Cycle Préparatoire de Bordeaux,
Université de Bordeaux

— Projet final —

Le problème du logarithme discret

et ses applications en cryptographie à clé publique

Eloïse Brocas Alix Kien Adrien Maurin

Encadré par Aurel Page

2017–2018

Résumé

La cryptographie à clé publique repose sur l'usage de fonctions à sens unique. Il est aisé de calculer les images de ces fonctions mais il est très difficile de retrouver l'antécédent à partir d'une image. Ce projet vise à étudier le problème du logarithme discret défini dans des groupes cycliques. Nous cherchons à déterminer sous quelles conditions il peut être utilisé dans des protocoles de chiffrement à clé publique. Pour vérifier la difficulté d'inversion de la fonction, nous étudions l'efficacité des algorithmes de résolution Baby-Step Giant-Step et Rho de Pollard. Leurs complexités en temps et en mémoire sont déterminées respectivement dans le pire des cas et en moyenne. Après les avoir implémentés pour des groupes $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$ avec le langage Python3, nous vérifions expérimentalement ces complexités théoriques. Dans un second temps, nous présentons deux protocoles de cryptographie asymétrique basés sur la difficulté de résolution de problèmes du logarithme discret : le protocole d'échange de clés Diffie-Hellman et le cryptosystème de ElGamal. Nous obtenons que la complexité en temps de l'algorithme de résolution Baby-Step Giant-Step est en $\mathcal{O}(\sqrt{n}\log n)$ et en $\mathcal{O}(\sqrt{n})$ pour Rho de Pollard contre une complexité en $\mathcal{O}(n)$ pour l'algorithme exhaustif. Les tests des différentes implémentations nous permettent de donner des conditions d'utilisation du problème en cryptographie. Si les seules attaques considérées sont nos implémentations des algorithmes étudiés, les protocoles seront considérés sécurisés pour des groupes d'ordre supérieur à 10^{30}. Cependant, ce projet ne traite pas le cas des meilleurs algorithmes de résolution du problème de complexité sous-exponentielle. Par conséquent, les paramètres déterminés pour l'implémentation des deux cryptosystèmes sont inférieurs aux normes standards de sécurité.

Mots-clés : logarithme discret, cryptographie à clé publique, Baby-Step Giant Step, Rho de Pollard, protocole d'échange de clés Diffie-Hellman, cryptosystème de ElGamal.

Abstract

Public-key cryptography is based on the use of one-way functions. It is easy to compute images of these functions but it is very hard to find the preimage from an image. This project aims to study the discrete logarithm problem defined in cyclic groups to determine under which conditions it can be used in public-key cryptography protocols. To verify the function inversion difficulty, we study the resolution algorithms efficiency : Baby-Step Giant-Step and Pollard's Rho. We determine their complexities in time and in memory respectively in the worst case and in average. After having implemented them for groups $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$ with the language Python3, we experimentally verify these theoretical complexities. In the second part, we present two public-key cryptography protocols based on the difficulty to solve discrete logarithm problems : the Diffie-Hellman key agreement protocol and the ElGamal encryption system. We obtain that the complexity in time of the solving algorithm Baby-Step Giant-Step is in $\mathcal{O}(\sqrt{n}\log n)$ and in $\mathcal{O}(\sqrt{n})$ for Pollard's Rho, to be opposed to $\mathcal{O}(n)$ for the trial multiplication algorithm. Tests of the different implementations allow us to give usage restrictions of this problem in cryptography. If we only consider attacks with our implementations of the studied algorithms, then the protocols will be considered secure for groups with an order superior to 10^{30}. However this project does not consider the best solving algorithms of this problem, which are with a subexponential complexity. As a consequence, the determined parameters for both cryptosystems implementations are below security standards.

Keywords : discrete logarithm, public-key cryptography, Baby-Step Giant-Step, Pollard's Rho, Diffie-Hellman key exchange, ElGamal cryptosystem.

Remerciements

Ce projet n'aurait pas pu se dérouler ainsi sans notre tuteur Aurel Page qui a su nous guider avec pédagogie en répondant à nos nombreuses questions lors des longs rendez-vous que nous avons eu ; tout en nous poussant à chercher par nous même et à développer nos connaissances. Nous le remercions aussi pour ses relectures détaillées et exigentes mais toujours réalisées rapidement.

Nous tenons aussi à remercier Isabelle Escolin-Contensou pour nous avoir amené à prendre d'autres points de vue sur notre sujet afin d'en saisir tous les enjeux.

Enfin, nous remercions Alice Esnault pour sa relecture attentive de ce mémoire.

Avant-propos

Ce projet [1] est né de la volonté des trois membres du groupe à trouver un sujet mêlant à la fois mathématiques et informatique. Notre intérêt pour ces deux domaines reflète à la fois des questionnements et attirances nés durant nos études mais aussi la volonté que nous avons d'intégrer l'an prochain les filières Informatique ou Matmeca de l'ENSEIRB-MATMECA.

Nous utilisons des systèmes de chiffrement — ou protocoles cryptographiques — tous les jours quand nous allons sur Internet, nous payons par carte bancaire, nous utilisons des cartes sans contact, *etc.* Ainsi, le lien entre théorie et pratique est très important : de nombreuses avancées — création de la cryptographie à clé publique par exemple — ont directement eu des applications dans la vie courante. Nous avons voulu à travers ce projet explorer le côté théorique d'une des applications de la cryptographie.

Ce sujet nous a permis d'aborder et d'approfondir des domaines peu ou pas étudiés dans notre cursus comme l'arithmétique ou la sécurité informatique. Nous avons pu découvrir de nouveaux aspects des filières dans lesquelles nous voulons continuer notre formation ainsi que nourrir notre volonté d'approfondir nos cours et nos connaissances personnelles sur certains sujets.

La plupart de nos choix portant sur l'étude d'algorithmes ou d'objets mathématiques ont été réalisés au vu de notre niveau et du programme en informatique et mathématiques du Cycle Préparatoire de Bordeaux. C'est pourquoi nous ne pouvons dire que ce mémoire comporte de nouveautés scientifiques. Il s'agit principalement de l'approche d'un questionnement scientifique à l'aide d'outils dont la compréhension est possible par un étudiant de notre cursus (Bac +2).

La cryptographie, plus particulièrement la cryptographie à clé publique, est un domaine jeune (créé dans les années 1970) qui évolue vite. Les études que nous avons menées dans ce mémoire sont ici appliquées sur des structures mathématiques qui ne sont plus utilisées aujourd'hui. Cependant, les différents protocoles étaient originellement codés dans ces structures. Par conséquent, nous pouvons aussi voir une partie de notre projet comme une étude de protocoles de cryptographie à clé publique tels qu'ils ont été pensés et créés.

1. Il est a noter que l'organisation de ce projet est détaillée en annexe F.

Table des matières

Introduction

En 1976, deux chercheurs de l'université de Stanford, Whitfield Diffie et Martin Hellman, publient un article révolutionnaire « New directions in cryptography », qu'ils commencent par la phrase annonciatrice :

> « We stand today on the brink of a revolution in cryptography. »

Cette période fut cruciale dans la démocratisation de l'usage de la cryptographie. À la croisée d'avancées scientifiques et technologiques et d'enjeux politiques, militaires et commerciaux, les années soixante-dix furent le théâtre d'une bataille politique et médiatique [2] dont cette publication fut l'un des catalyseur. Cette bataille aboutit des deux côtés de l'Atlantique à la fin du monopole étatique sur l'usage de la cryptographie, créant ainsi de nouveaux enjeux pour la société civile. En effet, en plus d'une utilisation importante dans les secteurs commerciaux et bancaires, cette technologie amène la question du respect des libertés fondamentales au cœur d'une société numérique. Les citoyens peuvent accéder à de nouveaux moyens pour garantir leur vie privée redéfinissant ainsi l'équilibre entre les libertés individuelles et les intérêts étatiques [3].

Dans ce contexte, les deux chercheurs présentent dans leur article un nouveau type de cryptographie appelé cryptographie à clé publique. Jusqu'alors, les seuls protocoles de cryptographie connus étaient de type symétrique : il était nécessaire d'utiliser une même clé pour chiffrer et déchiffrer le message. À l'inverse, les cryptosystèmes — un protocole de cryptographie avec les différentes clés nécessaires — à clé publique reposent sur une double information propre à chaque utilisateur : une clé publique stockée dans des annuaires accessibles par tous et une clé privée qui reste secrète.

La cryptographie à clé publique est basée sur l'utilisation d'un type particulier de fonctions dites à *sens unique*. La fonction f est dite à sens unique s'il est facile de calculer la valeur $f(x)$ mais difficile de retrouver x en connaissant seulement $y = f(x)$. Notre projet vise à étudier une famille particulière de fonctions à sens unique : les logarithmes discrets. Le problème du logarithme discret peut être défini dans un premier temps comme la recherche de x tel que $h = g^x$ avec h et g connus.

Il est aisé de remarquer que le calcul de g^x est facilement réalisable. Nous allons donc nous concentrer sur la vérification de la difficulté de résolution du problème. Pour cela, nous allons étudier différents algorithmes de résolution. Nous utiliserons des outils qui nous permettront d'estimer le temps nécessaire à l'exécution de l'algorithme et donc le temps nécessaire à l'obtention d'une solution juste.

La taille d'un nombre sera ici entendue comme le nombre de chiffres composant un nombre. Notre but est de déterminer à partir de quelle taille de nombre le problème du logarithme discret sera difficile à résoudre avec nos algorithmes. Nous étudierons ensuite deux protocoles de cryptographie basés sur les logarithmes discrets. Grâce à la taille calculée, nous saurons quels paramètres prendre pour coder ces protocoles. Ils seront ainsi sécurisés vis-à-vis d'attaques réalisées avec nos algorithmes de résolution étudiés au préalable.

Dans un premier temps, nous présenterons des notions mathématiques nécessaires à la définition du problème du logarithme discret et à la suite de l'étude. Nous aborderons ensuite les différents algorithmes de résolutions. Enfin, nous étudierons les deux protocoles de cryptographie à clé publique.

2. Tréguer, « Pouvoir et résistance dans l'espace public : une contre-histoire d'Internet (XVe -XXIe siècle) », sous-section 7.3.1 « Le génie cryptographique sort de sa bouteille », pp. 261–264.

3. Cet aspect est développé plus longuement dans notre synthèse culturelle disponible en annexe A.

1. Prérequis mathématiques nécessaires à la définition et à l'étude du problème

Une première définition du problème du logarithme discret a été donnée dans l'introduction, cependant, mathématiquement elle est incomplète. En effet, la définition correcte fait appel à certaines notions mathématiques qu'il est nécessaire d'introduire au préalable. De même, les différentes étapes de l'étude de ce problème utilisent aussi différents concepts et objets mathématiques qu'il est utile de rappeler ou définir. À ces fins, cette partie du mémoire présentera des définitions et théorèmes.

Notations Les notations suivantes seront adoptées dans l'ensemble du mémoire :

- $a|b$ signifie que a divise b, de même $a \nmid b$ signifie que a ne divise pas b.
- La partie entière supérieure d'un réel x est notée $\lceil x \rceil$, la partie inférieure $\lfloor x \rfloor$.
- $A \subseteq B$ signifie que A est inclu dans B ou que A est égal à B.

1.1 Généralités sur les groupes

Un *groupe* est un ensemble G muni d'une opération, notée ici $\cdot$, définie par $G \times G \to G$ (*id est* loi de composition interne binaire), qui vérifie les propriétés suivantes :

Associativité Quels que soient x, y et z, trois éléments de G, la relation $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ est vérifiée.

Existence d'un élément neutre Il existe un élément e de G qui vérifie $x \cdot e = e \cdot x = x$, quel que soit $x \in G$.

Existence d'un symétrique de tous les éléments du groupe Pour tout $x \in G$, il existe un élément $y \in G$ tel que $x \cdot y = y \cdot x = e$ où e est l'élément neutre de l'opération.

De plus, si l'opération vérifie la relation $x \cdot y = y \cdot x$ pour tout $x, y \in G$, elle est dite commutative et le groupe est alors appelé *abélien* ou *commutatif*.

Un groupe est dit *additif* s'il est muni de l'opération addition $(+)$ dont l'élément neutre est 0. Le symétrique d'un élément x est alors noté $-x$. Un groupe est dit *multiplicatif* s'il est muni de l'opération multiplicative $(\times)$. L'élément neutre de cette opération est 1. Le symétrique d'un élément x de ce groupe est noté x^{-1}, il est aussi appelé *inverse* de x. Cependant, la distinction entre additif et multiplicatif n'est qu'une différence de notation et non de propriétés intrinsèques.

Étant donnés deux groupes $(G, \cdot)$ et $(G', \bullet)$, une application $f : G \to G'$ est un *morphisme de groupe* si pour tout élément g_1, g_2 dans G, la relation $f(g_1 \cdot g_2) = f(g_1) \bullet f(g_2)$ est vérifée. L'élément neutre de G est noté e, celui de G' est noté e'. L'ensemble des $g \in G$ tels que $f(g) = e'$ est appelé le *noyau* de f, il est noté $\ker f$. En particulier, e appartient au noyau de f.

De plus, si l'application f est *bijective*, alors on dit que f induit un *isomorphisme de groupe*. La bijection réciproque de f, notée f^{-1}, induit aussi un morphisme de groupe.

Proposition 1. *Soit $f : G \to G'$ un morphisme de groupe. Alors f est injective* si et seulement si $\ker f = \{e\}$.

Démonstration. Supposons f injective. Alors pour tout élément g du noyau de f, nous avons la relation $f(g) = e' = f(e)$, par injection $g = e$ et $\ker f \subset \{e\}$. Or par définition $\{e\} \subset \ker f$ ce qui nous permet d'en déduire que $\ker f = \{e\}$.

Réciproquement, supposons que $\ker f = \{e\}$. Soient $g_1, g_2 \in G$ tels que $f(g_1) = f(g_2)$. Nous avons alors $f(g_1 \cdot g_2^{-1}) = f(g_1) \bullet (f(g_2))^{-1} = e'$. Par conséquent $g_1 \cdot g_2^{-1} \in \ker f$ et par hypothèse, nous en déduisons que $g_1 \cdot g_2^{-1} = e$, c'est-à-dire que $g_1 = g_2$. Ainsi f est injective. □

Groupes finis

Un groupe G dont le nombre d'éléments qui le composent est fini est appelé *groupe fini*. Ce nombre est appelé *ordre* de G et est noté $\#G$ (il s'agit de la même notation que celle du cardinal d'un ensemble).

Notation

Dans un groupe multiplicatif $(G, \times)$, nous utiliserons la simplification consistant à sous-entendre l'opération : pour tout $x, y \in G$, $x \times y = xy$.

Un groupe formé par l'ensemble G muni de l'opération $\cdot$ est noté $(G, \cdot)$. Cependant pour une lecture plus facile, nous ferons régulièrement référence au groupe quelconque G ce qui signifiera le groupe $(G, \cdot)$ où $\cdot$ est une opération quelconque. Il en sera de même pour les groupes multiplicatifs et additifs. Dans un groupe quelconque $(G, \cdot)$, appliquer k fois l'opération $\cdot$ à un élément $g \in G$, se note g^k. Quand le groupe est muni de la loi multiplicative, cette notation sera conservée. Cependant, dans le cas d'une loi additive, la notation kg sera préférée.

