alignement dans la mémoire) et qu'un élément de $\mathbb{F}_3$ tient facilement sur 2 bits, on constate que dans notre cas cela entraîne un certain gaspillage de la mémoire.

L'idée qui vient spontanément à l'esprit est de ranger plusieurs éléments dans un mot de 32 bits, afin d'économiser la place. Cependant, cela ne doit pas se faire au détriment des performances, qui restent le critère numéro un. En partant de l'idée que le processeur a moins de mots mémoires à charger, on aimerait même améliorer celles-ci ! On souhaiterait en outre traiter nos éléments « en parallèle », un peu à la manière d'une machine vectorielle : on peut espérer effectuer le même traitement sur tous les éléments que contient un mot simultanément.

C'est justement la direction prise dans [16] : les auteurs y montrent comment additionner deux mots contenant chacun dix éléments de $\mathbb{F}_3$ en environ cinq opérations. Malheureusement tout n'est pas encore parfait et ils ne fournissent une multiplication parallèle rapide qu'avec quatre éléments par mot... Leur idée est d'utiliser une représentation redondante des éléments de $\mathbb{F}_3$: comme de toute façon, pour stocker le bit et demi d'information que représente un élément de $\mathbb{F}_3$, il nous faut deux bits de mémoire (on ne fait pas de compression !), autant utiliser le demi-bit restant. On décide donc que la valeur "3" est équivalente à 0. Toutes les valeurs qu'un groupe de deux bits

peut prendre ont ainsi un sens : $00b = 11b = 0$; $01b = 1$ et $10b = 2$ (Voir la figure 2.1).

L'algorithme d'addition présenté dans [16] peut s'écrire ainsi en pseudo-C :

```
res = op1 + op2 ;
r = (res » 2) & 0x9249249 ;
res = res - 3*r ;
```

On le voit, l'espace interstitiel entre deux éléments reçoit les retenues éventuelles de l'addition de deux éléments et empêche des collisions désastreuses. Les retenues, qui indiquent où l'on a dépassé 3 et montrent donc où les réductions modulaires doivent être effectuées, sont ensuite récupérées et on soustrait trois là où c'est nécessaire, terminant l'addition modulaire.
Notons que si la soustraction peut être effectuée par un algorithme similaire, elle nécessite la présence d'un bit libre pour la retenue *avant* les éléments. Si c'est possible ici comme on a deux bits libres sur les 32 et que l'on peut donc décaler nos éléments, il est plus simple (et c'est plus facilement transposable à d'autres tailles de mots) de prendre l'opposé du second opérande (un simple *xor*) avant de faire une addition.

FIG. 2.1 – L'algorithme d'addition de deux mots (32 bits) contenant 10 éléments

Après avoir essayé différentes idées (recherche d'une addition avec seize éléments par mot, ...) nous avons finalement choisi un compromis : notre implémentation stocke donc dix éléments (avec la représentation redondante indiquée ci-dessus) dans un mot, en laissant un espace d'un bit entre deux éléments (et deux bits inutilisés). Ce qui permet d'avoir des addition, soustraction, négation et multiplication « externe » rapides au prix d'un léger surcoût dans la multiplication « interne » par rapport à l'article cité précédemment.

En effet nous avons remarqué qu'avec notre représentation redondante, la multiplication externe est très efficace : multiplier tous les éléments d'un mot par 0, 1, ou 2 ne pose pas de problèmes : les deux premiers cas sont bien sûr triviaux tandis que le dernier peut se faire simplement en effectuant un *ou exclusif* des bits qui nous intéressent et de 1.

Nous utilisons donc cette multiplication externe pour réaliser la multiplication de deux mots (une multiplication de polynômes donc) de manière classique. Cette multiplication de deux mots peut à son tour servir de brique de base pour l'implémentation d'une multiplication interne "complète" naïve (ce que nous avons fait afin de pour consacrer un peu de temps à d'autres parties intéressantes) ou utilisant la méthode de Karatsuba.

2.3.2 Fonctions sur les courbes ellitiques

Les fonctions sur les courbes elliptiques sont généralement des implémentations directes des algorithmes données plus tôt. Elles s'appuyent sur des structures de données basée sur des pointeurs pour éviter des copies inutiles de structures.

