Discrétion assurée ?

• A. Joux
• C. Pierrot

« Je ne peux pas vous le dire à tous à la fois, et si vite que ça. Parce qu'un secret, ce n'est pas quelque chose qui ne se raconte pas. Mais c'est une chose qu'on se raconte à voix basse, et séparément. »

César [1]

Depuis un autre César et son célèbre chiffrement, les « codes secrets » intriguent, la cryptographie fascine. Entrez dans la confidence et cheminons ensemble à la découverte de cette science du secret. Comment ? Grâce aux différentes facettes d'un problème à l'intersection de la théorie des nombres et de l'algorithmique : le problème du logarithme discret.

Soient $(G, \times)$ un groupe cyclique et g un générateur de G. Considérons alors $x \mapsto g^x$, l'exponentiation discrète à la puissance x en base g, x étant un entier. Cette opération partage de nombreuses propriétés avec l'exponentiation ordinaire, comme l'égalité $g^{x+y} = g^x \cdot g^y$. L'inverse de cette opération consiste, étant donné un élément h dans G, à déterminer un entier x tel que :

$$h = g^x.$$

L'exponentiation discrète étant périodique, de période $|G|$, cet entier x n'est pas unique. En revanche $x \bmod |G|$ est bien déterminé : il sera appelé le *logarithme discret* de h en base g et noté $\ln_g(h)$. Par abus de langage, on confondra souvent ce logarithme avec son unique représentant dans l'intervalle $[0, |G| - 1]$. Comme pour les logarithmes classiques :

$$\ln_g(h \cdot j) \equiv \ln_g(h) + \ln_g(j) \pmod{|G|}.$$

On considère que le problème du logarithme discret est résolu dans un groupe G dès lors que pour tout élément de G il est possible de déterminer efficacement son logarithme discret (ou l'un de ses représentants). Lorsque l'ordre de G est connu, résoudre le problème du logarithme discret dans G, c'est simplement expliciter l'isomorphisme :

$$\begin{array}{ccc} \mathbb{Z}/|G|\mathbb{Z} & \to & G \\ x & \mapsto & g^x \end{array}$$

sous une forme calculatoirement efficace. Les récentes découvertes à ce sujet que l'on se propose de vous raconter ont eu un impact conséquent en cryptographie.

Mais... quel est le rapport vous demandez-vous ?

1. Logarithmes discrets et cryptographie

Au début du siècle dernier, Kraïtchik [12, Chap. V] manipulait déjà notre logarithme discret x, qu'il appelait l'indice de g^x dans le groupe ; ce qui valu, au passage, le nom de la méthode sur laquelle nous nous appuyons, la méthode du calcul d'indice. Toutefois, la réelle notoriété de ce problème remonte à la naissance de la cryptographie moderne.

À la recherche des problèmes difficiles. Lorsque vous souhaitez chiffrer et déchiffrer des messages (plus ou moins) secrets, signer un document numérique ou vous authentifier auprès de votre banque, vous [2] utilisez des techniques (ou *cryptosystèmes*) construites à partir d'hypothèses calculatoires réputées difficiles. Bien choisir ces hypothèses est plus délicat qu'il n'y paraît : il faut disposer d'un problème difficile à résoudre pour un attaquant alors que le déchiffrement doit être facile pour un utilisateur légitime muni de la bonne clef. Il nous faut donc disposer d'un cryptosystème pour lequel il est difficile de trouver la bonne clef, alors que par construction il est nécessairement facile de tester si une clef donnée est correcte. Du point de vue de la théorie de la complexité, cela veut dire que la cryptographie ne peut exister que

1. Marcel Pagnol, *César*, Livre de Poche n° 161, p. 115.
2. Pris au sens large, la tâche est généralement déléguée à votre ordinateur ou à votre carte bancaire.

si deux classes [3] bien connues **NP** (notre problème est dans **NP** car il est facile de tester si une clef est correcte) et **P** (on ne le veut pas dans **P** pour que la bonne clef ne se trouve pas facilement) sont distinctes. Or, cette question n'est pas résolue. Pire, elle est considérée comme le problème ouvert le plus important du domaine, à un point tel qu'elle fait partie des sept problèmes à 1 million de dollars du *Clay Millenium Challenge*.

Par conséquent, on ne sait pas aujourd'hui s'il existe de véritables problèmes difficiles sur lesquels appuyer les fondements de la cryptographie. On ne peut que choisir des problèmes qui ont été suffisamment étudiés pour que l'absence de solution soit un bon indice de leur difficulté. Pour cette raison, la cryptographie à clef publique repose généralement sur des problèmes issus des mathématiques. Par ailleurs, l'essentiel des calculs se fait sur des ordinateurs qui préfèrent manipuler un monde d'objets discrets plutôt qu'un monde continu. C'est ainsi que le vaste champ de la théorie des nombres nous fournit deux candidats incontournables : le problème de la factorisation des entiers et celui du logarithme discret.

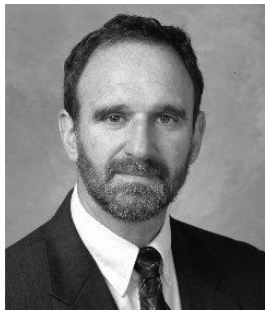
Les projecteurs se tournèrent en 1976 sur nos logarithmes, discrets jusque là, lorsque Diffie et Hellman proposèrent un mécanisme d'échange de clef entre deux personnes [5], sans connaissance d'un secret commun préalable. La cryptographie à clef publique était née ! Pour bien comprendre l'impact de ce résultat, très simple à expliquer par ailleurs, il faut se rappeler que, des siècles durant, de Jules César à la Seconde Guerre Mondiale, tous les échanges d'informations sensibles nécessitaient… un échange préalable, celui de la clef *secrète* du protocole. Le serpent se mordait la queue : comment communiquer de manière sécurisée avec quelqu'un que l'on n'avait jamais vu auparavant ? L'introduction de la cryptographie à clef publique grâce au logarithme discret permit de contourner cet écueil millénaire.

