Factorisation et logarithme discret : Méthodes exponentielles

Dans la suite on va étudier des méthodes pour les problèmes difficiles en cryptographie qui sont exponentielles, mais avec une exposant mieux que les algorithmes naïfs.

Pas de bébé, pas de géant

L'algorithme pas de bébé, pas de géant permet de calculer un logarithme discret dans un groupe G cyclique (avec générateur α) d'ordre N avec $\sqrt{N}$ opérations dans le groupe. Ici on fixe $G = \mathbb{F}_p^*$ et $N = p-1$, mais le principe est pareil pour tout groupe cyclique. On rappelle que le logarithme discret par rapport à α est l'inverse du homomorphisme

$$\mathbb{Z}/N\mathbb{Z} \longrightarrow G = \langle \alpha \rangle$$

donné par $k \mapsto \alpha^k$. En particulier, en donnant α et β on cherche k tel que $\beta = \alpha^k$.

Algorithme. Pour un constant $c > 0$, on pose $m = \left\lfloor c\sqrt{N} \right\rfloor$ et on calcule la suite :

$$1, \alpha, \alpha^2, \ldots, \alpha^{m-1} \in G,$$

en gardant la bijection avec la suite $0, 1, 2, \ldots, m-1 \in \mathbb{Z}/N\mathbb{Z}$. L'ensemble

$$\mathcal{B} = \{1, \alpha, \ldots, \alpha^{m-1}\}$$

s'appelle *les pas de bébé*.

Ensuite on contruit *les pas de géant*, en partant de β :

$$\beta, \alpha^m\beta, \alpha^{2m}\beta, \ldots$$

jusqu'on trouve une collision $\ldots \alpha^{jm}\beta = \alpha^i \in \mathcal{B}$, avec $0 \leq i \leq m$, d'où $\beta = \alpha^{i-jm}$, et donc $k = i - jm \in \mathbb{Z}/N\mathbb{Z}$ est le logarithme discret $\log_\alpha(\beta)$.

Complexité. On note qu'on a obligatoirement $j \leq \lfloor N/m \rfloor$, car l'ensemble

$$\begin{aligned}\bigcup_{j=0}^{\lfloor N/m \rfloor} (\{0, 1, 2, \ldots, m-1\} - jm) &= \{i - jm : 0 \leq i \leq m-1, 0 \leq j \leq \lfloor N/m \rfloor\} \\ &= [\![-\lfloor N/m \rfloor m, m-1]\!]\end{aligned}$$

est un intervalle de longueur $(\lfloor N/m \rfloor + 1)m \geq N$, recouvrant toutes les classes de $\mathbb{Z}/N\mathbb{Z}$. Par conséquent, l'algorithme utlise $m + \lfloor N/m \rfloor \in O(\sqrt{N})$ opérations dans le groupe G.

Remarque. L'algorithme *pas de bébé, pas de géant* donne une réduction de $O(N)$ à $O(\sqrt{N})$ opérations par rapport à un algorithme naïf. On note que l'espace en mémoire est aussi $O(\sqrt{N})$ et le constant c permet de diminuer (par un constant) le coût en mémoire en échange pour un facteur constant de plus dans le temps. Dans la suite on décrit un algorithme, aussi avec complexité $O(\sqrt{N})$, qui permet de réduire à $O(1)$ le coût en mémoire.

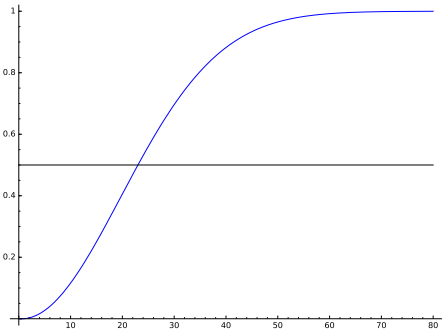
Pollard rho

Les algorithmes de Pollard rho (d'après la forme de la lettre grecque ρ) est basé sur l'algorithme de Floyd, qui se base sur le paradoxe des anniversaires. Après une description de l'algorithme de Floyd, on va introduire des algorithmes (de Pollard) pour les problèmes du log discret et de la factorisation. On rappelle d'abord ce paradoxe.

Paradoxe des anniversaires

Le paradoxe des anniversaires n'est pas un paradoxe dans le sens d'être une contradiction logique, mais il est dans le sens original du mot grec — contraire à l'opinion commune, allant contre l'intuition. Il s'agit de la réponse à la question suivante.

Coïncidences des anniversaires. Dans une salle de classe de k éléves, quelle est la probabilité que deux personnes partagent le jour de leur anniversaire ?