2.3.3 Structures de données

Nos structures de données sont, à l'exception de celles pour les corps finis décrites précédemment et sur lesquelles nous ne reviendrons pas, simples. Elle sont principalement constituées de structures C, que l'on passe par référence. Cela évite des recopies à chaque appel de fonction et donc gagne du temps.

Afin toujours d'éviter les copies et donc les allocations et déallocations de petites quantités de mémoire, on peut avoir, par exemple quand un même polynôme est utilisé plusieurs fois, plusieurs pointeurs sur un même objet, quand on sait que cet objet ne sera pas modifié. Le problème est d'éviter les fuites de mémoire : en effet, une fonction ne peut du coup supprimer un objet lorsqu'elle n'en a plus l'usage. Il faut donc savoir quand on peut supprimer les objets. Nous avons donc introduit un mécanisme simple de *glanage de cellules* en introduisant un compteur de réferences dans la structure des polynômes. Au lieu de copier le polynôme, on se contente d'incrémenter son compteur. Au lieu de le détruire, on décrémente le compteur et la libération de la mémoire n'intervient que lorsque celui-ci tombe à zéro.

Une idée pour économiser encore de tels appels, si l'on dispose d'une quantité de mémoire suffisante, pourrait être de calculer une borne sur la quantité de mémoire nécessaire, de l'allouer au début et de l'utiliser de la manière où on l'entend.

Chapitre 3

Le protocole de Joux

Le protocole de Joux est l'équivalent du protocole de Diffie-Hellman dans le cas où l'on a trois personnes qui doivent se mettre d'accord sur une même clef pour, par exemple, un algorithme de chiffrement symétrique. Il est basé sur une utilisation élégante des couplages qui fait qu'il ne nécessite de chaque partie que l'envoi d'un seul message. Comme le protocole de Diffie-Hellman, il est sensible à une *attaque de l'intercepteur*.

3.1 Description

Nous avons donc trois parties, Alice, Bob et Carol qui veulent avoir une clef secrète commune. Ils choisissent chacun un entier dans Z_q^*, respectivement a, b, c. On suppose que les autres paramètres (courbe elliptique, point sur la courbe, application bilinéaire $\hat{e}$, ...) sont fixés (par exemple par le logiciel qu'ils utilisent), sinon ils peuvent se mettre d'accord par le réseau, ces para-mètres n'étant pas nécessairement secrets. Le protocole est alors le suivant :

Alice envoie aP à Bob et Carol.

Bob envoie bP à Alice et Carol.

Carol envoie cP à Alice et Bob.

Ils calculent ensuite tous trois la clef secrète : Alice calcule $K_A = \hat{e}(bP, cP)^a$; Bob calcule $K_B = \hat{e}(aP, cP)^b$ et Carol calcule $K_C = \hat{e}(aP, bP)^c$. $\hat{e}$ étant bilinéaire, $K_A = K_B = K_C = \hat{e}(P, P)^{abc}$ peut servir en tant que clef partagée.

Le couplage de Weil modifié décrit précédemment convient en tant qu'application $\hat{e}$. Il n'est néanmoins pas le seul. Citons à titre d'exemple le couplage de Tate.

3.2 Notes sur l'implémentation

Notre implémentation du protocole de Joux est plus un exemple d'utilisation de notre couplage de Weil qu'une réalisation utilisable, pour la simple raison qu'il ne s'agit que d'une simulation où les trois parties sont dans le même programme...

En effet, si les fonctions assurant la partie "mathématique" sont bien là et sont les mêmes que celles que nous aurions utilisées en réseau, nous n'avons pas réalisé un programme qui utilise réellement le réseau pour des questions de portabilité : en effet un tel programme devrait, pour éviter un *deadlock*, à un moment donné à la fois émettre sur le réseau et attendre une donnée sur ce même réseau. Les solutions pour résoudre un tel problème, si elles sont bien connues, ne sont malheureusement pas utilisables à la fois avec Unix et Windows...

Nous avons donc réalisé un canevas simulant, moyennant les paramètres adéquats (courbe elliptique, automorphisme pour la modification du couplage de Weil, ...), le déroulement de ce protocole. Les fonctions réalisant le protocole seraient néamoins directement réutilisables dans une implémentation en réseau.

Conclusion et perspectives

Nous avons donc expliqué la manière de réaliser une implémentation efficace du couplage de Weil en caractéristique trois. Nous avons vu que le fait d'être dans ce cas particulier permet de nombreuses simplifications dans les algorithmes et donc des calculs plus rapides finalement par rapport à une implémentation "généraliste".