Échange de clefs de Diffie-Hellman. Si nous souhaitons créer une clef secrète commune en suivant le protocole de Diffie et Hellman, nous nous accordons tout d'abord sur un groupe cyclique G et un générateur g. Antoine choisit ensuite secrètement un entier a uniformément dans l'intervalle $[0, |G|-1]$, puis calcule g^a, qu'il envoie à Cécile sur un canal non sécurisé. De son côté, Cécile choisit de la même manière un entier aléatoire c et communique l'élément g^c à Antoine. Ainsi, chacun de nous deux est capable de calculer une valeur commune, à savoir :

$$(g^c)^a = g^{a\cdot c} = (g^a)^c$$

qui nous servira maintenant de secret commun.

Figure 1 – Whitfield Diffie et Martin Hellman.

La sécurité de ce système dépend de plusieurs hypothèses. Tout d'abord, il faut supposer qu'une tierce personne cherchant à mettre en défaut ce protocole se contentera d'écouter les communications sans chercher à les modifier. En effet, pour un attaquant actif, il est très aisé de parler à Cécile en se faisant passer pour Antoine et à Antoine en se faisant passer pour Cécile. De cette façon, il établit une clef commune avec chacun des deux de manière à faire suivre à son gré en les déchiffrant et rechiffrant tous les messages qui circuleront. En l'absence de secret commun préalable, nous ne nous rendrons compte de rien. C'est une faille bien connue de cet échange de clef (lorsqu'il est mal utilisé) que l'on appelle généralement *l'attaque par l'homme du milieu*.

Face à un attaquant passif, par exemple un lecteur indiscret, qui écouterait le canal et apprendrait donc g, g^a et g^c, que peut-on dire de la sécurité du système ? L'attaquant sera-t-il capable de calculer g^{ac} ? Pour que cela soit difficile, il est indispensable que le problème du logarithme discret le soit aussi. Sinon, il suffirait de calculer a (ou c), à partir de g^a, et d'en déduire le secret $g^{ac} = (g^c)^a$.

Peu après l'invention de l'échange de clef de Diffie et Hellman, Rivest, Shamir et Adleman [18] proposèrent une manière alternative de construire la cryptographie à clef publique en utilisant le problème de la factorisation des grands entiers. C'est

3. La classe **P** correspond aux problèmes résolubles en temps polynomial tandis que la classe **NP** regroupe ceux pour lesquels une solution, une fois donnée, est vérifiable en temps polynomial.

le fameux cryptosystème RSA, pouvant être utilisé soit pour chiffrer, soit pour signer des messages. Quelques années plus tard, ElGamal [6] montra qu'il était aussi possible de se baser sur le logarithme discret pour les mêmes utilisations.

Depuis, le logarithme discret reste un acteur majeur du domaine. Sa plus grande flexibilité figure parmi ses avantages. Il offre en effet plus de choix que la factorisation, puisqu'il autorise de nombreuses possibilités pour le groupe G : par exemple, le groupe des points rationnels d'une courbe elliptique (définie sur un corps fini) qui améliore en outre la sécurité du Diffie-Hellman. Sur une courbe elliptique générale, le problème est en effet plus difficile que dans un corps fini, dont la structure plus riche facilite la résolution. L'utilisation de telles courbes permet donc à niveau de sécurité égal de réduire la taille des clefs.

En près de quatre décennies, afin d'appréhender plus précisément la sécurité des protocoles cryptographiques, de nombreuses recherches se sont consacrées au calcul de logarithmes discrets. Deux grandes familles se distinguent :

- les algorithmes génériques, qui utilisent uniquement les opérations de groupe et la connaissance de $|G|$ et de sa factorisation,
- les algorithmes spécifiques, qui tirent parti de la description particulière du groupe considéré. La méthode du calcul d'indice donne ainsi un levier efficace principalement pour le cas des corps finis. Les récents progrès algorithmiques concernant les corps finis de petite caractéristique que nous aborderons plus loin font partie de cette famille.

2. Algorithmes génériques

À titre de comparaison future, il est bon d'introduire les algorithmes génériques qui n'utilisent pas la description spécifique du groupe G. Sans rentrer dans les détails, voici un bref point sur la question.

- La *méthode brutale* consiste simplement, pour retrouver le logarithme de h en base g, à énumérer les puissances successives $g^2, g^3, g^4, \cdots$ de g et à les comparer avec h. Sa complexité est en $O(|G|)$. Elle est bien entendu sans intérêt pratique.
- Les *méthodes en racine carrée*. Supposant connue la factorisation de $|G|$, Pohlig et Hellman (le même !) montrent en 1978 qu'il n'est pas plus compliqué de calculer des logarithmes dans G que dans tous ses sous-groupes d'ordre premier. Pour retrouver un logarithme discret dans le groupe tout entier, l'idée consiste à projeter l'élément dont on cherche le logarithme sur ses sous-groupes d'ordre premier, et à y calculer un petit nombre de logarithmes discrets dont l'assemblage fournira le résultat recherché. La même année, Pollard détaille comment résoudre le problème du logarithme discret dans un groupe d'ordre premier en $O(\sqrt{|G|})$ opérations : c'est la méthode Rho de Pollard. En combinant les algorithmes de Pohlig–Hellman [15] et Rho de Pollard [16], il est possible de calculer des logarithmes discrets en $O(\sqrt{p})$, où p est le plus grand facteur premier intervenant dans la factorisation de $|G|$.
- *Optimalité.* Shoup introduit en 1997 un cadre théorique, le modèle du groupe générique [19], pour étudier la complexité de ces algorithmes. Dans ce modèle, il montre qu'il n'est pas possible de descendre en dessous de la complexité en racine carrée obtenue par Pohlig, Hellman et Pollard. Notons que ces résultats d'optimalité ne s'appliquent pas pour les méthodes de calcul qui exploitent de façon fine la description de G.