1.1.1 Sous-groupe

Une partie H d'un groupe $(G, \cdot)$ est un *sous-groupe* de G si elle vérifie les propriétés suivantes :

— H est stable par l'opération $\cdot$ si pour tout $x, y \in H$, $(x \cdot y) \subset H$.

— L'élément neutre de G est contenu dans H.

— Pour tout $x \in H$, son symétrique est compris dans H.

Soit $x \in G$. L'ensemble $\langle x \rangle$ est définit par $\langle x \rangle = \{kx \,:\, k \in \mathbb{Z}\} \subseteq G$ si G est un groupe additif et $\langle x \rangle = \{x^k \,:\, k \in \mathbb{Z}\} \subseteq G$ si G est un groupe multiplicatif.

L'ensemble $\langle x \rangle$ est un sous groupe de G : on montre trivialement que les propriétés citées précédemment sont validées. On dit que $\langle x \rangle$ est *engendré* par x. L'ordre de x désigne alors l'ordre du sous-groupe $\langle x \rangle$. Il sera noté avec la même notation que le cardinal du groupe : $\#x$.

Théorème 2 (Théorème de Lagrange)**.** *Soit G un groupe fini et H un sous-groupe de G. Alors l'ordre de H divise l'ordre de G : $\#H | \#G$.*

Démonstration. Soit $\mathcal{R}$ la relation binaire définie pour tout $x, y \in G$ par $x\mathcal{R}y \Leftrightarrow xy^{-1} \in H$. Montrons que $\mathcal{R}$ est une relation d'équivalence, c'est-à-dire qu'elle satisfait les points suivants :

— Réflexivité ($\forall x \in G$, $x\mathcal{R}x$) : on a bien $xx^{-1} = e \in H$.

— Symétrie ($\forall x, y \in G$, $x\mathcal{R}y \Rightarrow y\mathcal{R}x$) : si $xy^{-1} \in H$, comme les sous-groupes sont stables par opération et admettent des symétriques, alors on a $yx^{-1} \in H$.

— Transitivité ($\forall x, y, z \in G$, $(x\mathcal{R}y \wedge y\mathcal{R}z) \Rightarrow x\mathcal{R}z$) : par produit on a $xy^{-1} \cdot yz^{-1} = xz^{-1} \in H$.

Donc $\mathcal{R}$ est une relation d'équivalence.

Soit $x \in G$. Déterminons la classe d'équivalence $\overline{x}$. Comme H est un sous-groupe alors :

$$\overline{x} = \{y \in G : x\mathcal{R}y\} = \{y \in G : \exists h \in H, xy^{-1} = h\} = \{y \in G : \exists h \in H, y = h^{-1}x\} = H \cdot x.$$

Par définition : $G = \bigcup_{x \in G} \{\overline{x}\}$. Soient $x, y \in G$. Si $\overline{x} \neq \overline{y}$ alors $\overline{x} \cap \overline{y} = \varnothing$. L'ensemble G est partitionné en un nombre fini r de classes d'équivalences notées $\overline{x_i}$. Nous avons alors :

$$G = \bigcup_{1 \leqslant i \leqslant r} \overline{x_i} = \bigcup_{1 \leqslant i \leqslant r} H \cdot x_i. \quad \Rightarrow \quad G = \sum_{i=1}^{r} \#(H \cdot x_i).$$

Soit φ l'application définie par $\begin{array}{rccc} \varphi : & H & \longrightarrow & Hx \\ & y & \mapsto & yx \end{array}$. Soient $y_1, y_2 \in H$. Si $xy_1 = xy_2$, alors $y_1 = y_2$. D'où, par contraposée, $y_1 \neq y_2 \Rightarrow xy_1 \neq xy_2$. Ainsi, φ est injective. De plus, elle est aussi surjective car $Hx = \{yx : y \in H\} = \{\varphi(y) : y \in H\} = \varphi(H)$. Nous avons donc prouvé la bijectivité de φ.

Nous obtenons donc l'implication $\#(H \cdot x_i) = \#H \ \Rightarrow \ \#G = r\#H$ qui permet de démontrer le théorème. □

1.1.2 Groupes cycliques

Lorsqu'il existe $g \in G$ tel que $G = \langle g \rangle$ alors G est un groupe *monogène*. L'élément g est appelé générateur du groupe G. De plus, si G est fini, alors c'est un groupe *cyclique*.

Soit n l'ordre de G. L'ordre de g est alors égal à n. Le nombre n est le plus petit entier naturel strictement positif à vérifier l'égalité $g^n = e$, où e est l'élément neutre de l'opération du groupe G[1].

Théorème 3. *Soit G un groupe fini d'ordre premier alors G est cyclique.*

Démonstration. Soit $g \in G \setminus \{1\}$. Comme 1 et g appartiennent à $\langle g \rangle$ alors $\#g \geqslant 2$. Par le théorème 2 (Lagrange), l'ordre de g divise l'ordre de G, or ce dernier est premier ce qui aboutit à l'égalité $\#g = \#G$. □

1.1.3 Définition du problème du logarithme discret

Après avoir introduit différentes définitions autour du concept de groupe, nous pouvons à présent poser la définition complète du problème du logarithme discret dans un groupe générique.

Définition 1 (Problème du logarithme discret)**.** Soit G un groupe cyclique connu d'ordre n engendré par g. Étant donné $h \in G$, le problème du logarithme discret consiste à déterminer l'unique entier x tel que $h = g^x$, avec $0 \leqslant x < n$. On note alors $x = \log_g h$.

1.2 Utilisation de la division euclidienne et des nombres premiers entre eux

Pour la poursuite de notre étude, nous avons besoin de présenter deux théorèmes fondamentaux concernant la division euclidienne : le théorème de Bézout et le théorème de Lamé. Le premier sert à donner une condition pour que deux entiers soient premiers entre eux. Le second sera utile lors de la manipulation de l'algorithme d'Euclide étendu.

Rappelons que deux entiers a et b sont premiers entre eux *si et seulement si* $\text{pgcd}(a, b) = 1$. Un nombre p est dit *premier* s'il admet exactement deux diviseurs : 1 et lui-même.

Théorème 4 (Théorème de Bachet-Bézout — Identité de Bézout)**.** *Soient $a, b \in \mathbb{Z}^*$. Alors il existe $u, v \in \mathbb{Z}$ tels que $au + bv = \text{pgcd}(a, b)$.*

1. Wassef, *Arithmétique : application aux codes correcteurs et à la cryptographie : cours & 122 exercices corrigés : licence de mathématiques*, théorème 4.3, page 60.

Démonstration. Soient $a, b \in \mathbb{Z}^*$. Notons E l'ensemble des entiers naturels non nuls pouvant s'écrire sous la forme $au + bv$ avec $u, v \in \mathbb{Z}$. Il existe des éléments de $\mathbb{N}^*$ qui peuvent s'exprimer sous la forme $au + bv$. Par exemple, $(u, v) = (1, 0)$ si $a > 0$ ou $(u, v) = (-1, 0)$ sinon. Par conséquent, E n'est pas vide. De plus, E est un sous-ensemble de $\mathbb{N}$, donc E admet un plus petit élément noté c. Ainsi, $c \in \mathbb{N}^*$.

Notons $c = au_0 + bv_0$, avec $u_0, v_0 \in \mathbb{Z}$. Soit $D = \text{pgcd}(a, b)$. L'entier D divise c car D divise a et b, donc il divise $au_0 + bv_0$.

Démontrons que c divise D. La division euclidienne de a par c donne $a = cq + r$ où $r, q \in \mathbb{Z}$ avec $0 \leqslant r < c$. Supposons $r \neq 0$, alors $r = a - cq = a(1 - u_0 q) + b(-v_0 q)$. Comme $r > 0$, alors $r \in E$. Or $r < c$, nous obtenons alors une contradiction car c est le plus petit élément de E. Nous avons donc démontré par l'absurde que $r = 0$ et donc que c divise a. On démontre de façon analogue que c divise b. Ainsi, c est un diviseur commun à a et b donc c divise D. Finalement,puisque c et D sont strictement positifs, $c = D$. □

Théorème 5 (Théorème de Bézout)**.** *Soient $a, b \in \mathbb{Z}^*$. Les entiers a et b sont premiers entre eux* si et seulement s'il *existe $u, v \in \mathbb{Z}$ tel que $au + bv = 1$.*

Démonstration. Si a et b sont premiers entre eux, alors $\text{pgcd}(a, b) = 1$. D'après l'identité de Bézout, il existe donc $u, v \in \mathbb{Z}$ tels que $au + bv = 1$.

Réciproquement, s'il existe deux entiers relatifs u et v tels que $au + bv = 1$ alors, en posant $D = \text{pgcd}(a, b)$, D divise a et D divise b donc D divise $au + bv$, c'est-à-dire que D divise 1. Finalement, comme $D > 0$, nous pouvons en conclure que $D = 1$. □

Théorème 6 (Théorème de Lamé)**.** *Soient $a, b \in \mathbb{N}^*, b < a$. Alors le nombre N de division euclidienne lors de l'algorithme d'*Euclide *est au maximum de $N_{max} = \lfloor \log_\varphi b \rfloor + 1$, où $\varphi = \frac{1+\sqrt{5}}{2}$ est le nombre d'or.*

Démonstration. Soient $a, b \in \mathbb{N}^*, b < a$. Posons $r_0 = a$ et $r_1 = b$ et appliquons l'algorithme d'Euclide comme si nous voulions calculer le pgcd de r_0 et r_1.

$$\begin{cases} r_0 = r_1\, q_0 + r_2 & 0 < r_2 < r_1 \\ r_1 = r_2\, q_1 + r_3 & 0 < r_3 < r_2 \\ (...) & \\ r_{n-1} = r_n\, q_{n-1} + r_{n+1} & 0 < r_{n+1} < r_n \\ r_n = r_{n+1}\, q_n. & \end{cases}$$

Remarquons que $q_n > 1$: en effet, si q_n valait 1, alors nous aurions l'égalité $r_n = r_{n+1}$ qui contredit l'inégalité $0 < r_{n+1} < r_n$ donnée par l'algorithme d'Euclide.

En effectuant les remontées successives de l'algorithme d'Euclide et en raisonnant avec des minorations, nous obtenons :

$$\begin{cases} r_{n+1} \geqslant 1 & \\ r_n = r_{n+1}\, q_n \geqslant 1 \times 2 = 2 & \\ r_{n-1} = r_n\, q_{n-1} + r_{n+1} \geqslant 2 \times 1 + 1 = 3 & \\ r_{n-2} = r_{n-1}\, q_{n-2} + r_n \geqslant 3 \times 1 + 2 = 5 & \\ (...) & \\ r_{k-2} = r_{k-1}\, q_{k-2} + r_k \geqslant r_{k-1} + r_k \quad \text{car} \quad \forall k \in \{2, 3, \ldots, n+1\},\ q_{k-2} \geqslant 1. & \end{cases}$$

Par récurrence directe, les termes de la suite $(r_i)_{i \in [\![0, n+1]\!]}$ sont minorés par les termes de la suite de Fibonacci, définie par :

$$\begin{cases} F_0 = 1, F_1 = 2 \\ F_{n+2} = F_{n+1} + F_n. \end{cases}$$

Montrons par récurrence la propriété $(P_n) : \forall n \in \mathbb{N},\ F_n \geqslant \varphi^n$ où φ est le nombre d'or. Initialisation : P_0 est vraie car $F_0 = 1 \geqslant \varphi^0 = 1$. À présent, supposons P_n vraie, nous obtenons alors l'inégalité suivante $F_{n+1} = F_n + F_{n-1} \geqslant \varphi^n + \varphi^{n-1} = \varphi^{n-1}(\varphi + 1)$. Or $\varphi^2 = \varphi + 1$, donc $F_{n+1} \geqslant \varphi^{n+1}$. Ainsi, nous avons montré par récurrence que pour tout $n \in \mathbb{N}$, $F_n \geqslant \varphi^n$.

Comme $b \geqslant F_n$, nous en déduisons l'inégalité $\log b \geqslant n \log \varphi$ soit $\log_\varphi b \geqslant n$. De plus, le nombre N de divisions euclidiennes vaut $n + 1$ puisque nous comptons la dernière, au reste nul. Nous avons alors $N \leqslant 1 + \log_\varphi b$ qui équivaut à $N \leqslant 1 + \lfloor \log_\varphi b \rfloor$ car N est un entier. □

1.3 Arithmétique dans $\mathbb{Z}/n\mathbb{Z}$

Dans le cas de notre mémoire, nous travaillerons principalement dans l'ensemble $\mathbb{Z}/n\mathbb{Z}$. À cet effet, il est nécessaire d'une part de définir précisément cet ensemble et d'autre part de poser des propositions et théorèmes utiles pour la suite de l'étude.

1.3.1 Congruences

Définition 2. Soient n, a, b trois entiers, $n > 0$. On dit que que a est *congru* à b *modulo* n s'il existe un entier relatif k tel que $a = b + kn$. On note $a \equiv b \mod n$. À l'opposé, si a n'est pas congru à b modulo n, on note $a \not\equiv b \mod n$.

Proposition 7. *Soient $a, b, c, d \in \mathbb{Z}$ et $n \in \mathbb{N}^*$. On a les propriétés suivantes :*

(i) Réflexivité : $a \equiv a \mod n$;

(ii) Symétrie : $a \equiv b \mod n \ \Rightarrow\ b \equiv a \mod n$;

(iii) Transitivité : $(a \equiv b \mod n \wedge b \equiv c \mod n) \ \Rightarrow\ a \equiv c \mod n$;

(iv) $(a \equiv c \mod n \ \wedge\ b \equiv d \mod n) \ \Rightarrow\ a + b \equiv c + d \mod n$;

(v) $(a \equiv b \mod n \ \wedge\ c \equiv d \mod n) \ \Rightarrow\ ac \equiv bd \mod n$.

Démonstration. Soient k et k' deux entiers relatifs.

(i) $a = a + 0 \times n$;

(ii) $a = b + kn$ donc $b = a - kn = a + k'n$;

(iii) $(a = b + kn \wedge b = c + k'n) \ \Rightarrow\ a = c + k'n + kn = c + (k' + k)n$;

(iv) $(a = b + kn \wedge c = d + k'n) \ \Rightarrow\ a + c = b + kn + d + k'n = b + d + (k + k')n$;

(v) $(a = b + kn \wedge c = d + k'n) \ \Rightarrow\ ac = (b + kn)(d + k'n) = bd + (bk' + dk + kk'n)n$.

□

1.3.2 Construction des groupes associés à l'ensemble $\mathbb{Z}/n\mathbb{Z}$

Définition 3. Soit $n \in \mathbb{N}^*$. L'ensemble $\mathbb{Z}/n\mathbb{Z}$ est défini par $\mathbb{Z}/n\mathbb{Z} = \{\overline{0}, \ldots, \overline{n-1}\}$ où $\overline{a}$ désigne la classe d'équivalence $\overline{a} = \{b \equiv a \mod n : b \in \mathbb{Z}\}$.