N.B. Pour modéliser ce problème, on suppose que la contribution des personnes nées le 29 février est néglible, que l'année est de $N = 365$ jours, et que les anniversaires sont équidistribuées dans l'année.

Plus généralement, on suppose qu'on choisit k fois un élément d'un ensemble de N éléments. On note par $q(N, k)$ la probabilité que ces k éléments soient distincts, donc

$$q(N,k) = \prod_{i=0}^{k-1} \frac{(N-i)}{N} = \prod_{i=0}^{k-1} \left(1 - \frac{i}{N}\right),$$

Alors la valeur $p(N, k) = 1 - q(N, k)$ est la probabilité d'une collision (que les choix d'éléments ne soient pas tous distincts). En utilisant l'approximation dans un voisinage de 0,

$$\exp(x) \in 1 + x + o(x^2),$$

on obtient, pour $x = -i/N$,

$$q(N,k) \sim \prod_{i=0}^{k-1} \exp(-\frac{i}{N}) = \exp\left(-\frac{k(k-1)}{2N}\right) \sim \exp\left(-\frac{k^2}{2N}\right).$$

Pour $k = \sqrt{2N}$ cette approximation descend à $e^{-1} < 1/2$, et dans un intervalle logarithmique autour de $\sqrt{2N}$, on trouve :

k	$q(N,k)$
$\sqrt{N}$	$\sqrt{e^{-1}} = 0.606530\ldots$
$\sqrt{2\log(2)N}$	$e^{-\log(2)} = 0.500000\ldots$
$\sqrt{2N}$	$e^{-1} = 0.367879\ldots$
$\sqrt{4\log(2)N}$	$e^{-\log(4)} = 0.250000\ldots$
$2\sqrt{N}$	$e^{-2} = 0.135335\ldots$

La *fonction de density* pour $1 - \exp(-k^2/2N) \sim p(N,k)$ est

$$\rho(N,k) = \frac{d}{dk}\left(1 - \exp\left(\frac{-k^2}{2N}\right)\right) = \frac{k}{N}\exp\left(-\frac{k^2}{2N}\right)$$

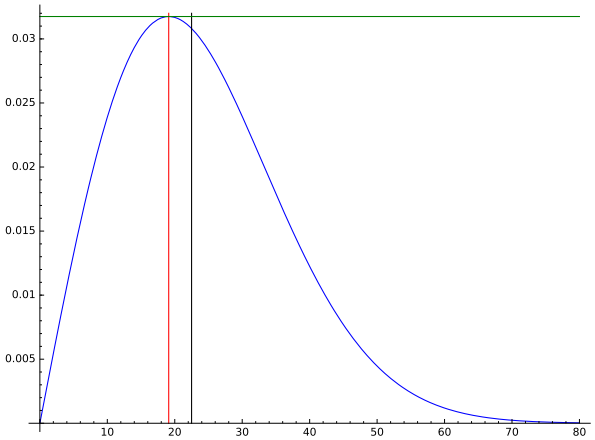
avec density maximum $\rho(N,k) = 1/\sqrt{eN}$ à $k = \sqrt{N}$. En retournant aux coïncidences des anniversaires, pour $N = 365$, on trouve que la valeur de density maximale est

$$k = \sqrt{N} = 19.104973\ldots$$

et la valueur donnant $p(N,k) \sim 1 - \exp(-k^2/2N) = 1/2$ est :

$$k = \sqrt{2\log(2)N} = 22.49438\ldots$$

Par conséquent, quand le nombre de personnes s'élève à 23, la probabilité d'une coïncidence des anniversaires dépasse 50%. Dans le graphe ci-dessous de la fonction $\rho(N,k)$, avec $N = 365$, on voit en rouge la droite $k = \sqrt{N}$ pour la valeur maximum de $\rho(N,k)$ et en noire la droite $k = \sqrt{2\log(2)N}$ qui divise l'aire sous la courbe en deux parties égales, avec aire $1/2$.



Algorithme de Floyd

On va étudier l'algorithme du lièvre et de la tortue pour détection d'un cycle dans une suite récurrente (u_n) avec $u_n \in S$, engendrée par une fonction $f : S \to S$, tel que $u_{n+1} = f(u_n)$.

Définition. Une suite (u_n) est *récurrente* si le terme u_n est déterminé par un nombre r fixé de termes précédents : il existe une fonction $f : S^r \to S$ tel que

$$u_n = f(u_{n-r}, \ldots, u_{n-1}).$$

On dit que r est *l'ordre* de la récurrence. Si $r = 1$, on dit que la récurrence est *autonome*. Si S est un anneau, la récurrence est *linéaire*, si

$$u_n = a_r u_{n-r} + \cdots + a_1 u_{n-1},$$

pour $a_1, \ldots, a_r$ dans S. On s'interesse ici aux suites récurrentes non linéaires autonomes, dans un ensemble fini S.