3. Méthode du calcul d'indice

3.1 – Description générale

Cette méthode fonctionne aussi pour factoriser des entiers, mais nous ne nous intéresserons ici qu'au cas du logarithme discret, encore une fois dans un groupe cyclique G engendré par g. Les algorithmes de cette famille se découpent en plusieurs phases.

1. **Phase préliminaire.** Tout algorithme par calcul d'indice commence par fixer la description de G qui sera utilisée pour la suite, et qui peut différer de celle initialement fournie. Par exemple, on pourra choisir de travailler sur $\mathbb{F}_{2^4}$ vu comme $\mathbb{F}_2[Y,Z]/(Y^2+Y+1, Z^2+Z+Y)$ même s'il est initialement donné par $\mathbb{F}_2[X]/(X^4+X+1)$. On détermine aussi un sous-ensemble relativement petit d'éléments particuliers de G, que l'on nomme la base de lissité. Celle-ci est constituée d'éléments eux-même considérés comme petits, en un sens restant à définir.
2. **Phase de création de relations (ou phase de crible).** L'objectif de cette étape est de

créer un grand nombre de relations multiplicatives entre éléments de la base de lissité. Si la base utilisée est $\{g_i,\ i \in I\}$ nous nous intéresserons à des équations de la forme :

$$\prod_{i\in I} {g_i}^{m_i} = \prod_{i\in I} {g_i}^{n_i}. \qquad (1)$$

En prenant le logarithme discret de chacun des deux côtés nous en déduisons que :

$$\sum_{i\in I} m_i \ln_g g_i \equiv \sum_{i\in I} n_i \ln_g g_i \mod |G|.$$

On obtient ainsi une équation linéaire entre les logarithmes des g_i, qui sont nos inconnues. La phase de crible s'arrête lorsque l'on a récolté un nombre suffisant d'équations de cette forme, de sorte de pouvoir en extraire une solution unique (à constante multiplicative près). Autrement dit, le rang du système doit être égal au nombre d'inconnues moins un, modulo chacun des facteurs de $|G|$.

Notons qu'à l'issue de l'algèbre linéaire, puisque l'on obtient un élément arbitraire du noyau de la matrice, les logarithmes retrouvés ne le sont qu'à constante multiplicative près. Autrement dit, au lieu d'obtenir $\{\ln_g(g_i),\ i \in I\}$, on a $\{\lambda \ln_g(g_i),\ i \in I\}$ pour un certain λ. Toutefois, il est facile de retrouver les bons logarithmes, en supposant que g appartienne à la base de lissité – le logarithme de g, qui vaut 1, nous donnant la valeur de ce λ artéfact.

3. **Phase d'algèbre linéaire.** Cette étape cherche à résoudre le système linéaire issu des relations, afin d'obtenir les logarithmes discrets de tous les éléments de la base de lissité [4]. Une observation importante qui apparaît dans la plupart des algorithmes par calcul d'indice est la suivante : peu d'éléments interviennent dans chacune des relations, le système produit est donc *creux*. Cela accélère grandement l'algèbre linéaire car la complexité de l'algorithme de résolution est alors quadratique et non cubique comme dans le cas général.
4. **Calcul d'un logarithme individuel (ou phase de descente).** Afin de résoudre réellement le problème du logarithme discret dans G, nous devons pouvoir retrouver le logarithme d'un élément arbitraire et pas seulement des éléments de la base de lissité. Soit $z \in G$ un tel élément arbitraire et x son logarithme en base g. En quelques mots, l'idée consiste alors à décomposer z en produits d'autres éléments qui peuvent être considérés comme plus petits que lui. En itérant ce procédé, z s'exprime finalement comme produit d'éléments de la base de lissité $g^x = z = \prod_{i\in I} {g_i}^{\alpha_i}$. Puisque l'on connait les logarithmes discrets de ces derniers, on retrouve alors facilement x en écrivant $x = \sum_{i\in I} \alpha_i \ln g_i$.

3.2 – Lissité et complexité

Les algorithmes par calcul d'indice reposent sur l'idée de décomposer des éléments [5] comme produits d'éléments considérés comme petits. Les éléments qui peuvent se factoriser de cette manière sont dits *lisses*. Un problème essentiel pour l'analyse de ces algorithmes consiste donc à estimer la probabilité d'obtenir de tels éléments lisses. Dans de nombreux cas, on procède heuristiquement en supposant que les éléments créés se comportent comme des éléments aléatoires de même taille. Bien qu'inélégante, car non prouvée, cette heuristique a permis d'obtenir de nombreux progrès algorithmiques et a conduit à un grand nombre de factorisations explicites d'entiers et de calculs de logarithmes discrets.