Groupe $(\mathbb{Z}/n\mathbb{Z}, +)$

L'addition dans l'ensemble $\mathbb{Z}/n\mathbb{Z}$ est définie par $\overline{a} + \overline{b} = \{x + y : x \in \overline{a},\ y \in \overline{b}\}$. Démontrons à présent la relation $\overline{a} + \overline{b} = \overline{a + b}$:

$$\overline{a} + \overline{b} = \{x + y : x \in \overline{a},\ y \in \overline{b}\} = \{a + kn + b + k'n : k, k' \in \mathbb{Z}\} = \{a + b + (k + k')n : k, k' \in \mathbb{Z}\}.$$

Comme k et k' sont des entiers relatifs, alors $(K = k + k') \in \mathbb{Z}$. Nous obtenons donc :

$$\overline{a} + \overline{b} = \{a + b + Kn : K \in \mathbb{Z}\} = \{z \equiv x + y \mod n : z \in \mathbb{Z}\} = \overline{a + b}.$$

Muni de cette addition, l'ensemble $\mathbb{Z}/n\mathbb{Z}$ forme un groupe commutatif fini d'ordre n. De plus, $(\mathbb{Z}/n\mathbb{Z}, +)$ est cyclique de générateur $\overline{1}$.

Groupe $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$

Nous définissons la multiplication dans cet ensemble par $\overline{a} \times \overline{b} = \overline{ab}$. Associée à l'ensemble $\mathbb{Z}/n\mathbb{Z}$, cette opération vérifie la propriété d'associativité et possède l'élément neutre $\overline{1}$. Pour former un groupe il faut qu'il existe dans l'ensemble un symétrique pour ses élément. Nous devons donc chercher si quel que soit $\overline{a} \in \mathbb{Z}/n\mathbb{Z}$, il existe $\overline{b} \in \mathbb{Z}/n\mathbb{Z}$ tel que :

$$\overline{ab} = \overline{1} \quad \Leftrightarrow \quad \{y \equiv ab \mod n : y \in \mathbb{Z}\} = \{z \equiv 1 \mod n : z \in \mathbb{Z}\} \quad \Leftrightarrow \quad ab \equiv 1 \mod n.$$

Cela revient donc à chercher l'inverse modulo n de a.

Théorème 8. *Soit $a \in \mathbb{Z}$. Alors a est inversible modulo n* si et seulement si $\text{pgcd}(a, n) = 1$.

Démonstration. Par définition, a est inversible modulo n si et seulement s'il existe un entier b tel que $ab \equiv 1 \mod n$. Par conséquent, il existe $k' \in \mathbb{Z}$ tel que $ab = k'n + 1$ ce qui équivaut à la relation $ab + kn = 1$ où $k = -k'$. Or d'après l'identité de Bézout, il existe $b, k \in \mathbb{Z}$ si et seulement si a et n sont premiers entre eux. Ainsi, l'inverse modulaire b de a existe si et seulement si cette condition est respectée. □

Par conséquent, pour former un groupe multiplicatif à partir de l'ensemble $\mathbb{Z}/n\mathbb{Z}$, il faut le réduire à un ensemble noté $(\mathbb{Z}/n\mathbb{Z})^\times$ défini par :

$$(\mathbb{Z}/n\mathbb{Z})^\times = \{\overline{a} \in \mathbb{Z}/n\mathbb{Z} : \text{pgcd}(a, n) = 1\}. \tag{1.1}$$

Le groupe $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$ ainsi formé est un groupe abélien fini. Si n est un nombre premier alors le groupe est cyclique [2].

Notations

Dans la suite du mémoire, nous utiliserons les notations suivantes.

— Le groupe $G = (\mathbb{Z}/n\mathbb{Z})$ signifiera que le groupe G peut être $(\mathbb{Z}/n\mathbb{Z}, +)$ ou $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$.

— $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$ signifiera que p est un nombre premier.

1.3.3 Caractérisation des éléments générateurs des groupes issus de $\mathbb{Z}/n\mathbb{Z}$

Proposition 9. *Soit $G = (\mathbb{Z}/n\mathbb{Z})$ un groupe cyclique d'ordre n engendré par g. Soit $x \in G$ tel que $x = g^k$, avec $0 \leqslant k \leqslant n - 1$. Alors x est un générateur de G* si et seulement si *k est premier avec n.*

Démonstration. L'entier x engendre G si et seulement s'il existe un entier j tel que $x^j = g$.

Sachant que $x = g^k$, alors $x^{kj-1} = 1$. Nous en déduisons que $kj - 1 = nm$, avec $m \in \mathbb{Z}$. D'après le théorème de Bézout (5), nous pouvons conclure que k est premier avec n. □

De plus, si G est un groupe cyclique d'ordre n, il y a autant de générateurs de G que d'entiers de $\{1, ..., n\}$ premiers avec n.

Éléments générateurs de $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$

Pour la suite de ce paragraphe, il est nécessaire de préciser que le groupe $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$ est d'ordre $p - 1$. En effet, $(\mathbb{Z}/p\mathbb{Z})^\times = \{\overline{a} \in \mathbb{Z}/p\mathbb{Z} : \text{pgcd}(a, n) = 1\} = \{\overline{1}, \ldots, \overline{p-1}\}$ car p est premier.

Soit $G = ((\mathbb{Z}/p\mathbb{Z})^\times, \times)$. Le nombre premier p est choisi tel que $p - 1 = n$ est facile à factoriser. L'ordre n se factorise alors sous la forme $n = \prod_{i=1}^{r} q_i^{e_i}$.

Théorème 10. *Soient $g \in G$, G d'ordre n, et d tel que $d|n$, $d \neq n$. Alors g est un générateur d'ordre d* si et seulement si *pour tout q_i, $g^{\frac{n}{q_i}} \neq 1$.*

2. Meunier, *Cours d'algèbre et d'algorithmique : applications à la cryptologie du RSA et logarithme discret*, corollaire 1, page 23.

Démonstration. Montrons d'abord que s'il existe un q_i tel que $\frac{g^n}{q_i} = 1$ alors g n'est pas un générateur. Si $d|\frac{n}{q_i}$ pour au moins l'un des q_i, alors $g^{\frac{n}{q_i}} = 1$ et donc g n'est pas un générateur d'ordre d. Par contraposée, si g est un générateur d'ordre d alors pour tous les éléments q_i, on a $g^{\frac{n}{q_i}} \neq 1$.

Montrons l'implication réciproque. Pour cela, prouvons que g n'est pas un générateur implique qu'il existe un q_i tel que $g^{\frac{n}{q_i}} = 1$. Soit $d = \#g$ tel que $g^d = 1$ et $d|n$. Si g n'est pas un générateur, alors $d \neq n$. Or $d|n$, donc $d|\frac{n}{q_i}$ pour un certain q_i. De plus, comme $g^d = 1$, alors $g^{\frac{n}{q_i}} = 1$ pour au moins l'un des q_i. Par contraposée, si $g^{\frac{n}{q_i}} \neq 1$ pour tout q_i, alors g est générateur d'ordre d. □

1.3.4 Groupes cycliques et $(\mathbb{Z}/n\mathbb{Z}, +)$

Théorème 11. *Soit G un groupe cyclique d'ordre n. Alors G est isomorphe à $(\mathbb{Z}/n\mathbb{Z}, +)$ et on note $G \cong \mathbb{Z}/n\mathbb{Z}$.*

Démonstration. Soient g un générateur de G d'ordre n et f l'application $\begin{array}{rcl} f : (\mathbb{Z}/n\mathbb{Z}, +) & \to & (G, \cdot) \\ k & \mapsto & g^k \end{array}$.

Soient $k_1, k_2 \in (\mathbb{Z}/n\mathbb{Z}, +)$, alors $f(k_1 + k_2) = g^{k_1+k_2} = g^{k_1} \cdot g^{k_2} = f(k_1) \cdot f(k_2)$, et donc f est un morphisme de groupe.

Montrons d'abord que f est bien définie pour tout $k \in \mathbb{Z}/n\mathbb{Z}$. Autrement dit, montrons que :

$$\forall x \in \mathbb{Z}, \quad x \equiv y \mod n \Rightarrow g^x = g^y.$$

Soit $x \in \mathbb{Z}$. Alors $x \equiv y \mod n \Leftrightarrow \exists z \in \mathbb{Z}, x = y + nz \Leftrightarrow g^x = g^{y+nz} \Leftrightarrow g^x = g^y \cdot g^{nz} = g^y \cdot (g^n)^z$. Or g engendre G qui est d'ordre n, donc $g^n = 1$. Ainsi, $g^x = g^y \cdot g^{nz} \Rightarrow g^x = g^y$. Finalement, $x \equiv y \mod n \Rightarrow g^x = g^y$ pour tout $x \in \mathbb{Z}$, et donc f est bien définie.

Montrons maintenant que f est bijective. Soit $k \in \ker f$. Comme k est un élément de $\mathbb{Z}/n\mathbb{Z}$ $(0 \leqslant k < n)$, on a donc $g^k = 1$. Cependant, n est le plus petit entier strictement positif vérifiant cette propriété. Ainsi, $k = 0$ et $\ker f = \{\overline{0}\}$. Nous déduisons alors de la proposition 1 que f est injective. De plus, f est surjective car g engendre G, donc f est bijective. Finalement, G est isomorphe à $(\mathbb{Z}/n\mathbb{Z}, +)$. □

2. Algorithmes de résolution du problème du logarithme discret

Afin de savoir si un problème mathématique peut être utilisé dans des protocoles cryptographiques, il est nécessaire d'étudier au préalable les algorithmes de résolution de ce problème. Catégoriser la difficulté que représente cette résolution permet de déterminer le niveau de sécurité des protocoles basés sur le problème étudié en fonction des paramètres choisis pour l'implémentation [1].

2.1 Complexité d'un algorithme

Pour étudier les algorithmes de résolution du problème du logarithme discret, nous allons utiliser un outil nommé complexité qui permet de quantifier les besoins d'un algorithme pour renvoyer la solution en fonction de la variable (ou de sa taille) prise en entrée. La complexité ne dépend pas du type de machine utilisée ni du langage de programmation, elle est théorique.

Les besoins évalués peuvent être de plusieurs types mais nous utiliserons ici uniquement la complexité *en temps* et celle *en mémoire*. La première permet d'estimer le nombre d'opérations effectuées lors de l'exécution de l'algorithme, la seconde le nombre d'entrées en mémoire. Il est possible d'établir la complexité *en temps* et celle *en mémoire* d'un algorithme dans trois catégories : dans le meilleur des cas, dans le pire des cas ou enfin en moyenne.

Déterminer le nombre exact d'actions effectuées par l'algorithme est difficile. De plus, cet outil est utilisé pour comprendre l'évolution des besoins quand la taille de la variable en entrée augmente : c'est pourquoi la complexité est exprimée avec des notations asymptotiques [2]. Ici nous utiliserons la notation de Landau $\mathcal{O}$ qui permet de négliger les constantes. En notant $\mathcal{C}$ la complexité en fonction de n la variable d'entrée, $\mathcal{C} = \mathcal{O}(f(n))$ — où f est une fonction positive — signifie que la complexité est majorée par $f(n)$ à la constante près. Nous avons donc l'inégalité suivante : $0 < \mathcal{C} \leqslant cf(n)$ où c est une constante strictement positive.

Une fois que la complexité d'un algorithme a été définie, il est possible de déterminer s'il est efficace ou non en analysant la classe de la complexité obtenue. Dans tout ce mémoire, la variable d'entrée correspond à l'ordre n du groupe dans lequel nous travaillons. La taille de la variable d'entrée désignera ici le nombre de chiffre nécessaire à l'écriture de l'ordre : par exemple si l'ordre est $n = 123\,456\,789$, sa taille sera 9. Nous admettrons que la taille de la variable d'entrée est de l'ordre de $\log n$ [3]. Nous pouvons alors poser la définition des quatre classes de complexité suivantes.

Complexité constante en $\mathcal{O}(1)$.

Complexité polynomiale en $\mathcal{O}((\log n)^\alpha)$, avec $\alpha \in \mathbb{N}^*$.

Complexité sous-exponentielle en $\mathcal{O}\big(\exp\big(k\,(\log n)^\alpha(\log\log n)^{1-\alpha}\big)\big)$ avec $0 < \alpha < 1$ et k une constante strictement positive.

1. L'implémentation d'un algorithme consiste à écrire un algorithme théorique dans un langage de programmation pour qu'il puisse être exploité avec un ordinateur.
2. Menezes, Van Oorschot et Vanstone, *Handbook of applied cryptography*, page 58.
3. Meunier, *Cours d'algèbre et d'algorithmique : applications à la cryptologie du RSA et logarithme discret*, remarque 2, page 80.

Complexité exponentielle en $\mathcal{O}(\exp(\beta \log n)) = \mathcal{O}(n^{\beta})$, avec β une constante strictement positive. Un algorithme est dit efficace si sa complexité en temps est polynomiale [4]. En cryptographie, un problème est dit cassé s'il peut être résolu avec un algorithme d'une telle complexité.

Afin d'expliciter la notion de complexité, nous allons étudier l'exemple de l'algorithme de résolution du problème du logarithme discret le plus simple, dit naïf (ou exhaustif). Cette méthode consiste à calculer successivement les puissances du générateur g jusqu'à obtenir une égalité avec h. La solution correspondra alors à la valeur de l'exposant de g.

Le groupe engendré par g est d'ordre n donc au maximum il faut calculer n puissances de g pour obtenir la solution. Sachant qu'une puissance est obtenue en multipliant la puissance précédente par g, au maximum l'algorithme a besoin d'effectuer n opérations pour obtenir la solution. Plus précisément, à chaque fois que l'on calcule une puissance, il est aussi nécessaire de prendre en compte le fait de la comparer à h ainsi que d'incrémenter de 1 la variable qui compte le nombre de multiplications effectuées ("la valeur" de l'exposant). Pour résumer, à chaque calcul de puissance, l'algorithme effectue 3 actions donc le nombre maximal d'opérations à effectuer est de $3n$. Cependant, comme les constantes multiplicatives et additives sont négligées, la complexité en temps de l'algorithme naïf est en $\mathcal{O}(n)$.

Néanmoins, cet algorithme peut rapidement se montrer insuffisant dès lors que n est pris grand. Nous allons donc étudier des algorithmes ayant une complexité moindre.

2.2 Baby-Step Giant-Step

L'algorithme Baby-Step Giant-Step — en français : Pas de bébé, pas de géant — permet de résoudre le problème du logarithme discret dans un groupe cyclique quelconque G, de générateur g d'ordre n. Pour rappel, le but de l'algorithme est de déterminer x le plus petit élément de G vérifiant $h = g^x$ avec h et g connus.

2.2.1 Algorithme

Soit $m = \lceil \sqrt{n} \rceil$. La division euclidienne nous permet d'exprimer x sous la forme :

$$x = y + mz \quad \text{où} \quad \begin{cases} 0 \leqslant y < m \\ 0 \leqslant z < m. \end{cases}$$

En effet, par définition $0 \leqslant x < n$, par conséquent $0 \leqslant y + mz < n$. Nous obtenons alors la majoration recherchée puisque $0 \leqslant z \leqslant \frac{y}{m} + z < \frac{n}{m} \leqslant m$.

L'équation $h = g^x$ peut donc se réécrire $h = g^y g^{mz}$ ce qui conduit à :

$$h\left(g^{-m}\right)^z = g^y. \tag{2.1}$$

Pour résoudre le problème du logarithme discret, il suffit donc de trouver un couple $(y, z) \in G \times G$ satisfaisant cette équation. L'algorithme Baby-Step Giant-Step consiste à créer deux listes et à les comparer pour trouver ce couple.