Périodicité

Définition. Une suite (u_n) est *ultimement périodique* s'il existe des entiers $m > 0$ et $\ell \geq 0$ tels que

$$a_{n+m} = a_n \text{ pour tout } n \geq \ell.$$

Si m et ℓ sont minimaux avec cette propriété, on dit que m est sa *période* et ℓ sa *queue*.

Si (u_n) est une suite récurrente autonome qui est ultimement périodique, les valeurs $u_0, u_1, \ldots, u_\ell, \ldots, u_{\ell+m-1}$ sont distincts.

Remarque. Si S est fini de cardinal N et la suite (u_n) est récurrente, alors la suite est ultimement périodique. Comme il n'y a qu'un nombre fini ($= N^r$) de choix possibles pour les tuples

$$(u_{n-r}, \ldots, u_{n-1}),$$

qui déterminent le prochaine élément de la suite. Si les r-tuples $(u_{n-r}, \ldots, u_{n-1})$ d'éléments successifs de (u_n) se comportent comme des éléments aléatoires, alors par l'analyse du paradoxe des anniversaires, l'espérance mathématiques de la longueur de $\ell + m$ (somme de la période et la queue) est

$$\ell + m \in O\big(\sqrt{N^r}\big).$$

On s'intéresse aux suites récurrentes autonomes, pour lesquelles l'espérence mathématiques est $\ell + m \in O(\sqrt{N})$ par le paradoxe des anniversaires. On observe alors que S se décompose en sous-ensembles disjoints U tel que $f(U) \subseteq U$.

Algorithme de la tortue et du lièvre de Floyd

L'astuce derrière l'algorithme de Floyd est le suivante : si une suite est ultimement périodique, il existe un indice n tel que $u_{2n} = u_n$, avec $\ell \leq n \leq \ell + m$. L'algorithme de Floyd consiste à parcourir la suite simultanément à deux vitesses :

$$\begin{aligned} \text{(la tortue)} &: \quad u_0, u_1, u_2, u_3, \ldots \\ \text{(le lièvre)} &: \quad u_0, u_2, u_4, u_6, \ldots \end{aligned}$$

en construisant la suite des couples : $(u_0, u_0), (u_1, u_2), (u_2, u_4), \cdots = ((u_n, u_{2n}))$. En identifiant une collision $u_{2n} = u_n$, on a $2n - n = n \equiv 0 \bmod m$.

Pollard rho : Algorithme pour le log discret

On va construire une version de l'algorithme de Floyd adaptée au problème du logorithme discret. On suppose donné un groupe G, des élément $a, b \in G = \langle a \rangle$, et $N = |G|$. L'algorithme va déterminer k tel que $b = a^k$.

On partitione G en trois sous-ensembles disjoints S_0, S_1, S_2, de tailles égales, et on définit une fonction $f : G \to G$ par

$$f(x) = \begin{cases} bx & x \in S_0 \\ x^2 & x \in S_1 \\ ax & x \in S_2 \end{cases}$$

Remarque. Pour les logarithmes discrets dans $G = \mathbb{F}_p^*$, on peut utiliser $a \in S_j$ si et seulement si $a \bmod 3 = j$ (pour la réprésentant a dans $\{1, \ldots, p-1\}$.

Ensuite on définit une fonction $g : G \times \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$, compteur des multiplications par a :

$$g(x,k) = \begin{cases} k & x \in S_0 \\ 2k & x \in S_1 \\ k+1 & x \in S_2 \end{cases}$$

et une fonction $h : G \times \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$, compteur des multiplications par b :

$$h(x,\ell) = \begin{cases} \ell+1 & x \in S_0 \\ 2\ell & x \in S_1 \\ \ell & x \in S_2 \end{cases}$$

Finalement, on définit une application

$$F : G \times \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} \longrightarrow G \times \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$$

par

$$F((u,k,\ell)) = (f(u), g(u,k), h(u,\ell)).$$

Si (u_i, k_i, ℓ_i) est un triple satisfaisant $u_i = a^{k_i} b^{\ell_i}$, alors par la définition des compteurs g et h, le nouveau triple

$$(u_{i+1}, k_{i+1}, \ell_{i+1}) = F((u_i, k_i, \ell_i))$$

satisfait $u_{i+1} = a^{k_{i+1}} b^{\ell_{i+1}}$.