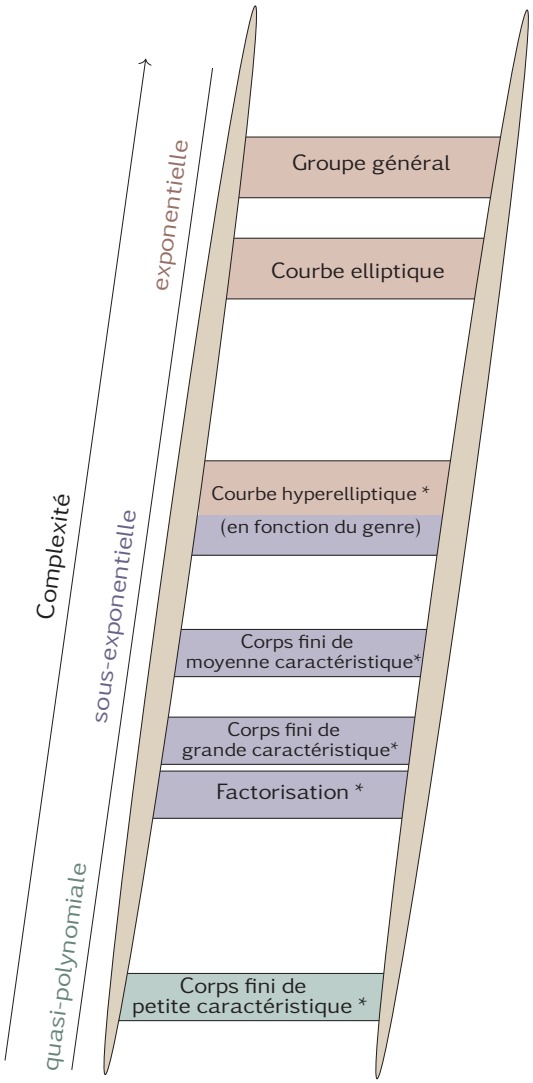
Dépendre d'une telle heuristique est inconfortable, et nous aimerions nous en abstraire. Pourtant, les algorithmes rigoureux qui existent actuellement se montrent bien moins efficaces que leurs homologues heuristiques. De manière tout à fait surprenante, les dernières avancées spectaculaires concernant les corps finis de petite caractéristique reposent justement sur le fait que, lorsque certaines heuristiques deviennent fausses (et que les éléments engendrés par la phase de création de relations ne se conduisent pas comme des éléments aléatoires), il devient possible de retourner cette faille à notre avantage.

La complexité des algorithmes par calcul d'indice s'exprime généralement avec une notation particulière qui provient justement des estimations de probabilité de lissité d'éléments aléatoires. Soyons un peu plus précis : un entier est dit y-lisse si tous ses facteurs premiers sont inférieurs à y tandis qu'un polynôme sur un corps fini est dit m-lisse si tous ses facteurs irréductibles sont de degré

4. Ou, au moins, une grande partie d'entre eux. En effet, certaines propriétés de la phase de crible peuvent mener à écarter quelques éléments de la base de lissité, qui ne seront présents dans aucune équation.
5. En pratique : des entiers, des idéaux ou des polynômes.

au plus m. Canfield, Erdös et Pomerance [4] calculèrent en 1983 les probabilités de lissité d'entiers. Une dizaine d'années plus tard, Panario, Gourdon et Flajolet [13] généralisèrent cette idée à la lissité d'un polynôme sur un corps fini.

FIGURE 2 – Echelle schématique des complexités asymptotiques actuelles de la factorisation et du problème du logarithme discret dans différents groupes. L'astérisque indique l'appartenance à la famille des algorithmes par Calcul d'Indice.



Ces deux résultats étonnamment proches l'un de l'autre peuvent se résumer ainsi : dans une large plage de paramètres, la probabilité qu'un entier aléatoire inférieur à x soit y-lisse (respectivement qu'un polynôme aléatoire de degré inférieur à n soit m-lisse) est $u^{-u+o(1)}$ où u est donné par $u = \ln x/\ln y$ (respectivement $u = n/m$). L'utilisation de ce résultat est simplifiée par la notation :

$$L_q(\alpha, c) = \exp\Big((c + o(1))(\ln q)^{\alpha}(\ln\ln q)^{1-\alpha}\Big)$$

où α et c sont des constantes telles que $0 \leqslant \alpha \leqslant 1$ et $c > 0$ et où $o(1)$ tend vers zéro pour $q \to \infty$. L'abréviation $L_q(\alpha)$ sera utilisée lorsque l'on voudra négliger c.

Dans la première phase d'un algorithme par calcul d'indice, le nombre d'opérations à effectuer est grossièrement égal au nombre de relations que l'on souhaite obtenir multiplié par l'inverse de la probabilité ; il est donc normal que l'on retrouve la notation L_q dans la complexité asymptotique finale de l'algorithme. Habituellement, q désigne le cardinal du groupe dans lequel nous souhaitons résoudre le problème du logarithme discret. Pour comparer un algorithme qui nécessite $L_q(\alpha, c)$ opérations au total à d'autres du même type, il faut d'abord étudier le premier paramètre, puisqu'il gouverne le passage d'un algorithme exponentiel (en temps) à un algorithme polynomial. Plus précisément, si α tend vers 1, $L_q(\alpha)$ devient exponentiel en la taille de q, c'est-à-dire en $\ln q$. Remarquez que $\ln q$ est le nombre de bits nécessaires pour encoder les éléments du groupe en question. Il est donc naturel que ce soit cette valeur que l'on considère lorsque l'on exprime la complexité des algorithmes. De même, lorsque α vaut 0, $L_q(\alpha)$ est *polynomial* en $\ln q$. On qualifie de *sous-exponentiel* tout algorithme de calcul de logarithme discret sur un groupe de taille q qui a une complexité en $L_q(\alpha)$ avec $0 < \alpha < 1$. De même, on qualifie de *quasi-polynomial* tout algorithme dont la complexité est en $L_q(o(1))$.

La Figure 2 donne une indication des complexités actuelles que l'on peut obtenir.