Comme nous l'avons indiqué, il est également fort utile d'avoir une représentation efficace des corps finis en caractéristique 3. Là aussi le fait d'être dans cette caractéristique bien particulière suggère souvent des algorithmes particuliers.

L'implémentation dans laquelle nous avons mis en oeuvre les techniques ici décrites, outre l'implémentation *stricto-sensu* du couplage de Weil, offre une base de développement qui permet d'essayer de nouvelles idées pour accélérer encore le calcul sans avoir à tout écrire : il suffit maintenant de remplacer ou de modifier la fonction que l'on veut améliorer.

En dehors du couplage de Weil en lui-même, les diverses fonctions que nous avons créées sont réutilisables et pourraient par exemple probablement être utilisées pour bâtir rapidement d'autres fonctions de haut niveau, par exemple le couplage de Tate.

Sur un plan plus personnel, ce stage nous a permis de nous familiariser avec les courbes elliptiques et leurs applications cryptographiques, les systèmes basés sur l'identité, mais nous avons également appris bien des choses sur l'utilisation (Pari, Gmp, ...) et l'implémentation des systèmes de calculs formels.

Annexe A

Notes diverses sur l'implémentation

A.1 Sécurité "dans le monde réel"

Notre implémentation est expérimentale : son but est de voir l'influence des algorithmes et des structures de données sur les performances. Elle n'est en aucun cas, au stade actuel, utilisable pour une application cryptographique réelle car probablement vulnérable à de nombreuses attaques, par exemple des *timing attacks*, le temps de calcul pouvant varier selon les paramètres. Le problème de la résistance aux attaques physiques est d'ailleurs, un sujet de recherche en soi et dépasse le cadre de ce travail.

A.2 Différentes versions

Le développement s'est effectué en différentes étapes. A part pour les premières, le système de gestion de versions *subversion* a été utilisé. Il est donc possible de retrouver les différentes versions depuis le *repository*, par exemple si l'on veut récupérer la version n'ayant pas encore l'implémentation évoluée des corps.

A.3 Options de compilation

Pour des raisons de performances, un certain nombre de fonctions sont indiquées comme "inline" dans le code source.

Pour des raisons de sûreté du développement, un certain nombre de *warnings* sont activés, de même que l'option indiquant de traiter tous les war-

nings comme des erreurs. Sur certaines plateformes avec des compilateurs différents, il est possible que cela empêche la compilation, il peut alors falloir désactiver cette option.

A.4 Multithreading

Les fonctions implémentées ne sont pas réentrantes, il ne faut donc pas les utiliser simultanément dans différents *threads*.

A.5 Utilisation avec Windows

Bien que le développement se soit effectué sous Linux, il doit être possible d'utiliser le code sous Windows. La bibliothèque *gmp* est disponible sous Windows (voir [14]). Il suffit donc de compiler nos fichiers avec cette bibliothèque. Le code source est constitué de *C* ANSI et ne devrait donc pas poser de problème de compilation.

A.5.1 Documentation et Doxygen

Nous avons utilisé *Doxygen* pour documenter l'API fournie par notre implémentation : il est donc possible d'avoir en différents formats une liste des fonctions réutilisables avec des indications sur leurs paramètres et leurs effets.

La documentation doxygen n'explique généralement pas le fonctionnement interne du code, ceci étant expliqué dans le présent rapport.

A.6 Conseils pour les développeurs voulant modifier le code fourni

Les options de compilation (voir ci-dessus) sont volontairement restrictives : ne les désactivez pas ! Il est suffisamment difficile dans ce genre de programme de trouver d'ou vient l'erreur dans la partie "mathématique" sans que l'on se rende la tâche plus difficile en y ajoutant des erreurs de programmation qui auraient pu être évitées par l'usage des warnings adéquats.

Pour ce qui est de la traque des fuites de mémoire, notons que l'utilisation de l'outil encore trop peu connu *Valgrind* peut souvent aider. Cet outil vous permettra également de trouver des erreurs qui seraient souvent sinon passées inaperçues. Pensez à l'utiliser.

Annexe B

Notes sur PARI

PARI/GP est un système de calcul formel, conçu notamment pour faire rapidement des calculs en théorie des nombres. Comme son nom l'indique, PARI/GP est constitué de PARI, une bibliothèque de fonctions mathématiques, et de *gp*, un shell qui permet l'accès à ces fonctions et qui dispose d'un langage de programmation propre, GP.