La tortue et le lièvre. L'algorithme procède par la construction de deux suites de triples :

$$\begin{array}{rl} \text{(la tortue)} : & (u_i, k_i, \ell_i), \text{ et} \\ \text{(le lièvre)} : & (u_{2i}, k_{2i}, \ell_{2i}) \end{array}$$

par l'initialisation $(u_0, k_0, \ell_0) = (e, 0, 0)$ et la récurrence pour $i = 0, 1, 2, \ldots$

$$\begin{gathered} (u_{i+1}, k_{i+1}, \ell_{i+1}) = F((u_i, k_i, \ell_i)), \\ (u_{2i+2}, k_{2i+2}, \ell_{2i+2}) = F(F((u_{2i}, k_{2i}, \ell_{2i}))). \end{gathered}$$

Terminaison. Si $u_{2n} = u_n$, on sort de la récurrence en posant $r = \ell_n - \ell_{2n}$. Si r et N sont premiers entre eux ($r \neq 0$ si N est premier), on retourne `ECHEC`, sinon on retourne

$$k = r^{-1}(k_{2n} - k_n).$$

Preuve. Pour démontrer que la sortie vaut $k = \log_a(b)$, on note que $u_0 = a^0 b^0 = e$ et ensuite on vérifie (comme observé précédemment) que $u_i = a^{k_i} b^{\ell_i}$. La coïncidence $u_{2n} = u_n$ donne lieu à l'égalité

$$a^{k_{2n}} b^{\ell_{2n}} = a^{k_n} b^{\ell_n},$$

et il suit que

$$b^r = b^{(\ell_{2n} - \ell_n)} = a^{(k_n - k_{2n})}.$$

Si $s = r^{-1}$ dans $\mathbb{Z}/N\mathbb{Z}$, on a alors

$$b = b^{sr} = b^{s(\ell_{2n} - \ell_n)} = a^{s(k_n - k_{2n})} = a^k$$

et donc $k = \log_a(b)$.

Remarque. La probabilité de retourner `ECHEC` est faible — par Pohlig-Hellman, on peut supposer que N n'a pas de diviseur premier petit, et sinon, $\text{pgcd}(r, n) \neq 1$ donne des informations concernant la factorisation de N.

Pollard rho : Algorithme pour la factorisation des entiers

On décrit un autre algorithme de Pollard sur la même principe, pour la factorisation des entiers. On suppose donnée un entier impair composé N. Soit p (inconnu) un diviseur ; on peut supposer que p est la plus petit diviseur premier. On a donc $N = pm$.

Soit $f : \mathbb{Z}/N\mathbb{Z} \longrightarrow \mathbb{Z}/N\mathbb{Z}$ une fonction polynomiale, par exemple $f(x) = x^2 + 1$.

Initialisation :
— $(a_0, a_0) = (2, 2)$ et $d = 1$.
Itération : pour $i = 1, 2, \ldots$, on calcule :
— $(a_i, a_{2i}) = (f(a_{i-1}), f(f(a_{2i-2})))$,
— $d = \text{pgcd}(a_{2i} - a_i, N)$.
Si $d \neq 1$ on retourne le diviseur d de N.

Remarque. Si $d = N$, le résultat est `ECHEC`.

Idée. On ne connaît pas p, mais on peut détecter la congruence $a_{2i} \equiv a_i \bmod p$ par le $\text{pgcd}(a_{2i} - a_i, N)$. Si la suite (a_i) se comporte comme une suite aléatoire, la première fois qu'on rencontre une telle congruence est après $O(\sqrt{p})$ étapes.

Supposant que $q > p$ est un autre diviseur premier de n. Par l'hypothèse que la suite (a_i) se comporte de manière aléatoire, par le théorème chinois, les congruences modulo p et modulo q sont indépendentes, donc la probabilité que la premier indice i tel que $a_{2i} \equiv a_i \bmod p = 0$ satisfait aussi

$$a_{2i} \equiv a_i \bmod q = 0,$$

est propositionnelle à $1/\sqrt{q}$. Alors la probabilité d'échec est très faible.

Exercice. Sous l'hypothèse que la suite (a_i) est aléatoire dans $\mathbb{Z}/N\mathbb{Z}$, montrer que la probabilité d'échec est proporsionelle à $1/\sqrt{m}$ où $N = pm$. Montrer que la suite n'est pas aléatoire dans $\mathbb{Z}/N\mathbb{Z}$ si $N \equiv 0 \bmod p^2$. Dans ce cas, quelles sont les conséquences pour la factorisation de N par l'algorithme rho de Pollard ?

Exemple. On suppose que $N = 194291$. On obtient la suite suivante :

i	a_i	a_{2i}	d
0	2	2	1
1	5	26	1
2	26	69748	1
3	677	155974	97

Alors on trouve la factorisation de $N = 97 \cdot 2003$ après trois itérations.