4. Calcul d'indice dans les corps finis

4.1 – La récolte des relations, une étape clef

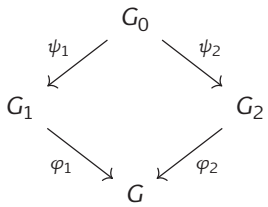
Les algorithmes de calcul d'indice reposant sur la construction de relations multiplicatives, les différentes méthodes pour obtenir de telles relations forment un fil conducteur dans l'historique du logarithme discret dans les corps finis.

Les premiers algorithmes en $L(1/2)$. Les premiers algorithmes de calcul d'indice pour les corps finis ont été proposés à la fin des années 70 par Adleman [1] pour les corps de cardinalité première. Par exemple, pour calculer le logarithme discret de u modulo un nombre premier p, l'approche la plus simple consiste à prendre un entier aléatoire a à calculer un représentant de u^a modulo p dans l'intervalle $[0, p-1]$ et à vérifier si

$$u^a \bmod p = \prod g_i$$

avec g_i des nombres premiers inférieurs à une certaine borne B. On obtient ainsi des relations dans une base de lissité formée des nombres premiers plus petits que B et de u. En supposant que g est bien dans la base de lissité, on obtiendra directement le logarithme recherché sans qu'une phase de descente ne soit nécessaire [6]. Notons que pour la plupart des valeurs de a, u^a ne sera pas B-lisse, et sera donc écarté. La complexité dépendra donc de la probabilité qu'un entier aléatoire inférieur à p soit B-lisse. Bien que primitive, cette approche permet déjà d'obtenir une complexité sous-exponentielle, de la forme $L_p(1/2)$. Elle se généralise à de nombreux corps finis et présente l'intérêt de fournir un algorithme probabiliste rigoureusement prouvé [17], contrairement aux autres algorithmes de la même famille, qui, bien que plus performants, reposent encore sur certaines hypothèses heuristiques.

La méthode du diagramme commutatif. Une idée essentielle pour améliorer la recherche de relations multiplicatives consiste à préalablement décrire le groupe G de façon à pouvoir l'inclure dans un diagramme commutatif de la forme suivante :



Dans ce diagramme commutatif, G_0, G_1 et G_2 sont des groupes bien choisis (la plupart du temps, il s'agit même d'anneaux) et toutes les applications sont des morphismes. Ainsi, pour tout x dans G_0, on obtient une égalité dans G :

$$\varphi_1(\psi_1(x)) = \varphi_2(\psi_2(x)).$$

Comme dans la description générale, il n'est gardé qu'une petite partie de ces relations. À cette fin, il faut distinguer dans chaque groupe intermédiaire G_1 et G_2 un petit sous-ensemble d'éléments, disons $B_1 \subset G_1$ et $B_2 \subset G_2$. Le sous-ensemble $\varphi_1(B_1) \cup \varphi_2(B_2)$ de G est alors la base de lissité de cet algorithme. Cela signifie que l'on ne conserve que les relations pour lesquelles $\psi_1(x)$ se décompose comme $\prod_{b_{1,i} \in B_1} b_{1,i}$ et $\psi_2(x)$ comme $\prod_{b_{2,i} \in B_2} b_{2,i}$. On obtient ainsi une relation entre produits dans G :

$$\prod_{b_{1,i} \in B_1} \varphi_1(b_{1,i}) = \prod_{b_{2,i} \in B_2} \varphi_2(b_{2,i}).$$

Cette méthode se révèle assez générale, mais les choix de G_0, G_1 et G_2 sont spécifiques à chaque type de groupe. Nous donnons un exemple concret de ces choix au paragraphe suivant. En 1984, Coppersmith obtient ainsi les premiers algorithmes de complexité $L_q(1/3)$ où q est la taille du corps considéré. Ce résultat, limité au cas des corps de caractéristique 2, est ensuite étendu progressivement à tous les corps finis. Ainsi, le crible par corps de fonctions permet de calculer des logarithmes dans tous les corps finis de petite caractéristique, tandis que le crible par corps de nombre (et ses variantes) concerne les corps de caractéristique moyenne à grande.

Petite, moyenne et grande caractéristiques. Il est probablement temps d'examiner cette notion de taille de caractéristique : en réalité, lorsque l'on parle de petite ou de grande caractéristique, on considère implicitement la taille *relative* de cette caractéristique avec celle du degré de l'extension, pour une taille de corps fixée. Ainsi, si l'on s'intéresse au corps fini $\mathbb{F}_{p^n}$ et que l'on souhaite évaluer la taille de sa caractéristique p, on commencera par écrire p sous la forme $p = L_{p^n}(l, c)$ avec $0 < l < 1$ et c une constante proche de 1. Si $l < 1/3$, on parlera de petite caractéristique, si $1/3 < l < 2/3$, de moyenne caractéristique, et de grande si $2/3 < l$.

4.2 – Corps de nombres ou corps de fonctions ?

Dans le diagramme commutatif, les groupes G_0, G_1 et G_2 donnent une représentation implicite du corps fini $G = \mathbb{F}_{p^n}$. G_1 et G_2 ne peuvent donc pas être tous les deux très petits. Cette restriction limite

6. Si g n'est pas dans la base, il est facile de l'ajouter en considérant des relations un peu plus générales $u^a g^b = \prod g_i$.

le choix de la base de lissité et la forme des relations multiplicatives créées, ce qui induit enfin une barrière de complexité naturelle en $L_q(1/3)$. Mais regardons de plus près qui sont ces groupes.