Tout d'abord, il faut chercher l'ensemble des éléments — dits Baby-Step— de la forme g^y. Nous obtenons alors la liste :

$$Baby - Step = \left[1, g, g^2, \ldots, g^{m-1}\right]. \tag{2.2}$$

Ensuite, il faut déterminer l'ensemble des éléments — dits Giant-Step — de la forme $h(g^{-m})^z$. Ils sont stockés dans la liste :

$$Giant - Step = \left[h, hg^{-m}, h\left(g^{-m}\right)^2, \ldots, h\left(g^{-m}\right)^{m-1}\right]. \tag{2.3}$$

4. Menezes, Van Oorschot et Vanstone, *Handbook of applied cryptography*, page 59.

Les éléments des listes *Baby-Step* et *Giant-Step* seront définis comme les termes de deux suites géométriques de raison respective g et g^{-m}.

Par une comparaison efficace des deux listes jusqu'à l'obtention d'une égalité, nous pouvons trouver le couple (y, z) solution de l'équation (2.1). Pour obtenir la solution du problème, il ne reste plus qu'à calculer la valeur de x tel que $x = y + mz$.

2.2.2 Complexités de l'algorithme et améliorations

Complexité en temps

Pour déterminer la complexité en temps de l'algorithme, il faut prendre en compte le temps nécessaire pour la réalisation des différentes étapes : la construction des listes *Baby-Step* et *Giant-Step* ainsi que les comparaisons pour trouver le couple (y, z) présent dans les deux listes.

La création de chaque liste nécessite un nombre d'opérations de l'ordre de $\mathcal{O}(m) = \mathcal{O}\left(n^{1/2}\right)$. En effet, chaque liste étant composée des m premiers termes d'une suite géométrique, le calcul d'un élément revient donc à réaliser une multiplication de l'élément précédent par la raison de la suite.

Les listes *Baby-Step* et *Giant-Step* contiennent chacune m éléments. Pour optimiser la comparaison des éléments, il est préférable de trier les listes préalablement ce qui prend $\mathcal{O}(m \log m)$ opérations. Ainsi, la comparaison des éléments entre les deux listes se fait de manière dichotomique. Pour rechercher la présence d'un élément de la première liste dans la seconde, il faut $\mathcal{O}(\log m)$ comparaisons. Comme la première liste comporte m éléments, nous en déduisons qu'il faudra $\mathcal{O}(m \log m)$ comparaisons. Ainsi par cette méthode, en tenant compte que $m^2 \simeq n$, nous transformons la complexité en temps de l'étape de comparaisons de $\mathcal{O}(n)$ à $\mathcal{O}\left(n^{1/2} \cdot \frac{1}{2} \log n\right)$. Dans la version actuelle de notre algorithme, et sans prendre en compte les améliorations qui vont suivre, une comparaison entre les deux listes de manière linéaire [5] est plus efficace en $\mathcal{O}(m)$. Néanmoins, cela n'a pas d'impact sur la complexité finale et n'est pas compatible avec les améliorations présentées ensuite.

Finalement, la complexité en temps, notée $\mathcal{C}_t$, de l'algorithme Baby-Step Giant-Step est :

$$\mathcal{C}_t = \mathcal{O}\left(2n^{1/2} + n^{1/2} \log n + n^{1/2} \cdot \frac{1}{2} \log n\right) = \mathcal{O}\left(n^{1/2} \log n\right). \tag{2.4}$$

Complexité en mémoire

Les listes *Baby-Step* et *Giant-Step* contiennent chacune m éléments ce qui correspond à une complexité en mémoire, notée $\mathcal{C}_m$, en $\mathcal{O}(m)$. L'algorithme Baby-Step Giant-Step a donc une complexité en mémoire : $\mathcal{C}_m = \mathcal{O}\left(n^{1/2}\right)$ Cette complexité en mémoire élevée peut rapidement créer une situation de saturation de la mémoire, d'autant plus que pour le problème du logarithme discret, n est souvent pris grand. Il existe cependant plusieurs méthodes pour diminuer cette consommation de mémoire.

Premièrement, nous pouvons éviter de garder en mémoire tous les éléments de la liste *Giant-Step* en même temps. Pour cela, il faut générer le premier élément de la liste puis le comparer par dichotomie à la liste *Baby-Step* (triée). Ensuite, après avoir multiplié cet élément afin d'obtenir l'élément suivant de la liste, nous recommençons la comparaison. Au maximum, cette procédure est effectuée m fois. Cette méthode permet de réduire l'ordre du nombre de variables stockées de $\mathcal{O}(2m)$ à $\mathcal{O}(m)$.

La seconde méthode consiste à considérer dans un premier temps tous les éléments des listes *Baby-Step* et *Giant-Step* avec les valeurs de y et z comprises entre 0 et $\frac{m}{2}$ et de les comparer. Tout en conservant la première liste, nous générons la seconde pour des valeurs de z comprises entre $\frac{m}{2}$ et m. Ensuite, nous recommençons la procédure en gardant la seconde liste avec z compris entre $\frac{m}{2}$ et m et en générant la première pour y variant de $\frac{m}{2}$ à m. Cette méthode est un compromis entre le temps et la mémoire. Elle permet de diviser la consommation en mémoire par deux, mais multiplie cependant le temps d'exécution par quatre. D'une manière générale, l'utilisation de cette méthode pour obtenir

5. Par parcours des deux listes en simultanée et avançant dans la liste où l'élément regardé est le plus petit.

des listes de taille $\frac{m}{k}$ permet de diviser par k l'utilisation de la mémoire mais multiplie la complexité en temps par k^2.

2.3 Rho de Pollard

L'algorithme Baby-Step Giant-Step est fonctionnel mais a pour grand défaut sa consommation importante de mémoire ce qui pose rapidement problème dès que les nombres utilisés deviennent grands. À l'inverse, l'algorithme Rho de Pollard est un algorithme probabiliste dont la consommation de mémoire est constante [6].

2.3.1 Paradoxe des anniversaires

> Dans une classe de 23 élèves, alors qu'une année est composée de 365 (ou 366) jours, la probabilité que deux personnes aient la même date d'anniversaire est de plus de 50%.

Le paradoxe ainsi soulevé est appelé le paradoxe des anniversaires [7]. En généralisant ce paradoxe probabiliste dans un groupe G d'ordre n : nous allons étudier la probabilité d'obtenir deux éléments identiques parmi k éléments tirés dans G.

Soit $F_{k,n}$ l'évènement « parmi les k éléments tirés dans un groupe d'ordre n, au moins deux sont identiques » et son complémentaire $F_{k,n}^C$: « les k éléments tirés dans un groupe d'ordre n sont distincts ».

Le tirage successif de k éléments forme une liste ordonnée. Dans un groupe de cardinal n, il est possible de former n^k listes ordonnées différentes. De plus, il y a $\frac{n!}{(n-k)!}$ cas satisfaisant l'évènement $F_{k,n}^C$, la probabilité de ce dernier est donc :

$$P(F_{k,n}^C) = \frac{n!}{(n-k)!\,n^k} = \prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right) \quad \Rightarrow \quad P(F_{k,n}) = 1 - \prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right).$$

Soit $P(F_{k,n}) = \alpha$, avec $\alpha \in [0,1[$. Sachant que pour tout $x \in \mathbb{R}$, $1+x \leqslant e^x$, il est possible de minorer cette probabilité :

$$\alpha \geqslant 1 - \prod_{i=1}^{k-1}\exp\left(-\frac{i}{n}\right) = 1 - \exp\left(-\frac{1}{n}\sum_{i=1}^{k-1} i\right) \quad \Leftrightarrow \quad \alpha \geqslant 1 - \exp\left(-\frac{k(k-1)}{2n}\right).$$

Cette inégalité équivaut alors à $-2n \cdot \log(1-\alpha) \geqslant k(k-1)$. De plus, $k(k-1) \geqslant (k-1)^2$. Nous obtenons alors une majoration du nombre d'éléments à tirer pour obtenir une collision en fonction de la probabilité choisie :

$$k \leqslant 1 + \sqrt{-2n\,\log(1-\alpha)}. \tag{2.5}$$

Ainsi, si nous nous replaçons dans le cadre du paradoxe des anniversaires énoncé au début — $\alpha = 0.5$ et $n = 365$ — nous obtenons bien $k \simeq 23$.

2.3.2 Probabilité d'être inversible

À la fin de l'algorithme Rho de Pollard, nous avons besoin d'isoler x dans une équation de la forme $a \equiv xb \mod n$. Il nous faut donc utiliser l'inverse modulaire pour obtenir la solution. L'efficacité de cet algorithme repose sur la probabilité que cette inversion soit possible.

Soit $x \in \mathbb{Z}$, nous voulons déterminer la probabilité que x soit inversible modulo n. Autrement dit, déterminons la probabilité que x et n soient premiers entre eux.

6. Pollard, « Monte Carlo Methods for Index Computation (mod p) ».
7. Pomerance, « Elementary thoughts on discrete logarithms ».

Soit $n = p^k$ (décomposition en facteurs premiers). Alors nous avons :

$$\#\{x \in [\![0, n-1]\!] : p|x\} = \frac{n}{p} \qquad \Rightarrow \quad \#\{x \in [\![0, n-1]\!] : p \nmid x\} = n - \frac{n}{p} = p^k\left(1 - \frac{1}{p}\right).$$

Soit $n = {p_1}^{k_1}{p_2}^{k_2}\dots{p_r}^{k_r}$ (décomposition en facteurs premiers). Nous en déduisons :

$$\#\{x \in [\![0, n-1]\!] : \forall p_i,\ p_i \nmid x\} = \prod_{p_i|n} \#\{x \in [\![0, n-1]\!] : p_i \nmid x\} = n\prod_{p_i|n}\left(1 - \frac{1}{p_i}\right).$$

Ainsi, la probabilité que n et x soient premiers entre eux est égale à :

$$\frac{n\prod_{p_i|n}\left(1 - \frac{1}{p_i}\right)}{n} = \prod_{p_i|n}\left(1 - \frac{1}{p_i}\right).$$

Nous cherchons à minorer cette probabilité. Il est possible de montrer [8] que pour tout $x \in [-0.5, +\infty[$ il existe une constante $C \geqslant 2$ telle que $\log(1+x) \geqslant x - Cx^2$. Nous pouvons alors écrire :

$$\prod_{p_i|n}\left(1 - \frac{1}{p_i}\right) = \exp\left(\sum_{p_i|n}\log\left(1 - \frac{1}{p_i}\right)\right) \geqslant \exp\left(-\sum_{p_i|n}\frac{1}{p_i} - C\sum_{p_i|n}\frac{1}{p_i^2}\right).$$

La série $\sum_{p_i|n}\frac{1}{p_i}$ est majorée par la série $\sum_{p\leqslant n}\frac{1}{p}$, avec p premier, qui diverge et est équivalente à $\log\log n$ [9]. La série $\sum_{p_i|n}\frac{1}{p_i^2}$ est convergente et telle que $-C\sum_{p_i|n}\frac{1}{p_i^2} = K$, où K une constante. Par conséquent :

$$\exp\left(-\sum_{p_i|n}\frac{1}{p_i} - C\sum_{p_i|n}\frac{1}{p_i^2}\right) \geqslant \exp(-\log\log n + K) = \frac{e^K}{\log n}. \tag{2.6}$$

Quand n tend vers $+\infty$, $\frac{e^K}{\log n}$ tend vers 0. Cependant, cette décroissance est très faible et garantit l'existence d'un inverse modulaire avec une probabilité correcte lorsque l'on se place dans des groupes où n augmente fortement. Ainsi, la possibilité de devoir recommencer la résolution du problème est faible par rapport à la taille du groupe.

Par ailleurs, en notant $\Phi(N)$ le nombre d'éléments inversibles dans l'intervalle $[\![1, N[\![$, nous avons l'inégalité $\frac{\Phi(N)}{N} \geqslant \frac{e^K}{\log n}$. Si N est premier, alors en posant $N = p$, nous avons $\text{pgcd}(a, p) = p$ si $a = p$ et $\text{pgcd}(a, p) = 1$ sinon. Nous en déduisons $\frac{\Phi(p)}{p} = \frac{p-1}{p} = 1 - \frac{1}{p}$.

Si p tend vers $+\infty$, alors $\frac{\Phi(p)}{p}$ tend vers 1. Cela signifie que si p est premier, alors les éléments de l'ensemble $\mathbb{Z}/p\mathbb{Z}$ ont une probabilité quasi certaine d'être inversible quand p devient grand. Cette proposition nous sera utile pour l'amélioration de l'algorithme.

2.3.3 Algorithme

Le principe de l'algorithme est de comparer successivement deux éléments distincts dans un ensemble de n éléments jusqu'à trouver une collision (*id est* deux éléments égaux) afin de résoudre le problème du logarithme discret. D'une part, grâce au paradoxe des anniversaires (*cf* section 2.3.1), nous savons que la probabilité de l'obtenir au bout d'un nombre limité de comparaison est grande. Par exemple, après environ $3\sqrt{n}$ comparaisons, la probabilité d'avoir eu une égalité est de plus de 99%. L'usage de ce paradoxe probabiliste permet donc dans la plupart des cas de limiter les opérations à

8. La démonstration est disponible en annexe B.

9. Dans notre mémoire, ce résultat sera admis d'après Hardy et Wright, *Introduction à la théorie des nombres*, théorème 427, page 451.

effectuer. De plus, la différence principale avec l'algorithme précédent consiste à comparer ces éléments *sans les stocker* en introduisant une relation de récurrence entre eux.

Pour sélectionner les éléments à comparer, deux suites $(x_i)_{i\in\mathbb{N}}$ et $(y_i)_{i\in\mathbb{N}}$ sont générées à partir d'une fonction pseudo-aléatoire[10] f de la forme $f(h^{a_n}g^{b_n}) = h^{a_m}g^{b_m}$, où le couple (a_m, b_m) est facilement calculable à partir de (a_n, b_n). Les deux suites satisfont la relation de récurrence suivante :

$$\begin{cases} x_{i+1} = f(x_i) \\ y_{i+1} = f(f(y_i)) \end{cases} \quad \text{avec} \quad y_i = x_{2i}$$

et ont pour termes initiaux $x_0 = y_0$, un élément choisi aléatoirement dans le groupe G.

La fonction f associe une unique valeur à un x donné, c'est pourquoi une fois que la fonction redonnera une valeur précédemment obtenue la suite entrera dans un cycle. Néanmoins, les deux suites vont le parcourir à des vitesses différentes. Ainsi, l'une va rattraper l'autre et donc il existe un rang particulier où les termes des deux suites vont être égaux.

Étant donné que $f(x_i) = h^{a_i}g^{b_i}$ et que $h = g^x$, l'équation $x_i = x_j$ équivaut à :

$$g^{xa_i}g^{b_i} = g^{xa_j}g^{b_j} \Leftrightarrow g^{x(a_i-a_j)} = g^{(b_j-b_i)}.$$

L'égalité entre les exposants permet de retrouver la valeur de x, solution du logarithme discret si $a_i - a_j$ est premier avec n. Sinon, il faut recommencer l'algorithme avec une valeur initiale de notre suite différente. L'étude faite sur la probabilité d'être inversible (*cf* section 2.3.2) nous permet heuristiquement d'estimer que cette probabilité n'est pas trop faible. Par ailleurs, si le groupe choisi est d'ordre n premier, cette probabilité est quasi-certaine. Ainsi, la solution du problème du logarithme discret est :

$$x = (b_j - b_i)(a_i - a_j)^{-1}. \tag{2.7}$$

2.3.4 Complexité en temps et en mémoire

Le principal avantage de cet algorithme est sa faible consommation en mémoire. En effet, sa complexité en mémoire est constante (en $\mathcal{O}(1)$) puisque seuls les deux termes comparés sont stockés dans l'ordinateur.