Nous avons utilisé PARI (la bibliothèque) pour programmer une version naïve du couplage de Weil sur $\mathbb{F}_p$ (p premier). Ce chapitre donne quelques clefs pour commencer rapidement à utiliser PARI à partir du langage C, suivies de quelques commentaires.

B.1 *Getting Started* avec la bibliothèque PARI et les courbes elliptiques

B.1.1 Fichiers d'entêtes et liens

Avant d'utiliser les fonctions de PARI, il est nécessaire d'indiquer au compilateur leurs déclarations via la directive préprocesseur suivante :
`#include<pari/pari.h>`

Pensez également à fournir `-lpari` à l'éditeur de liens.

B.1.2 Types

Le type GEN

PARI définit un seul type C, appelé `GEN`. C'est en fait un `long` qui est utilisé pour stocker un pointeur... *Le type GEN de PARI est en fait un simple pointeur, avec tout ce que cela implique.*

Déclarer et initialiser des variables GEN

La déclaration d'un `GEN` se fait de manière classique, par exemple :
`GEN foo, bar;`

L'initialisation des tableaux se fait grâce à la fonction **cgetg** :
`foo = cgetg(6, t_VEC);` (déclare un tableau de taille 5).

Conversions entre les types du C standard et le type GEN

La manière la plus simple de définir un entier PARI, s'il n'est pas trop grand, est de le définir comme un entier C et de le convertir. La fonction de conversion est `stoi`.

La conversion en sens inverse se fait avec la fonction `itos`. Notez que l'utilisation de cette fonction provoque l'arrêt de votre programme avec le message **** impossible assignement I->S* s'il n'est pas possible de faire tenir la valeur dans un entier long de C.

B.1.3 Les courbes elliptiques

La courbe elliptique d'équation de Weierstrass $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ est représentée par un vecteur `E=[a_1,a_2,a_3,a_4,a_6]`.

Les points de la courbe sont représentés par des vecteurs du type `[x,y]`, sauf le point à l'infini qui est représenté par le vecteur `[0]`. Il faut donc faire attention à la taille du vecteur si l'on désire écrire une fonction qui utilise les coordonnées du point, sinon l'utilisation de cette fonction avec le point à l'infini provoquera un erreur de segmentation dans la plupart des cas. La fonction `long lg(GEN)` renvoie la taille du vecteur. (Rappelez-vous que PARI utilise le premier emplacement d'un tableau C pour stocker des informations, donc un point normal sera de taille 3 et le point à l'infini de taille 2 !)

B.1.4 Déboguage des programmes

C'est à notre avis là le principal problème de PARI : pour des raisons de performances, les fonctions de la bibliothèque ne font aucun test sur les arguments que l'on leur passe. Aussi, en cas d'erreur dans notre programme (par exemple un accès à un tableau non alloué), dans la plupart des cas, l'erreur se produira au sein d'une fonction de PARI. Cela rend des outils de déboguage comme *valgrind* ou *gdb* beaucoup plus difficiles à utiliser. D'autant plus que PARI rattrape les signaux pour afficher un message d'erreur[1]...

[1] C'est tellement exapérant que nous avons choisi de modifier la bibliothèque pour éviter cela : il suffit, du moins pour la version 2.1.5 de PARI, de commenter le `return signal(sig,f)` ; dans la fonction **os_signal** qui se trouve dans le fichier **src/language/es.c** à la ligne 2211.

Annexe C

Notes sur GMP

GMP, abréviation de *GNU Multi Precision*, est une bibliothèque bien connue permettant de faire de l'arithmétique sur les grands nombres.

GMP ne fournit que des fonctions sur les grands nombres, aussi bien entiers qu'en virgule flottante. Ces fonctions comprennent également un générateur de nombres aléatoires.

La sémantique des fonctions GMP est plus proche de l'assembleur que celles de PARI, en ce sens que les premiers paramètres sont généralement des pointeurs sur les variables où l'on désire stocker le résultat. Ces fonctions ne renvoient en général rien (type *void*).

C.0.5 Fichiers d'entêtes et liens

Il faut faire l'inclusion
`#include <gmp.h>`
pour pouvoir utiliser les fonctions de GMP (en passant `-lgmp`) à l'éditeur de liens.