Moyenne et grande caractéristique : crible par corps de nombres. Les différentes variantes de cribles par corps de nombres se basent sur un diagramme qui part de l'anneau de polynômes à coefficients entiers $G_0 = \mathbb{Z}[X]$ et passe par deux corps de nombres $G_1 = \mathbb{Q}(X)/(f_1(X))$ et $G_2 = \mathbb{Q}(X)/(f_2(X))$. Les deux polynômes f_1 et f_2 sont choisis pour avoir une racine commune dans $\mathbb{F}_{p^n}$. La connaissance de G_1 et G_2 permet donc de retrouver G, c'est la représentation implicite que nous évoquions précédemment.

À l'intérieur de chaque corps de nombres, les petits éléments qui nous servent à construire la base de lissité sont les idéaux dont la norme est plus petite qu'une certaine borne de lissité. Le choix d'utiliser des idéaux vient du besoin d'obtenir une factorisation unique de chaque côté d'une relation. La manière de descendre explicitement ces idéaux vers le corps fini dans le diagramme est complexe, et nous en passerons les détails sous silence.

En compliquant encore, on peut utiliser un diagramme à plusieurs branches et tirer parti de l'utilisation de corps de nombres multiples. Cela a permis quelques avancées présentées cette année. En moyenne caractéristique, le crible par corps de nombres multiples [14] atteint une complexité asymptotique de $L_{p^n}(1/3, (8(9+4\sqrt{6})/15)^{1/3})$ tandis qu'en grande caractéristique le crible présenté dans [2] est de complexité $L_{p^n}(1/3, (2(46 + 13\sqrt{13})/27)^{1/3})$, cette seconde constante étant plus faible. De telles expressions peuvent sembler un peu barbares. Le second résultat est pourtant particulièrement intéressant puisque l'on retrouve exactement la complexité la plus basse connue pour factoriser un entier de même taille ! Et c'est plutôt heureux, car le crible par corps de nombres multiples est l'adaptation d'un algorithme de factorisation similaire.

Petite caractéristique : crible par corps de fonctions. En petite caractéristique, les groupes utilisés ont une structure plus simple puisqu'il s'agit de corps de fonctions. Depuis 2006, la simplification est telle que l'on ne considère plus que de simples anneaux de polynômes, bien que l'on conserve, par tradition, le terme de *crible par corps de fonctions*. Pour cette dernière construction, on a $G_0 = \mathbb{F}_p[X, Y]$, $G_1 = \mathbb{F}_p[X]$ et $G_2 = \mathbb{F}_p[Y]$, liés par des relations $Y = f_1(X)$ et $X = f_2(Y)$ permettant de définir les fonctions φ et ψ (par exemple $\psi_1(P(X, Y)) = P(X, f_1(X))$). La contrainte d'avoir une racine commune dans $\mathbb{F}_{p^n}$ s'exprime simplement en exigeant que $f_2(f_1(X)) - X$ ait un facteur irréductible de degré n.

Au lieu de chercher à factoriser des entiers (les normes des idéaux dans les corps de nombres), on travaille cette fois sur des polynômes à coefficients dans un petit sous-corps. Toutefois, la mécanique générale reste la même.

Cependant, la structure des polynômes sur des corps finis est bien mieux comprise ! Dans ce contexte, nous allons voir que l'un des ingrédients majeurs pour briser la barrière en $L_q(1/3)$ consiste à invalider l'heuristique usuelle en exhibant des polynômes de grands degrés , i.e., des éléments plus si petits que cela, qui ont la propriété agréable de se factoriser systématiquement en termes de petits degrés.

5. Du fracas en petite caractéristique

En 2013, plusieurs découvertes majeures ont profondément modifié la difficulté du problème du logarithme discret en petite caractéristique. Deux articles publiés en début d'année [10, 7] abaissent tout d'abord la deuxième constante de la complexité en $L_q(1/3, \cdot)$. La véritable rupture est la découverte [9] d'un algorithme en $L_q(1/4 + o(1))$. Ces trois articles travaillent essentiellement sur la façon de représenter le corps fini et de construire les relations multiplicatives. Quelques mois plus tard, une modification de la phase de descente du troisième algorithme conduit à un algorithme heuristique de complexité asymptotique [7] quasi-polynomiale [3].

Ces améliorations théoriques ont été couplées à de nouveaux et surprenants records de calculs : à l'heure actuelle, la cardinalité du plus gros corps fini dans lequel le problème du logarithme discret a été résolu s'écrit sur 9234 bits. Même s'il s'agit d'un corps un peu particulier, sa taille est dix fois supérieure au dernier record de 923 bits établi en 2012 avant ces avancées surprenantes.

7. Notons cependant que cette version de la descente n'est pas compétitive par rapport à la méthode en $L(1/4)$ pour les tailles actuelles des records de calcul.

5.1 – Représentation Frobeniale

Nous détaillons ici une version simplifiée de ces nouvelles méthodes, qui font appel à ce que l'on nomme la Représentation Frobeniale[8] (en opposition aux méthodes de crible). Cette version permet d'améliorer la complexité des deux premières phases. Elle reprend des travaux que nous avons menés récemment dans [11].

Les algorithmes par Représentation Frobeniale commencent par construire le corps fini $\mathbb{F}_{q^k}$ (où q n'est pas nécessairement premier) à l'aide d'un polynôme irréductible I de degré k tel que :

$$I(X) \quad \text{divise} \quad h_1(X)X^q - h_0(X)$$

où h_0 et h_1 sont deux polynômes de petits degrés. Si θ est une racine de I, on a ainsi choisi la représentation $\mathbb{F}_{q^k} = \mathbb{F}[\theta]$ de notre corps fini. En fait, comme il est facile de calculer explicitement les isomorphismes entre les différentes représentations de $\mathbb{F}_{q^k}$, ce choix n'est nullement une contrainte. De plus, il nous permet d'obtenir la relation $\theta^q = h_0(\theta)/h_1(\theta)$ dans le corps fini que l'on considère (d'où la notion de représentation par action de Frobenius). L'égalité absolument essentielle par la suite est l'identité polynomiale bien connue sur $\mathbb{F}_q[X]$:

$$\prod_{\alpha\in\mathbb{F}_q}(X-\alpha) = X^q - X.$$

En remplaçant X par $A(\theta)/B(\theta)$ pour A et B deux polynômes de degré au plus D et en multipliant par $B(\theta)^{q+1}$, on trouve dans le corps fini :

$$\begin{aligned} B(\theta)\prod_{\alpha\in\mathbb{F}_q}(A(\theta)-\alpha B(\theta)) &= A(\theta)^q B(\theta) - A(\theta)B(\theta)^q \\ &= A(\theta^q)B(\theta) - A(\theta)B(\theta^q) \end{aligned}$$

car $A(X)^q = A(X^q)$ par linéarité du Frobenius. On obtient finalement :

$$B(\theta)\prod_{\alpha\in\mathbb{F}_q}(A(\theta)-\alpha B(\theta)) = \frac{[A,B]_D(\theta)}{h_1(\theta)^D}, \qquad (2)$$

où $[A,B]_D$ désigne le polynôme de petit degré suivant :

$$h_1(\theta)^D\left(A\left(\frac{h_0(\theta)}{h_1(\theta)}\right)B(\theta) - A(\theta)B\left(\frac{h_0(\theta)}{h_1(\theta)}\right)\right).$$

Puisque les polynômes $A(\theta)-\alpha B(\theta)$ sont de degré au plus D, l'équation (2) donne une relation multiplicative entre polynômes en θ de degré au plus D pour une fraction constante des paires de polynômes (A,B). Comparativement aux algorithmes des générations précédentes, nous obtenons une relation dont nous ne testons la lissité que d'un seul côté (le côté gauche étant systématiquement lisse).

En utilisant ce principe avec des polynômes h_0 et h_1 bien choisis, il est possible de retrouver les logarithmes de tous les polynômes de degrés 2 en temps $O(q^5)$. Ceux-ci forment alors notre base de lissité initiale. Toutefois, cette base est trop petite pour directement permettre de s'intéresser à la phase de logarithmes individuels. Il est donc nécessaire de l'étendre pour lui adjoindre des polynômes de degrés plus élevés.

Pour cette extension, en regroupant astucieusement les éléments de la base étendue par paquets de polynômes *cousins* – par exemple, deux polynômes qui partagent le même coefficient constant sont cousins – on parvient à retrouver les logarithmes des polynômes de degrés 3 puis 4 en $O(q^6)$. Notons qu'il est bien plus efficace de procéder ainsi par extensions successives que de directement écrire des équations sur une base incluant tous les polynômes de degré 3, voire 4. En effet, ces approches directes amènent à des complexités bien moins bonnes en $O(q^7)$ ou même $O(q^9)$. Malgré tout, même avec ces complexités dégradées, le calcul initial des logarithmes des éléments de la base de lissité est polynomial en le logarithme de la cardinalité du corps considéré, ce qui est un énorme progrès par rapport aux algorithmes en $L(1/3)$ des générations précédentes.

5.2 – Logarithme individuel

Avec ce progrès sur la création des logarithmes de la base de lissité, on pourrait s'attendre à ce que l'ensemble du calcul de logarithme discret devienne polynomial. Mais ce n'est pas le cas, la phase finale permettant de calculer des logarithmes individuels, dont la contribution au coût du calcul était négligeable avec les anciennes méthodes, domine maintenant le coût asymptotique du calcul.

Sans se noyer dans les détails techniques, essayons de donner le principe général de la méthode. Pour trouver le logarithme d'un élément quelconque du corps fini $\mathbb{F}_{q^k}$, représenté par un polynôme $z(\theta)$ de degré au plus $k-1$, on cherche à

8. Il s'agit toujours de calcul d'indice, mais la première phase est modifiée en profondeur.

l'écrire comme produit (ou quotient) de polynômes de la base de lissité étendue (i.e. les polynômes de degré au plus 4). Comme on ne sait pas le faire directement, on procède par récurrence en cherchant à exprimer z comme produit (ou quotient) de polynômes de degré moitié au plus. Toutes ces expressions emboîtées peuvent se représenter par un arbre ayant z à sa racine et des polynômes de degré au plus 4 comme feuille. Dans cet arbre, chaque nœud a pour fils les polynômes de degrés plus petits qui servent à l'exprimer. Le nombre de nœuds de cet arbre correspond à la complexité du calcul du logarithme de z. Le degré étant divisé par 2 à chaque étage, la hauteur de l'arbre est une fonction logarithmique du degré de z, donc de k. Si le nombre de fils de chaque noeud pouvait être borné par une constante, la complexité serait polynomiale, mais nous allons voir que ce n'est pas le cas. En effet, pour créer des relations faisant intervenir z et capables de descendre jusqu'à la base de lissité, les méthodes connues nécessitent de réutiliser l'équation (2), cette fois avec des polynômes A et B de degrés plus élevés. Or, les équations de ce type contiennent au moins q termes, ce qui explique que la meilleure complexité atteignable par ce type de méthodes est de la forme $q^{O(\ln k)}$. Cette complexité est quasi-polynomiale dans la taille du corps $\ln q^k = k \ln q \approx q$ (puisque $k \approx q$).