La complexité en temps de l'algorithme, notée $\mathcal{C}_t$, est déterminée par le nombre de comparaisons à effectuer pour trouver des collisions. La démonstration de la complexité est difficile dans le cas de l'algorithme Rho de Pollard, néanmoins, nous pouvons faire une analogie avec celle du paradoxe des anniversaires et en déduire ainsi la complexité en temps.

L'algorithme Rho de Pollard est un algorithme probabiliste : le nombre de comparaisons peut être très élevé si nous n'avons pas de chance. Cependant, la démonstration précédente montre qu'il y a 99% de chances d'obtenir une égalité après $3\sqrt{n}$ comparaisons. Nous pouvons donc en conclure que la complexité en temps de l'algorithme est en moyenne de l'ordre de $\mathcal{O}\left(\sqrt{-2n\ \log(1-\alpha)}\right)$ — $\alpha \in [0, 1[$ est la probabilité d'avoir une égalité — soit $\mathcal{O}(\sqrt{n})$.

Pour résumer, le temps d'exécution moyen de l'attaque est de l'ordre de $\mathcal{O}\left(\sqrt{-2n\ \log(1-\alpha)}\right)$. Néanmoins, si nous n'avons pas de chance, ce temps peut tendre vers l'infini. Il est cependant raisonnable de se dire qu'il y a 50% de chance d'avoir une collision, soit pour $\alpha = 0,5$. Alors $k \geqslant \sqrt{1,39n} \simeq \sqrt{n}$.

Ainsi, nous pouvons espérer que l'algorithme affiche un résultat au bout de $\sqrt{n}$ comparaison. Si cela n'est pas le cas, il est préférable de relancer l'algorithme avec des valeurs initiales différentes.

10. Une fonction pseudo-aléatoire sera ici considérée comme une fonction ayant un comportement similaire à celui d'une fonction aléatoire. L'étude de ce type de fonction ne sera pas réalisée ici.

2.4 Réduction

En plus des algorithmes de résolution détaillés précédemment, il est possible de réduire la résolution du problème du logarithme discret défini dans un groupe cyclique G d'ordre n à la résolution de plusieurs sous-problèmes définis dans des groupes de cardinaux moindres. Cette division qui permet de simplifier le problème est possible grâce au théorème 11 sur les isomorphismes.

Dans toute cette partie, G est un groupe cyclique d'ordre n engendré par un élément g. Nous allons étudier différentes réductions du problème en fonction des diverses décompositions possibles de n.

2.4.1 Décomposition en produit d'éléments premiers

Supposons d'abord que n peut s'écrire sous la forme $n = ab$ avec $a, b \in \mathbb{Z}$ premiers entre eux. Soit $h \in G$, pour rappel nous cherchons x tel que $g^x = h$. En posant $g_a = g^b$ d'ordre a et $g_b = g^a$, nous avons $g^n = g^{ab} = (g^a)^b = \left(g^b\right)^a$. De même, nous posons $h_a = h^b$ et $h_b = h^a$. Comme $g^x = h$, alors :

$$\begin{cases} h_a = {g_a}^x = g^{bx} \\ h_b = {g_b}^x = g^{ax}. \end{cases}$$

Cela revient donc à étudier deux nouveaux problèmes du logarithme discret où il faut déterminer $x_a \equiv bx \mod a$ et $x_b \equiv ax \mod b$:

$$\begin{cases} h_a = g^{bx_a} \\ h_b = g^{ax_b}. \end{cases}$$

Ainsi, nous obtenons deux sous-problèmes du logarithme discret à résoudre dans des groupes d'ordre inférieur.

Par hypothèse a et b sont premiers entre eux, d'après l'identité de Bézout, il existe $u, v \in \mathbb{Z}$ tels que $au + bv = 1$. Nous pouvons donc en déduire :

$$h = h^1 = h^{au+bv} = (h^a)^u (h^b)^v = (g^{ax_b})^u (g^{bx_a})^v = g^{aux_b + bvx_a}.$$

Ainsi, le problème est résolu et la solution est $x = aux_b + bvx_a \mod n$.

2.4.2 Décomposition en puissance d'éléments premiers

Supposons que n soit la puissance d'un nombre premier tel que $n = p^e$. Soit $x = p^{e-1}q + r$ (division euclidienne de x par p^{e-1}). L'élément h est alors égal à $h = g^{p^{e-1}q+r}$. Comme $g^n = g^{p^e} = 1$, alors $h^p = g^{p^e} g^{pr} = (g^p)^r$. L'élément g^p est d'ordre p^{e-1}. Suite à la résolution de ce problème d'ordre $\frac{n}{p}$ du logarithme discret, nous trouvons r.

En partant de $h = g^{p^{e-1}q+r}$, nous en déduisons :

$$hg^{-r} = g^{p^{e-1}q} = (g^{p^{e-1}})^q.$$

La résolution de ce problème du logarithme discret d'ordre p permet de trouver q. Ensuite, x est retrouvé à partir de q et de r puisque $x = p^{e-1}q + r$.

La factorisation de n peut être difficile. Cependant, lors d'utilisations en cryptographie tous les utlisateurs doivent pouvoir connaître sa factorisation ; c'est pourquoi il est nécessaire de choisir un n facile à factoriser.

2.5 Implémentations

L'implémentation des algorithmes a été réalisée avec le langage Python3.

2.5.1 Influence du groupe d'application

Étudions la difficulté de résolution dans le groupe additif $G = (\mathbb{Z}/n\mathbb{Z}, +)$. Considèrons g un générateur de G avec $\#G = n$, d'après la proposition (9) $\text{pgcd}(g, n) = 1$. Le problème du logarithme discret consiste alors à trouver x tel que $h \equiv gx \mod n$.

Si $g = 1$, la solution est triviale puisque alors $h \equiv x \mod n$. Sinon, la solution s'obtient simplement en calculant l'inverse modulaire de g grâce au théorème 5 (de Bézout) et la solution x est donnée par le calcul de $hg^{-1} \mod n$.

La résolution de l'équation de Bézout peut se faire avec l'algorithme d'Euclide étendu qui a une complexité en temps en $\mathcal{O}(\log n)$. En effet, le théorème de Lamé nous indique que le nombre d'opérations de l'algorithme d'Euclide est majoré par $\lfloor \log_\varphi b \rfloor + 1$ d'où cette complexité. Ainsi, la complexité de résolution est polynomiale et donc ce groupe n'est pas du tout fiable pour développer un cryptosystème sécurisé basé sur le problème du logarithme discret.

Nous avons donc choisi, pour l'étude de la résolution du problème du logarithme discret et l'implémentation des algorithmes corespondants, de nous placer dans des groupes $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$. En effet, il n'existe pas aujourd'hui d'algorithme en temps polynomial pour résoudre le logarithme discret dans ces groupes [11]. De plus, le protocole d'échange de clés Diffie-Hellman était originellement utilisé dans ces groupes [12].

2.5.2 Choix d'implémentation

Lors de la conception de l'algorithme Baby-Step Giant-Step, nous avons choisi d'effectuer le tri des listes *Baby-Step* et *Giant-Step* par insertion dichotomique, qui nous semblait le plus optimisé dans notre cas. Nous n'avons cependant pas eu besoin d'implémenter la méthode consistant à générer les listes pour des valeurs du couple (y, z) inférieures à $\frac{m}{2}$ (*cf* section 2.2.2). En effet, lors de nos mesures le temps devenait problématique avant d'avoir pu utiliser toute la mémoire.

Lors de la conception de l'algorithme Rho de Pollard nous avons dû choisir une fonction pseudo-aléatoire. Nous avons décidé d'utiliser la fonction suivante [13] :

$$f(\omega) = \begin{cases} h\omega & \text{si } \omega \equiv 1 \mod 3 \\ \omega^2 & \text{si } \omega \equiv 0 \mod 3 \\ gx & \text{si } \omega \equiv 2 \mod 3 \end{cases} \tag{2.8}$$

avec $\omega = g^a h^b$, où a et b sont tels que :

$$f(a, b) = \begin{cases} (a, b+1) & \text{si } \omega \equiv 1 \mod 3 \\ (2a, 2b) & \text{si } \omega \equiv 0 \mod 3 \\ (a+1, b) & \text{si } \omega \equiv 2 \mod 3. \end{cases} \tag{2.9}$$

Ici, les valeurs des couples (a, b) sont toujours entendues comme modulo n.

Nous avons été obligés d'utiliser la réduction (*cf* partie 2.4) afin de traiter les groupes d'ordre pair en séparant le problème en un groupe d'ordre impair et un autre formé de puissances de 2. En effet, pour ce dernier, la fonction pseudo-aléatoire choisie nous a posés problème puisque dans les groupes pairs, il faut utiliser l'inverse modulaire pour résoudre le problème. Cependant, la condition d'inversibilité a dans ce cas moins de chances d'être vérifiée et donc l'algorithme ne donnait que très rarement la solution.

11. Delaunay, *Le « problème du logarithme discret » en cryptographie*.
12. Diffie et Hellman, « New directions in cryptography ».
13. Pomerance, « Elementary thoughts on discrete logarithms ».

Nous avons eu recours à l'utilisation de certaines méthodes arithmétiques utilisées dans chacun de nos algorithmes telles que l'inversion modulaire, l'exponentiation rapide, le calcul d'un pgcd et la résolution de l'identité de Bézout. Nous avons implémenté [14] les algorithmes correspondants, ainsi que l'algorithme exhaustif, Baby-Step Giant-Step et Rho de Pollard.

2.5.3 Temps de calculs et espace mémoire

Une fois nos algorithmes implémentés nous les avons testés sur différents problèmes. Le calcul expérimental des complexités de nos algorithmes a été réalisé sur un ordinateur possédant les caractéristiques suivantes :

— OS : Windows 10 version 1709

— RAM : 7,89GB

— CPU : Intel(R) Core(TM) i5-6300HQ CPU @ 2,30GHz.

Nous avons effectué des tests pour la résolution de problèmes du logarithme discret dans des groupes de taille croissante, allant d'environ 10 éléments à 10^{40} éléments. Cependant nous n'avons pas réussi à résoudre le problème dans les plus grands groupes. Nous avions au préalable décidé de considérer le problème comme non résoluble si l'algorithme ne donnait aucune réponse après six heures de calculs. En effet, expérimentalement il est difficile d'effectuer une résolution du problème sur une période supérieure à cette durée.

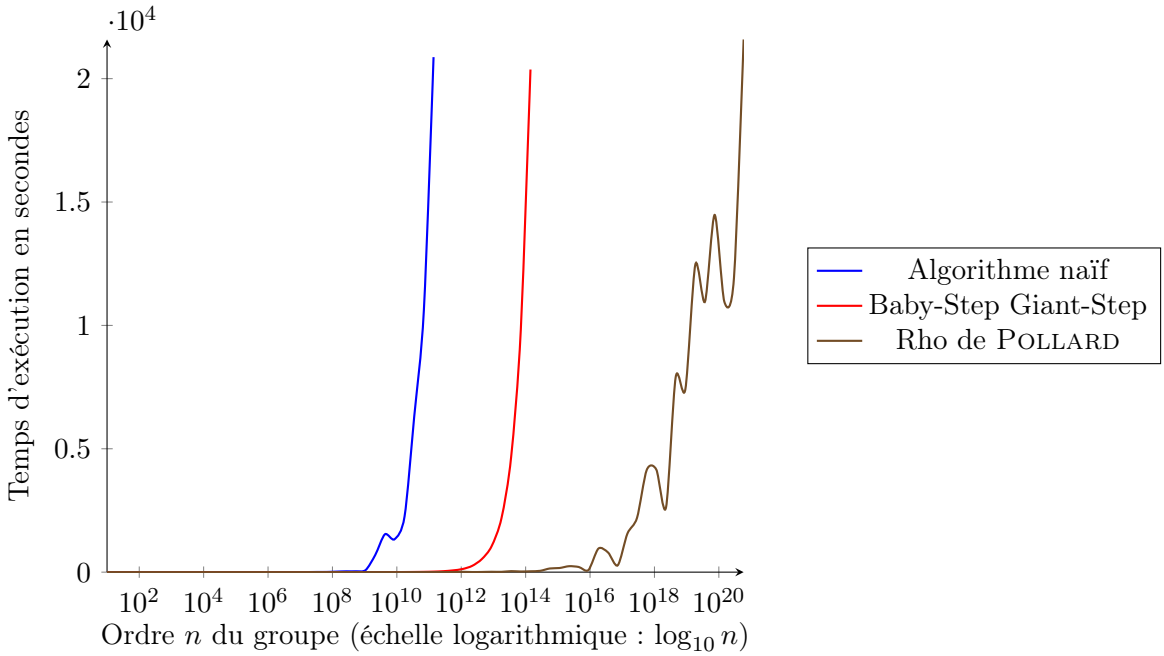

Figure 2.1 – Temps d'exécution des trois algorithmes implémentés en fonction du cardinal du groupe.

Les figures 2.1 et 2.2 représentent respectivement les courbes de nos résultats de mesure du temps d'exécution et de la mémoire utilisée [15]. Nous pouvons remarquer que le temps nécessaire à l'obtention d'une solution dans les différents groupes testés est proportionnelle à la taille du groupe. Pour les algorithmes Baby-Step Giant-Step et naïfs, les courbes sont régulières et croissantes. Cela vient du fait que ces algorithmes sont déterministes. En revanche, Rho de Pollard est un algorithme probabiliste, ce qui se traduit sur le graphique par une courbe irrégulière et l'apparition de "pics". Du fait de cette caractéristique de l'algorithme, nous avons effectué plusieurs fois les mêmes résolutions et fait une moyenne des temps obtenus.

14. Les détails concernant le code des algorithmes sont disponibles dans l'annexe E.
15. Les résultats détaillés de nos expériences sont disponibles dans l'annexe C.

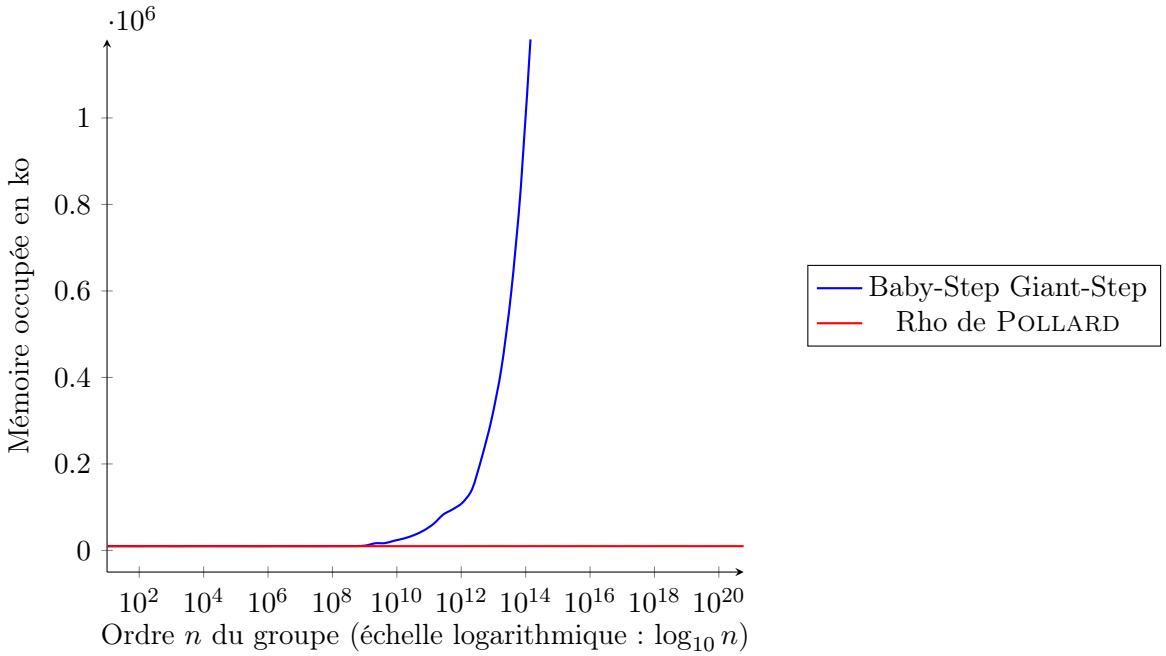

Figure 2.2 – Mémoire utilisée par les deux algorithmes implémentés en fonction du cardinal du groupe.