C.0.6 Types

Le seul type que nous avons utilisé dans notre réalisation est le type **mpz_t**, qui peut prendre un entier relatif de longueur arbitraire.

La déclaration se fait de la manière habituelle. L'initialisation (qui est en fait une allocation de mémoire et une mise à zéro) peut se faire avec des fonctions comme **mpz_init();**. On peut aussi donner une valeur initiale. Par exemple :

```
mpz_t foo, bar;
```

```
mpz_init(foo);              /* foo peut être utilisé. Il vaut 0. */

mpz_init_set_ui(bar, 37554); /* bar peut être utilisé. Il vaut 37554.
                                (notons que ce type d'initialisation
                                est limité par la taille des
                                types C.) */
```

Bibliographie

[1] Sattam S. Al-Riyami and Kenneth G. Paterson. Tripartite authenticated key agreement protocols from pairings, 2003. http://www.isg.rhul.ac.uk/~kp/cirenpaper.ps.

[2] Christian Batut Karim Belabas Dominique Bernardi Henri Cohen Michel Olivier. *User's Guide to PARI/GP*. http://www.parigp-home.de, Novembre 2000.

[3] Paulo S. L. M. Baretto, Hae Y. Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In Moti Yung, editor, *Proceedings of Crypto'2002*, volume 2442 of *LNCS*, pages 354–368. Springer, 2002.

[4] Paulo S. L. M. Barreto and Hae Yong Kim. Fast hashing onto elliptic curves over fields of characteristic 3. Cryptology ePrint Archive, Report 2001/098, 2001. http://eprint.iacr.org/.

[5] Dan Boneh. The decision diffie-hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423 of *LNCS*, pages 48–63, http://crypto.stanford.edu/~dabo/abstracts/DDH.html, 1998. Springer.

[6] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. In *Proceedings of Crypto'2001*. Elsevier, 2001.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Proceedings of Asiacrypt'01*, volume 2248 of *LNCS*, pages 514–532. Springer, 2001.

[8] Liqun Chen, Keith Harrison, Andrew Moss, David Soldera, and Nigel Smart. Certification of public keys within an identity based system. Technical report, HP Laboratories Bristol, Février 2003.

[9] Liqun Chen, Keith Harrison, David Soldera, and Nigel Smart. Application of multiple trust authorities in pairing based cryptosystems. Technical report, HP Laboratories Bristol, Février 2003.

[10] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In B. Honary, editor, *Cryptography and Coding 2001*, volume 2260 of *LNCS*, pages 360–363. Springer, 2001.

[11] Yvo Desmedt and Jean-Jacques Quisquater. Public-key systems based on the difficulty of tampering. In A. M. Odlyzko, editor, *Proceedings of Crypto'86*, Advances in Cryptology, pages 111–117. Springer, 1986.

[12] Martin Gagné. Identity-based encryption : a survey. *RSA Laboratories Cryptobytes*, 6(1) :10–19, été 2003.

[13] Steven Galbraith. Supersingular curves and the tate pairing.

[14] Torbjörn Grandlund. Gnu MP. `http://www.swox.com/gmp/#DOC`, 4.1.2 edition, Décembre 2003.

[15] Louis Claude Guillou and Jean-Jacques Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Proceedings of Crypto'88*, Advances in Cryptology, pages 216–231. Springer, 1988.

[16] Keith Harrison, Dan Page, and Nigel Smart. Software implementation of finite fields of characteristic three and for use in pairing-based cryptosystems. *LMS J. Comput. Math.*, 5 :181–193, Novembre 2002.

[17] Marc Joyce. Introduction élémentaire à la théorie des courbes elliptiques. Technical report, Groupe Crypto - Université Catholique de Louvain, 1995.

[18] Donald Ervin Knuth. *The Art of Computer Programming*, volume 2, chapter 4 : Arithmetic. Addison-Wesley, 3 edition, 1997.

[19] Alfred Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer internationnal series in engineering and computer science. Kluwer Academic Publishers, 1993. ISBN : 0-7923-9368-6.

[20] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley David Chaum, editor, *Proceedings of Crypto'84*, Advances in Cryptology, pages 47–51. Springer, 1984.

[21] David Singerman. Six lectures on fermat's last theorem. `http://www.maths.soton.ac.uk/staff/Leary/ma345/`. Il s'agit de notes de conférences. Contient une partie sur les tangentes des équations diophantiennes.