En pratique, la situation est un peu plus complexe car on peut choisir de faire apparaître z soit à gauche, soit à droite dans l'équation (2). La première méthode découverte place z à droite et s'appuie sur la résolution d'équations bilinéaires dans les coefficients de A et de B pour construire ces polynômes, elle permet d'obtenir une complexité en $L(1/4)$. La seconde méthode consiste à placer z à gauche. D'un point de vue théorique, elle a permis d'atteindre une complexité quasi-polynomiale, toutefois, elle nécessite de calculer simultanément le logarithme de z et de q^2 polynômes cousins, ce qui la rend totalement inutilisable en pratique. Une troisième méthode [8] met de nouveau z à droite dans l'équation mais procède différemment pour reconstruire A et B, cela permet d'obtenir différemment une complexité quasi-polynomiale, tout en étant utilisable dans certains calculs pratiques.

5.3 – Conclusion

Malgré ces avancées récentes, le problème du logarithme discret reste d'une grande actualité. En petite caractéristique, l'espoir est de réussir à s'affranchir des diverses hypothèses heuristiques qui sous-tendent les méthodes actuelles. En pratique, ces résultats ont forcé l'abandon des algorithmes cryptographiques s'appuyant sur les corps finis de petite caractéristique, comme en témoignent les nouvelles recommandations de l'ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) parues en 2013. Heureusement, la petite caractéristique étant assez peu utilisée pour des raisons historiques, cela n'a pas conduit à de trop lourdes conséquences.

Références

[1] L. M. Adleman. « A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography (Abstract) ». In : *FOCS*. 1979, p. 55–60.

[2] R. Barbulescu et C. Pierrot. « The multiple number field sieve for medium- and high-characteristic finite fields ». *LMS Journal of Computation and Mathematics* **17**, n° A (2014), p. 230–246. ISSN : 1461-1570.

[3] R. Barbulescu et al. « A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic ». In : *EUROCRYPT*. 2014, p. 1–16.

[4] E. Canfield, P. Erdös et C. Pomerance. « On a problem of Oppenheim concerning factorisatio numerorum ». In : *Journal of Number Theory*. Vol. 17. 1983, p. 1–28.

[5] W. Diffie et M. E. Hellman. « New directions in cryptography ». *IEEE Transactions on Information Theory* **22**, n° 6 (1976), p. 644–654.

[6] T. E. Gamal. « A public key cryptosystem and a signature scheme based on discrete logarithms ». *IEEE Transactions on Information Theory* **31**, n° 4 (1985), p. 469–472.

[7] F. Göloglu et al. « On the Function Field Sieve and the Impact of Higher Splitting Probabilities - Application to Discrete Logarithms in $\mathbb{F}_{2^{1971}}$ and $\mathbb{F}_{2^{3164}}$ ». In : *CRYPTO (2)*. 2013, p. 109–128.

[8] R. Granger, T. Kleinjung et J. Zumbrägel. « On the Powers of 2 ». *IACR Cryptology ePrint Archive* **300** (2014).

[9] A. Joux. « A New Index Calculus Algorithm with Complexity $L(1/4 + o(1))$ in Small Characteristic ». In : *Selected Areas in Cryptography*. 2013, p. 355–379.

[10] A. Joux. « Faster Index Calculus for the Medium Prime Case Application to 1175-bit and 1425-bit Finite Fields ». In : *EUROCRYPT*. 2013, p. 177–193.

[11] A. Joux et C. Pierrot. « Improving the Polynomial time Precomputation of Frobenius Representation Discrete Logarithm Algorithms. A Simplified Setting for Small Characteristic Finite Fields ». In : *ASIACRYPT*. 2014.

[12] M. Kraïtchik. *Théorie des nombres*. Gauthier–Villars, 1922.

[13] D. Panario, X. Gourdon et P. Flajolet. « An Analytic Approach to Smooth Polynomials over Finite Fields ». In : *ANTS*. 1998, p. 226–236.

[14] C. Pierrot. « The Multiple Number Field Sieve with Conjugation and Generalized Joux-Lercier Methods ». *IACR Cryptology ePrint Archive* (2014).

[15] S. C. Pohlig et M. E. Hellman. « An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (Corresp.) » *IEEE Transactions on Information Theory* **24**, n° 1 (1978), p. 106–110.

[16] J. Pollard. « Monte Carlo methods for index computations mod *p* ». In : *Mathematics of Computation*. Vol. 32. 143. 1978, p. 918–924.

[17] C. Pomerance. « Fast, rigorous factorization and discrete logarithm algorithms ». In : *Discrete algorithms and complexity*. Academic Press, 1987, p. 119–143.

[18] R. L. Rivest, A. Shamir et L. Adleman. « A Method for Obtaining Digital Signatures and Public-Key Cryptosystems ». *Communications of the ACM* **21** (1978), p. 120–126.

[19] V. Shoup. « Lower Bounds for Discrete Logarithms and Related Problems ». In : *EUROCRYPT*. 1997, p. 256–266.

Antoine Joux

Chaire de cryptologie de la Fondation partenariale de l'UPMC, 4 place Jussieu, 75005 Paris, France et CryptoExperts, Paris, France

`Antoine.Joux@m4x.org`

Antoine Joux occupe la chaire de cryptologie de la fondation partenariale de l'UPMC (Paris 6). Il est également expert en sécurité chez CryptoExperts. Il a reçu le prix Gödel en 2013 pour l'introduction de la cryptographie reposant sur les couplages et se spécialise dans l'étude des problèmes algorithmiques utiles en cryptologie.

Cécile Pierrot

Laboratoire d'Informatique de Paris 6, UMPC, 4 place Jussieu, Paris, France et Direction Générale de l'Armement, Ministère de la Défense

`Cecile.Pierrot@lip6.fr`

Cécile Pierrot est en deuxième année de thèse à l'université Pierre et Marie Curie (Paris 6) sous la direction d'Antoine Joux. Son sujet d'étude est le problème du logarithme discret sur les corps finis et elle a publié plusieurs améliorations notables des algorithmes précédemment connus.