Pour les premiers groupes, nous avons calculé cette moyenne avec cent valeurs, néanmoins avec la croissance de la taille des groupes et donc du temps de résolution, nous avons réduit le nombre de valeurs. Pour les derniers groupes, la moyenne a été calculée grâce à environ cinq tests. Cela peut expliquer, en partie du moins, les irrégularités observées. Lors de nos tests, les algorithmes ont toujours déterminé une solution, c'est-à-dire qu'ils n'ont jamais été arrêtés à cause d'une erreur. De plus, cette solution s'est trouvé être toujours exacte.

Nous avons approximé les courbes des figures 2.1 et 2.2 avec le logiciel GeoGebra afin d'en déterminer une équation. Pour cela, nous avons d'abord fait figurer les différentes mesures sous forme de points de coordonnées *(ordre, temps ou mémoire)*. Ensuite, nous avons déterminé au jugé une courbe qui pourrait convenir aux constantes près. Enfin, nous avons fait varier ces constantes pour trouver celles qui convenaient le mieux et ce faisant, nous avons pu estimer leurs incertitudes.

Ces équations (disponibles dans la table 2.1) sont — aux constantes près (additives et multiplicatives) — en adéquation avec les complexités théoriques déterminées précédemment, excepté pour la complexité en temps de Baby-Step Giant-Step. Cette irrégularité nous a amené à effectuer des mesures supplémentaires lors de la modification des listes en Python3. Nous avons constaté que le temps de génération d'une liste en fonction de sa taille évoluait expérimentalement en $\mathcal{O}(n^2)$ contre une évolution théorique en $\mathcal{O}(n)$. Nous pouvons expliquer ainsi une partie de la différence d'efficacité entre Rho de Pollard et Baby-Step Giant-Step. En effet, lors de nos résolutions du logarithme discret, la génération des listes prenait la majeure partie du temps — environ 95% — et les comparaisons étaient assez rapides.

		Complexité	
		en temps (secondes)	en mémoire (ko)
Algorithme	exhaustif	$(16 \pm 1)10^{-8} \cdot x$	9550 ± 50
	Baby-Step Giant-Step	$(14 \pm 1)10^{-11} \cdot x$	$(0.1 \pm 0.001)\sqrt{x} + (9854 \pm 50)$
	Rho de Pollard	$(30 \pm 1)10^{-7} \cdot \sqrt{x}$	9848 ± 50

Table 2.1 – Équations de tendances des courbes de temps d'exécution et d'espace mémoire utilisé.

3. Application du problème du logarithme discret en cryptographie

Nous avons vu que le problème du logarithme discret devient difficile à résoudre dès lors que l'ordre du groupe devient grand. Il est donc intéressant d'exploiter ce problème pour construire des protocoles de chiffrement. Nous présenterons dans ce chapitre deux de ces cryptosystèmes qui furent parmi les premiers protocoles à clé publique inventés. Les paramètres d'implémentations — dont l'ordre du groupe dans lequel nous travaillons — doivent cependant être choisis en fonction du niveau de sécurité attendu.

3.1 Protocole d'échange de clés Diffie-Hellman

En 1976 dans leur article « New directions in cryptography » qui posera le principe de la cryptographie à clé publique, Whitfield Diffie et Martin Hellman présentent un protocole d'échange de clés qui résout un des problèmes majeurs de la cryptographie symétrique. En effet, il permet d'éviter le recours à un canal de transmission d'information sécurisé pour échanger une clé commune qui servira au chiffrement et au déchiffrement du message.

3.1.1 Fonctionnement du protocole

Ce protocole d'échange de clés, comme tout système cryptographique à clé publique, repose sur des informations connues de tous — ici un groupe — et des informations secrètes — un élément du groupe — pour chaque partie prenante du système.

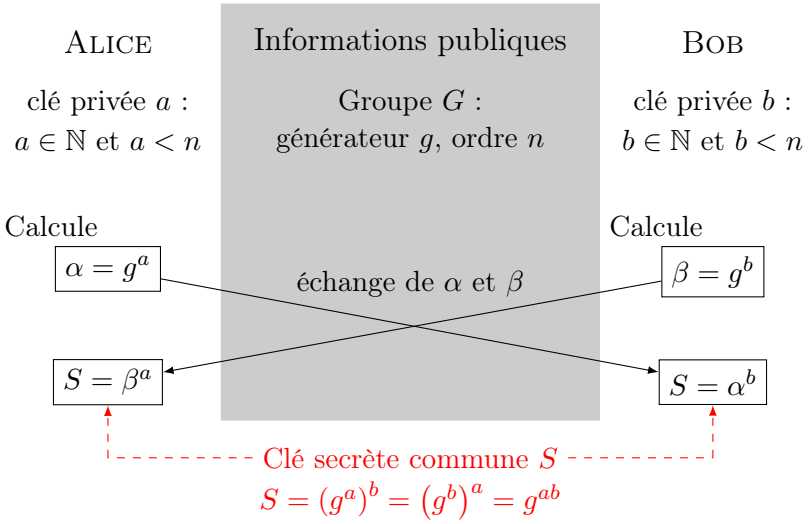

Figure 3.1 – Protocole d'échange de clés Diffie-Hellman dans un groupe cyclique G de générateur g et d'ordre n connus.

Le fonctionnement du protocole est décrit dans la figure 3.1. Le groupe cyclique G est connu de tous ainsi qu'un de ses générateurs g et son cardinal n. Chacune des deux personnes voulant échanger choisi une clé secrète — notée respectivement a ou b — qui est un entier positif strictement inférieur à n. Elles calculent respectivement $\alpha = g^a$ ou $\beta = g^b$. Elles s'échangent ensuite le résultat de ce calcul. Une fois échangée, chaque partie obtient la clé secrète commune S en élevant le résultat reçu à la puissance correspondant à sa clé privée : $S = \alpha^b = \beta^a = g^{ab}$.

Tout espion voulant recomposer la clé secrète obtenue ne connaît que g, g^a et g^b. Ce protocole repose ainsi sur le problème suivant.

Définition 4. Soit g un générateur d'un groupe cyclique. Le problème de Diffie-Hellman consiste à déterminer g^{ab} en connaissant seulement les résultats g^a et g^b.

Résoudre le problème du logarithme discret permet de résoudre le problème de Diffie-Hellman. Ce dernier est donc au maximum de la même difficulté que le problème du logarithme discret. Par conséquent, le protocole doit être utilisé dans des groupes où le logarithme discret est difficile à résoudre. Cependant, il n'a pas encore été prouvé que la difficulté des deux problèmes est identique [1].

Il est aussi possible que le groupe G ne soit pas décidé à l'avance et seulement par la personne qui initie l'échange. Elle enverra alors le triplet (g, n, α). Cependant, cette variante n'a pas d'impact significatif sur la sécurité de l'algorithme. En effet, le protocole est implémenté dans des groupes où le problème de Diffie-Hellman est difficile : par exemple il faut des milliers d'années pour le résoudre et ce avec une grande puissance de calcul. Connaître le groupe à l'avance et donc pouvoir commencer à calculer toutes les puissances de g avant l'échange n'aura donc pas de réel impact.

3.1.2 Sécurité du protocole d'échange de clés

La sécurité de ce protocole repose sur la difficulté de résolution du problème de Diffie-Hellman mais aussi sur la conception et l'implémentation du protocole en lui même. Nous n'étudierons pas ici ce sujet en profondeur.

Cependant, il est intéressant de noter que le protocole d'échange de clés de Diffie-Hellman ne permet pas de s'assurer de l'authenticité de la clé reçue. Plus particulièrement, ce protocole est vulnérable à l'attaque dite *man-in-the-middle* [2] durant laquelle un espion intercepte la clé envoyée par Alice et celle envoyée par Bob et les remplace par d'autres. Ainsi, l'espion obtient une clé partagée avec Alice en lui faisant croire qu'il est Bob et une autre avec Bob en lui faisant croire qu'il est Alice. Une solution simple à cette attaque consiste à coupler l'envoi de α et β à un protocole d'authentification type RSA [3].

3.2 Cryptosystème de ElGamal

En 1985 [4], Taher ElGamal propose un protocole de cryptographie à clé publique permettant l'échange de messages. Pour cela, il base sa proposition sur une amélioration du protocole d'échange de clés de Diffie-Hellman.

Le fonctionnement de ce système cryptographique dans un groupe cyclique quelconque G de générateur g et d'ordre n est présenté dans la figure 3.2.

Pour envoyer un message m à Bob, Alice choisit un élément k du groupe puis calcule les deux nombres $C_1 = g^k$ et $C_2 = m\beta^k$ où β est la clé publique de Bob. Cette dernière a été générée par Bob au préalable de la même manière que dans le protocole de Diffie-Hellman. Alice transmet ensuite le

1. Aumasson, *Serious cryptography : a practical introduction to modern encryption*, page 204.
2. Ibid., figure 11.3, page 210.
3. Ibid., figure 11.4, pages 210–211.
4. ElGamal, « A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms ».

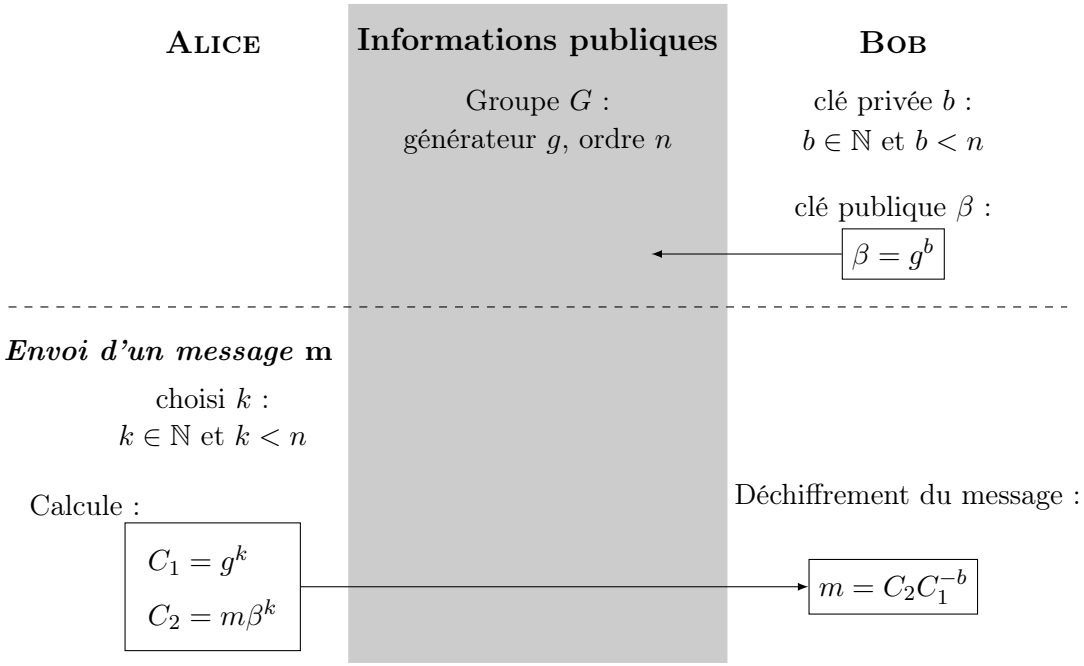

Figure 3.2 – Envoi d'un message par Alice à Bob grâce au cryptosystème de ElGamal dans un groupe cyclique G de générateur g et d'ordre n connus.

couple (C_1, C_2) à Bob. Ce dernier peut déchiffrer le mesage en calculant $C_2 C_1^{-b}$ où b est sa clé privée. En effet :

$$C_2 C_1^{-b} = m\beta^k \left(g^k\right)^{-b} = m g^{bk} g^{-bk} = m.$$

Si une personne extérieure veut essayer de déchiffer le message, elle doit trouver la valeur de g^{bk} or elle ne connaît que le générateur g ainsi que les nombres g^k et g^b. La sécurité de ce protocole repose donc sur le problème de Diffie-Hellman et par conséquent, sur la sécurité du protocole du même nom.

3.3 Implémentations

L'implémentation des protocoles a été réalisée avec le langage Python3 pour des groupes de type $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$.

3.3.1 Création d'un groupe $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$

Avant d'implémenter les protocoles, nous avons besoin de générer un groupe $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$. De plus, nous voulons le définir tel que le nombre premier p satisfait la relation $p - 1 = 2N$ avec N premier aussi. Nous allons déterminer une méthode pour construire un tel groupe ainsi que sa complexité.

Le théorème des nombres premiers[5] nous permet de poser l'équivalence $\Pi(X) = \#\{p \leqslant X : p \text{ est premier}\} \sim \frac{1}{\log X}$. Ainsi, en tirant aléatoirement un entier p entre 0 et X (pris grand), la probabilité pour que p soit premier est d'environ $\frac{1}{\log X}$.

Déterminons la probabilité que p et $N = \frac{p-1}{2}$ soient premiers, ce qui constituera la condition de validité du choix de l'entier p. Pour simplifier les calculs, nous allons supposer que la probabilité que p soit premier est indépendante de celle que N le soit aussi. Comme $p \in [\![0, X]\!]$ alors $N \in [\![0, \frac{X}{2}]\!]$. De plus, nous avons l'estimation $\mathcal{P}(p \text{ valide}) \simeq \frac{1}{(\log X)^2} = p_1$ car $\frac{1}{\log \frac{X}{2}} \simeq \frac{1}{\log X}$.

5. Nous ne chercherons pas à démontrer ce théorème dans ce mémoire. Nous l'admettons en nous basant sur Hardy et Wright, *Introduction à la théorie des nombres*, théorème 6, page 11.

Nous pouvons en déduire que la probabilité de tirer au moins un p valide parmi k tirages indépendants est : $\mathcal{P}(\text{au moins un } p \text{ valide parmi } k) = 1-(1-p_1)^k$. Si nous voulons que cette probabilité soit d'au moins α, avec $\alpha[0,1[$, alors le nombre de tirages minimal nécesaire est donné par :

$$1-(1-p_1)^k \geqslant \alpha \Leftrightarrow (1-p_1)^k \leqslant 1-\alpha \Leftrightarrow k\log(1-p_1) \leqslant \log(1-\alpha) \Leftrightarrow k \geqslant \frac{\log(1-\alpha)}{\log(1-p_1)}.$$

Ainsi, quand la probabilité p_1 tend vers 0, alors $k \simeq \frac{-\log(1-\alpha)}{p_1}$ et $k \simeq (\log X)^2 \log\left(\frac{1}{1-\alpha}\right)$.

Finalement le nombre de tirage nécessaire pour initialiser le groupe est de complexité polynomiale. Par exemple, en choisissant une probabilité de 99% ($\alpha = 0,99$), le nombre de tirages nécessaires afin de tirer un entier p premier tel que $N = \frac{p-1}{2}$ soit premier est de $k \simeq 4,6(\log X)^2$.

Ensuite, connaisssant p, nous avons choisi de prendre $n = p-1$ soit $n = 2N$. Ce choix est pratique puisque l'entier n est alors facile à décomposer en facteurs premiers. Enfin, il reste à déterminer un entier g qui engendre $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$ en utilisant le théorème 10. En l'appliquant pour $n = 2N$, il faut montrer que $g^N \not\equiv 1 \mod p$ *et* que $g^2 \not\equiv 1 \mod p$, pour que g soit bien le générateur recherché.

Démontrons à présent que trouver un tel générateur est de complexité polynomiale. Tout d'abord, montrons que l'assertion « g engendre $(\mathbb{Z}/n\mathbb{Z}, +)$ » équivaut à celle « g est inversible dans $(\mathbb{Z}/n\mathbb{Z}, +)$ » (*id est* il existe $u \in \mathbb{Z}/n\mathbb{Z}$ tel que $ug = 1$).

Supposons que g est un générateur de $(\mathbb{Z}/n\mathbb{Z}, +)$. Alors $1 \in \langle g \rangle$. Cela implique qu'il existe un $u \in \mathbb{Z}$ tel que $ug = 1$. Réciproquement, supposons que g est inversible dans $(\mathbb{Z}/n\mathbb{Z}, +)$. Par conséquent, pour tout $k \in \mathbb{Z}/n\mathbb{Z}$, $k = k(ug) = (ku)g$ et donc $k \in \langle g \rangle$.

Ensuite, d'après le théorème 11 , nous savons que $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$ est isomorphe à $(\mathbb{Z}/n\mathbb{Z}, +)$. Donc la complexité pour trouver un générateur est équivalente à la probabilité d'être inversible (section 2.3.2). Sachant que $n = 2N$, avec 2 et N premiers, alors la probabilité d'avoir $g^2 \not\equiv 1 \mod p$ est égale à $1 - \frac{1}{2} = \frac{1}{2}$ et celle d'avoir $g^N \not\equiv 1 \mod p$ vaut $1 - \frac{1}{N}$. Nous en déduisons donc que la probabilité pour que g soit un générateur est de $\frac{1}{2}(1 - \frac{1}{N})$.

Ainsi, trouver un générateur est équivalent à vérifier s'il est inversible. Calculer son inverse peut se faire à l'aide de l'algorithme d'Euclide étendu, qui d'après le théorème 6 (Lamé) peut s'effectuer en une complexité polynomiale.

Finalement, générer le triplet (g, p, n) qui définit le groupe dans lequel nous allons définir des cryptosystèmes a une complexité polynomiale.

3.3.2 Exponentiation rapide

Étant donné trois entiers (g, x, p), l'exponentiation rapide est une méthode permettant de calculer rapidement l'entier $g^x \mod p$. En effet, si d'un côté un cryptosystème doit être difficile à casser, de l'autre, l'utilisateur de celui-ci doit pouvoir facilement et rapidement être en mesure d'en changer les paramètres d'entrée. L'utilité de cette méthode dans notre projet est évidente puisque nous devons souvent calculer de tels nombres.

Le fonctionnement de l'algorithme est explicité dans l'algorithme 1.

Déterminons donc la complexité de cet algorithme. Pour cela nous définissons les variables suivantes :

- k le nombre d'exposants qu'il reste à calculer (au début, $k = x$ et à la fin, $k = 0$) ;
- $\gamma(x)$ le nombre de caractères dans l'écriture binaire de x ;
- $\delta(x)$ le nombre de "1" dans l'écriture binaire de x ;
- m le nombre de multiplications effectuées dans l'algorithme.

Initialement, nous avons $m + \gamma(k) + \delta(k) = 0 + \gamma(x) + \delta(x)$.

Algorithme 1 Calcule $r \equiv g^x \mod p$

Entrées : $g, x, p \in \mathbb{N}^*$

```
m ← x
a ← g
r ← 1
Tant que m ≠ 0 faire
  Si x%2 = 0 (% est le reste de la division euclidienne) faire
    m ← m//2
    a ← (a × a)%p
  Sinon
    m ← m − 1
    r ← (a × r)%p
  fin du « Si »
fin du « Tant que »
Retourner r
```

Sortie : $g^x \mod p$

Quand k est pair, alors $\gamma(k/2) = \gamma(k) - 1$ et $\delta(k/2) = \delta(k)$. En effet, en binaire lorsque l'on divise par deux un nombre pair cela équivaut à enlever le zéro final et donc à décaler tout les chiffres vers la droite. Quand k est impair, alors $\gamma(k-1) = \gamma(k)$ et $\delta(k-1) = \delta(k) - 1$.

Au final, $k = 0$, $\gamma(0) = 1$ et $\delta(0) = 0$.

$$m + \gamma(0) + \delta(0) = 0 + \gamma(x) + \delta(x) \Leftrightarrow m = \gamma(x) + \delta(x) - 1.$$

Nous pouvons à présent étudier la complexité de l'algorithme :

$$\forall x, \quad \delta(x) \leqslant \gamma(x) \Rightarrow m \leqslant 2\gamma(x) - 1 = \mathcal{O}(\gamma(x))$$

or $\gamma(x) = \lfloor \log_2 x \rfloor$, d'où $m = \mathcal{O}(\log x)$. Ainsi, la complexité de l'algorithme d'exponentiation rapide est polynomiale.

3.3.3 Envoi d'un message avec le cryptosystème de ElGamal

Le principe du cryptosystème de ElGamal présenté précédemment contient différentes équations permettant de chiffrer le message. Ce dernier y est représenté par un nombre. Lors de nos implémentations nous avons dû transformer notre message constitué de lettres en un nombre m. Nous présenterons ici la méthode utilisée.

Une méthode toute simple pour chiffrer un message consiste à associer un nombre à chaque lettre de l'alphabet et caractères utilisés, qui serait ensuite encodé grâce au protocole de chiffrement. La principale faiblesse de cette méthode réside dans le fait que chaque lettre sera chiffrée de la même manière. Ainsi, le message pourrait être découvert par un attaquant en utilisant une analyse de fréquence d'une part. Un attaquant pourrait d'autre part déchiffrer une unique fois la lettre associée à un nombre qui apparait de nombreuses fois et donner ce résultat à chaque nouvelle apparition du nombre en question. Cette méthode permet d'être plus rapide dans la découverte du message.

Pour pallier à ce problème, il est possible de se baser sur la méthode du chiffrement par bloc. Elle consiste à découper un texte clair en blocs de même longueur, puis à les chiffrer séparément. La taille e des blocs doit satisfaire la relation :

$$\frac{p}{n} < n^e \leqslant p \tag{3.1}$$

où p est défini par le groupe utilisé $G = (\mathbb{Z}/p\mathbb{Z}, \times)$, $n = \#B$ avec B la base des caractères utilisés. Une telle relation doit permettre à e d'exister. De plus, l'unicité de la solution e est essentielle. En

effet, quand Bob envoie le message chiffré, la personne qui le déchiffre doit connaître exactement la taille des blocs (qui n'est pas comprise dans le message) et donc ne pas avoir à choisir parmi plusieurs possibilités. Ainsi, nous pouvons montrer le théorème suivant.

Théorème 12. *Soient $G = ((\mathbb{Z}/p\mathbb{Z})^\times, \times)$, B un ensemble muni d'au moins deux éléments, tel que $p \geqslant \#B$. On pose $n = \#B$. Alors il existe un unique $e \in \mathbb{N}^*$ tel que $\frac{p}{n} < n^e \leqslant p$.*

Démonstration. Déterminons les solutions de $n^x \leqslant p$, avec $x \in \mathbb{N}^*$. Cette équation équivaut à :

$$x \log n \leqslant \log p \quad \Leftrightarrow \quad 0 < x < \lfloor \log_n p \rfloor + 1 \quad \Leftrightarrow \quad x \in [\![1, e]\!] \text{ avec } e = \lfloor \log_n p \rfloor.$$

Déterminons à présent les solutions de $\frac{p}{n} < n^x$, avec $x \in \mathbb{N}^*$, ce qui équivaut à

$$\log\left(\frac{p}{n}\right) < x \log n \quad \Leftrightarrow \quad \frac{\log p - \log n}{\log n} < x \quad \Leftrightarrow \quad \lfloor \log_n p \rfloor - 1 < x.$$

Les solutions de l'inégalité sont donc les $x \in \mathbb{N}^*$ tels que $x \in [\![e, +\infty[\![$ où $e = \lfloor \log_n p \rfloor$. Ainsi, si x vérifie $\frac{p}{n} < n^x \leqslant p$, alors $x = \lfloor \log_n p \rfloor$, ce qui montre l'existence et l'unicité de la solution. □

La formule que nous avons montrée présente un avantage au niveau de la sécurité puisqu'elle s'adapte à la taille du groupe et ainsi évite l'attaque par analyse de fréquence. Lorsque l'on découpe le message, si le dernier bloc n'est pas entièrement rempli par les éléments du message, il suffit de le compléter par des espaces ou un élément vide (padding).

3.3.4 Choix d'implémentation

Pour les vérifications de la primalité des nombres lors de la construction du groupe, nous avons utilisé le test de primalité de Miller-Rabin. Nous n'avons pas cherché à étudier en détail cet algorithme et nous avons simplement utilisé une implémentation disponible sur Internet [6] dans l'objectif de pouvoir vérifier rapidement si un entier est premier ou non. Cet algorithme est probabiliste mais peut nous garantir avec une grande certitude l'obtention rapide du bon résultat. Nous avons pu déterminer [7] le temps de génération de plusieurs groupes d'ordre allant de 10 à 10^{200}. Grâce au logiciel GeoGebra, nous avons approximé la courbe de la complexité en temps par la fonction $f(x) = (0,02 \pm 0,005)(\log_{10} x)^2 - (0,5 \pm 0,2) \log_{10} x$. Elle est, aux constantes près, en adéquation avec la complexité théorique que nous avions déterminée.

Lors de la conception de l'algorithme ElGamal, nous avons construit une base de 81 caractères à utiliser pour pouvoir s'envoyer des messages. Pour cela, nous disposons des 26 lettres en minuscules et en majuscules, de l'espace et de caractères spéciaux divers explicités dans le code. Nous avons effectué des mesures du temps nécessaire au chiffrement ou déchiffrement différentes tailles de message. Les résultats obtenus sont concluants [8] puisque, par exemple, pour un message de mille caractères le temps pour le chiffrer est de l'ordre d'un dixième de seconde. Le temps pour le déchiffrer — en étant l'utilisateur légitime — est certe un peu plus long mais reste du même ordre de grandeur (environ deux fois plus long) ce qui satisfait nos attentes. Par exemple, déchiffrer environ 15 caractères sans connaître la clé prendrait des jours.

L'implémentation du cryptosystème de ElGamal est constituée de deux fonctions. La première permet de chiffrer le message et la seconde de le déchiffrer. Par ailleurs, nous avons recouru à des fonctions pour convertir le message en liste de nombres et en liste de blocs de nombres comme explicité lors de l'analyse théorique ainsi que des conversions dans l'autre sens.

6. Le lien de l'algorithme est disponible dans les commentaires de notre code.
7. Le détail de nos résultats est disponible dans l'annexe D, plus précisément dans la figure D.1 et la table D.2 qui lui est associée.
8. Les résultats détaillés de nos tests sont disponibles dans la table D.1 l'annexe D.

Conclusion

À travers ce projet, nous avons vu l'importance de l'étude du problème du logarithme discret dans l'estimation du niveau de sécurité du protocole d'échange de clés de Diffie-Hellman et du cryptosystème de ElGamal. Nous avons pu constater l'importance d'effectuer une étude mathématique du problème puis dans un second temps de combiner approches mathématiques et informatiques.

Tout d'abord, la définition du problème du logarithme discret a nécessité d'introduire la notion de groupe cyclique ainsi que les théorèmes l'accompagnant. Nous avons ensuite construit les deux groupes associés à l'ensemble $\mathbb{Z}/n\mathbb{Z}$ à l'aide des congruences ainsi que du théorème de Bézout. Nous avons caractérisé certains générateurs de ces groupes. Enfin, le lien entre les différentes parties de ce chapitre a été fait en démontrant l'existence d'isomorphisme de tout groupe cyclique à $(\mathbb{Z}/n\mathbb{Z}, +)$.

L'étude des algorithmes de résolution Baby-Step Giant-Step et Rho de Pollard dans des groupes cycliques quelconques a mis en évidence leur complexité exponentielle en temps. Pour cela, nous avons dû définir différentes classes de complexité. Rho de Pollard étant un algorithme probabiliste, nous avons dû exploiter le paradoxe probabiliste sur lequel il est basé pour déterminer sa complexité moyenne. Nous avons ensuite implémenté ces algorithmes pour des groupes $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$ en Python3. La plupart des mesures de temps d'exécution et de mémoire obtenues étaient en accord avec la théorie à l'exception du temps pour Baby-Step Giant-Step. Nous avons alors réalisé d'autres mesures pour comprendre que le problème venait de la manipulation des listes dans ce langage. Si l'on considère comme seules attaques possibles nos implémentations des algorithmes de résolution, nous avons pu déterminer grâce à nos mesures que des protocoles de cryptographie basés sur le logarithme discret seront sécurisés pour des groupes dont l'ordre est supérieur à 10^{30}.

Dans notre dernier chapitre, nous avons étudié deux protocoles de cryptographie à clé publique basés sur le problème du logarithme discret : le protocole d'échange de clés de Diffie-Hellman et le cryptosystème de ElGamal. Le premier permet d'obtenir une clé commune secrète entre deux personnes pour qu'elles puissent ensuite utiliser un protocole de chiffrement symétrique. Le second permet directement un échange de message. La sécurité des deux systèmes repose sur le problème de Diffie-Hellman qui est résolu si l'on résout le problème du logarithme discret. Nous avons implémenté ces deux cryptosystèmes avec le langage Python3. Pour cela nous avons dû générer des groupes $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$. De plus, le chiffrement du message dans le cryptosystème de ElGamal a été réfléchi pour éviter qu'une analyse de fréquence permette son déchiffrement.

Nous avons étudié le problème du logarithme discret dans des groupes cycliques et déterminé des paramètres dans le groupe $((\mathbb{Z}/n\mathbb{Z})^\times, \times)$. Ces derniers sont inférieurs aux standards de sécurité conseillé. En effet, nous avons traité uniquement des algorithmes de résolution de complexité exponentielle alors qu'il en existe des plus efficaces — mais beaucoup plus compliqués — de complexité sous-exponentielle. De plus nos implémentations n'étaient pas optimales. Enfin, aujourd'hui les protocoles de cryptographie présentés sont toujours utilisés. Cependant, ils sont désormais implémentés en utilisant des courbes elliptiques. Le principal intérêt réside dans le fait que les algorithmes de résolution du problème du logarithme discret sont de complexité exponentielle et non plus sous-exponentielle.

Références bibliographiques

Livres

Aumasson, Jean-Philippe. *Serious cryptography : a practical introduction to modern encryption*. San Francisco : No Starch Press, 2017. 282 p. ISBN : 978-1-59327-826-7.

Hardy, G. H et Edward Maitland Wright. *Introduction à la théorie des nombres*. Paris ; Heidelberg : Vuibert ; Springer, 2007. ISBN : 978-2-7117-7168-4.

Menezes, A. J., Paul C. Van Oorschot et Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications. Boca Raton : CRC Press, 1997. 780 p. ISBN : 978-0-8493-8523-0.

Meunier, Pierre. *Cours d'algèbre et d'algorithmique : applications à la cryptologie du RSA et logarithme discret*. Toulouse : Cépaduès-éditions, 2014. ISBN : 978-2-36493-097-1.

Wassef, Pierre. *Arithmétique : application aux codes correcteurs et à la cryptographie : cours & 122 exercices corrigés : licence de mathématiques*. Paris : Vuibert, 2009. ISBN : 978-2-7117-2083-5.

Chapitres de livres

ElGamal, Taher. « A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms ». In : *Advances in Cryptology*. Sous la dir. de George Robert Blakley et David Chaum. T. 196. Berlin, Heidelberg : Springer Berlin Heidelberg, 1985, p. 10–18. DOI : 10.1007/3-540-39568-7_2. URL : http://link.springer.com/10.1007/3-540-39568-7_2.

Articles de recherche

Diffie, Whitfield et Martin Hellman. « New directions in cryptography ». In : *IEEE Transactions on Information Theory* 22.6 (nov. 1976), p. 644–654. DOI : 10.1109/TIT.1976.1055638. URL : http://ieeexplore.ieee.org/document/1055638/.

Pollard, John M. « Monte Carlo Methods for Index Computation (mod p) ». In : *Mathematics of Computation* 32.143 (juil. 1978), p. 918–924. DOI : 10.2307/2006496. URL : http://www.jstor.org/stable/2006496?origin=crossref.

Pomerance, Carl. « Elementary thoughts on discrete logarithms ». In : *Algorithmic Number Theory : Lattices, Number Fields, Curves and Cryptography* 44 (2008), p. 385–396. URL : http://www.msri.org/people/staff/levy/files/Book44/11carl.pdf.

Thèses

Tréguer, Félix. « Pouvoir et résistance dans l'espace public : une contre-histoire d'Internet (XVe - XXIe siècle) ». Thèse de doct. Paris : EHESS, nov. 2017. 600 p. URL : https://halshs.archives-ouvertes.fr/tel-01631122.

Sites internet

Delaunay, Christophe. *Le « problème du logarithme discret » en cryptographie*. CNRS — Images des mathématiques. 14 mar. 2015. URL : http://images.math.cnrs.fr/Le-probleme-du-logarithme-discret-en-cryptographie.html.

Tous les liens ont été vérifiés avec succès le mardi 8 mai 2018.

Annexes

A. Synthèse culturelle : la cryptographie est-elle le garant de nos libertés fondamentales à l'heure d'Internet et de la surveillance généralisée ?

En 1976, deux chercheurs de l'université de Stanford, Whitfield Diffie et Martin Hellman publient « New directions in cryptography » [1] un article qui va poser les bases de la cryptographie telle qu'elle est utilisée actuellement. La communication grâce à des ordinateurs en réseau commence à se développer, ce qui nécessite pour les deux scientifiques de développer des techniques de sécurisation des communications pour garantir, d'une part, la confidentialité des échanges, et donc la vie privée, et d'autre part l'intégrité du message reçu.

Cependant, les seuls systèmes cryptographiques connus à l'époque nécessitaient que l'émetteur et le destinataire du message chiffré aient connaissance d'une même clé secrète pour pouvoir chiffrer et déchiffrer le message. La transmission de cette clé posait un problème central dans la démocratisation du chiffrement : elle imposait par exemple de retarder la mise en place de relations commerciales [2]. Afin de résoudre ce problème, Whitfield Diffie et Martin Hellman inventent la cryptographie asymétrique, système qui repose sur deux clés : une privée, qui ne doit jamais être divulguée, et une publique, connue de tous. Si Alice veut envoyer un message à Bob, il suffit qu'elle chiffre le message avec la clé *publique* de Bob. Le message ne pourra être déchiffré qu'avec la clé *privée*. Pour chiffrer et déchiffrer le message on n'utilise pas la même clé, ce système enlève donc le problème de transmission de clés et est donc réellement utilisable à grande échelle dans les communications numériques. La création de la cryptographie asymétrique révolutionnera le domaine : elle est à la base des principaux algorithmes de chiffrement utilisés de nos jours.

Cet article sera un des actes majeurs qui mènera à la fin du monopole des agences de renseignement (notamment l'agence américaine nommée *National Security Agency* (NSA)) et des armées sur le chiffrement, au nom du droit à la vie privée [3]. Nous observons alors un conflit apparaître entre la notion de vie privée et les intérêts étatiques que doivent continuer à servir les agences de renseignement et le domaine militaire. À la lumière de cette controverse et en explorant l'exemple du chiffrement, nous étudierons les enjeux sociétaux soulevés par l'utilisation massive des technologies et des nouveaux moyens de communication numérique.

1. Diffie et Hellman, « New directions in cryptography ».

2. Pour reprendre un exemple de l'article, « il n'est pas réaliste de s'attendre à ce qu'un premier contact commercial soit reporté le temps que les clés soient échangées par un moyen physique ». (Notre traduction de « it is unrealistic to expect initial business contacts to be postponed long enough for keys to be transmitted by some physical means ».)

3. Tréguer, « Pouvoir et résistance dans l'espace public : une contre-histoire d'Internet (XVe -XXIe siècle) », sous-section 7.3.1 « Le génie cryptographique sort de sa bouteille », pp. 261–264.

L'usage de la cryptographie au service des libertés fondamentales à l'ère du numérique

Une célèbre métaphore de vulgarisation compare l'envoi d'un courriel non chiffré (en clair) à l'envoi d'une carte postale. Toute personne qui l'intercepte peut la lire mais à l'inverse, si elle se trouve dans une enveloppe scellée, elle n'est plus accessible à tous les regards. Bien que simple, cette image permet de comprendre certains enjeux du chiffrement des communications. En effet, ces dernières peuvent être interceptées à de nombreux niveaux, que ce soit par un virus, notre fournisseur de services de communication, un autre utilisateur de l'ordinateur, *etc.* Pour éviter que nos échanges soient intelligibles pour d'autres personnes que le destinataire, une des seules solutions consiste à recourir au chiffrement [4].

Les libertés fondamentales, notamment la liberté d'expression, sont intimement liées avec le droit à la vie privée [5]. En particulier à l'ère de l'omniprésence de l'informatique, la possibilité d'être surveillé — que ce soit par un État ou alors simplement par un proche, un logiciel espion, les différentes entreprises nous fournissant des logiciels, des services web ou un accès Internet — est nettement accrue. L'usage du chiffrement pour protéger nos échanges devient alors nécessaire pour s'assurer le respect de sa vie privée. La défense de ce droit fût notamment au cœur de la bataille judiciaire et politique [6] ayant amené en France à une libéralisation partielle puis totale de l'utilisation du chiffrement : en effet, depuis la publication de la loi pour la confiance dans l'économie numérique 2004 [7], l'*usage* du chiffrement est libre. Bien que régulièrement remis en cause par de nombreux politiciens, ce droit est toujours en vigueur aujourd'hui.

Comme nous l'avons évoqué précédemment, le premier usage du chiffrement auquel on pense pour protéger sa vie privée est celui de la protection des échanges. Cependant la cryptographie asymétrique a de nombreuses applications et l'une d'elle est de permettre l'anonymisation de l'internaute. Un des exemples les plus célèbre est le réseau d'anonymisation TOR qui permet d'avoir accès à Internet sans que l'on puisse identifier [8] l'ordinateur dont provient cette connexion. L'anonymat permet de lutter contre la censure quelle soit connue — dans une dictature par exemple — ou inconsciente lorsque nous nous savons surveillés. L'identification et la censure sont liées, par conséquent, nous avons démontré le lien entre anonymat et liberté d'expression [9].

Vie privée et surveillance : évolution de la controverse après les révélations d'Edward Snowden

En juin 2013, les deux journalistes Glenn Greenwald et Laura Poitras, avec l'aide de plusieurs journaux européens et américains, dévoilent des documents classifiés de la NSA fournis par Edward Snowden, employé d'un sous-traitant des renseignements américains. Ces documents révèlent différents programmes d'ampleur mettant en place une surveillance de masse au niveau mondial. Il y a notamment le programme PRISM [10] qui récolte la plupart [11] des données des utilisateurs des grandes compagnies américaines telles que Facebook, Yahoo ! et Google grâce à des partenariats avec ces entreprises.

Ces révélations ont permis de remettre en question la vision du public mondial sur la

4. Bosqué, « Tor, la face chiffrée d'Internet ».
5. Tréguer, « Pouvoir et résistance dans l'espace public : une contre-histoire d'Internet (XVe -XXIe siècle) », sous-section 6.2.1. « Premières inquiétudes sur l'impact de l'informatique pour la vie privée », pp. 207–209 .
6. Ibid., sous-section 9.1.3 « La libéralisation longtemps retardée du droit au chiffrement en France », pp. 309–314.
7. Loi n° 2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique - Article 30, paragraphe I.
8. Chaque site sur lequel nous naviguons peut identifier notre ordinateur grâce à une adresse — dite adresse IP — qui est propre à chaque ordinateur. Elle peut fournir de nombreux renseignements dont la localisation géographique (au niveau de la ville voire du quartier) de l'ordinateur.
9. Bosqué, « Tor, la face chiffrée d'Internet ».
10. Greenwald et MacAskill, « NSA Prism program taps in to user data of Apple, Google and others ».
11. Les diaporamas divulgués montrent que selon les études de la NSA, la majorité des communications numériques mondiales passent par les États-Unis, ce qui facilite leur interception.

controverse vie privée/surveillance [12]. Jusqu'alors seul un public technophile sensibilisé aux questions des libertés sur Internet s'inquiétait de la disparition de ce droit fondamental. Elles questionnent la gouvernance d'Internet, et donc des sujets législatifs, mais aussi les choix techniques de l'organisation de ce réseau. Jusqu'à présent, la gouvernance d'Internet pouvait être considérée comme en grande partie américaine de part le rôle important qu'a joué ce pays dans la création de ce réseau et la nationalité des entreprises majeures du domaine. Ces dernières sont des acteurs important de l'architecture actuelle du réseau. Les agissements des services de la NSA ont entraîné une grande méfiance envers les acteurs américains de l'Internet. La mobilisation citoyenne qui a dépassé le cercle des personnes aux connaissances techniques du sujet s'est emparée de cette controverse en s'apuyant sur les nouvelles informations disponibles. Nous pouvons considérer ici que l'agence mène une politique mettant clairement à mal la liberté de disposer de sa vie privée pour privilégier les raisons d'État.

Afin d'appliquer sa politique de collecte « anyone, anywhere, anytime [13] », la NSA a mis en place une stratégie de lutte prioritaire contre le chiffrement révélée en novembre 2013 par le *New York Times* [14]. L'agence a tout d'abord entrepris de diminuer les standards de cryptographie dès leur conception pour les rendres plus facilement déchiffrables par ses services. Elle mène aussi une politique active d'influence du marché commercial du chiffrement pour contenir au maximum l'accroissement de la sécurité des équipements et logiciels proposés à travers de nombreux liens avec les entreprises et des opérations d'espionnage. Malgré ses recherches, certaines technologies de chiffrement, comme le réseau TOR, le logiciel de chiffrement de mail PGP ou de stockage TrueCrypt restaient en 2012 très difficiles voire impossibles à déchiffrer par la NSA [15]. Paradoxalement, ces logiciels sont régulièrement utilisés par les agents de la NSA ce qui peut confirmer leur efficacité. Le chiffrement se pose ici dans un contexte de surveillance accrue, comme une technologie majeure mais controversée de la protection de la vie privée numérique .

Les libertés fondamentales à l'ère du tout numérique : un nouveau paradoxe ?

Nous avons vu que par ses différentes applications le chiffrement peut permettre de défendre nos libertés fondamentales — et plus particulièrement notre droit à la vie privée — sur Internet. Cependant, malgré un contexte de surveillance facilitée, son usage reste controversé, notamment pour des raisons de sécurité d'État.

L'usage de la cryptographie a permis de soulever un paradoxe. Nous voulons faire valoir notre droit à la vie privée, c'est pourquoi de nombreux mouvements citoyens sont apparus, à différentes périodes, pour mobiliser leurs gouvernements à légiférer sur l'usage de cette technologie. Lors des révélations d'Edward Snowden [16], ces mêmes mouvements ont manifesté leur opposition à la tolérance de leurs représentants politiques envers les pratiques de surveillance. *A contrario*, ces dernières années, malgré le contexte que l'on aurait pu croire favorable à un retour du respect des libertés fondamentales, de nombreux pays — dont la France et le Royaume-Uni — ont voté des lois légalisant de nombreuses pratiques de surveillance numérique. Ces décisions politiques ont catalysé de nombreuses réactions citoyennes et ont relancé des mouvements de défense des droits spécialisés dans la sphère numérique. Ainsi, nous pouvons observer une nouvelle distribution de la lutte pour la défense de nos libertés : des mouvements initialement issus de la sphère technique et « hacker » ont rencontré des associations spécialisées dans la défense des droits de l'Homme [17].

12. Musiani, « Edward Snowden, l'« homme-controverse » de la vie privée sur les réseaux ».
13. On peut traduire cette ligne directrice par : tout le monde, partout, tout le temps.
14. Poitras et Risen, « N.S.A. Report Outlined Goals for More Power ».
15. Appelbaum et al., « Inside the NSA's War on Internet Security ».
16. Musiani, « Edward Snowden, l'« homme-controverse » de la vie privée sur les réseaux ».
17. Tréguer, « Pouvoir et résistance dans l'espace public : une contre-histoire d'Internet (XVe -XXIe siècle) », section 11.2 « Le « paradoxe Snowden » », pp. 324–336.

Enfin, nous pouvons nous questionner : l'ère numérique n'impose-t-elle pas une redéfinition de la notion de vie privée[18] ? En effet, cette notion est multiple et peut regrouper de nombreuses définitions. Selon l'auteur, elle peut regrouper « la liberté de ne pas être surveillé dans certaines conditions spatiales ou matérielles, par exemple chez soi ; l'attente légitime qu'une communication entre individus qui satisfait certains critères soit limitée à ces individus ; la capacité à s'exprimer de façon anonyme ». Nous observons alors que cette notion peut complètement être redéfinie par les nouveaux usages introduits par Internet. De plus, notre société numérique a permis à l'État de développer une nouvelle stratégie de surveillance affaiblissant ainsi cette notion.

Références bibliographiques de la synthèse culturelle

Articles de recherche

Bosqué, Camille. « Tor, la face chiffrée d'Internet : entretien avec Lunar ». In : *Vacarme* 69.4 (2014), p. 79. ISSN : 1253-2479, 2107-092X. DOI : 10.3917/vaca.069.0079. URL : http://www.cairn.info/revue-vacarme-2014-4-page-79.htm (visité le 03/03/2018).

Diffie, Whitfield et Martin Hellman. « New directions in cryptography ». In : *IEEE Transactions on Information Theory* 22.6 (nov. 1976), p. 644–654. DOI : 10.1109/TIT.1976.1055638. URL : http://ieeexplore.ieee.org/document/1055638/.

Musiani, Francesca. « Edward Snowden, l'« homme-controverse » de la vie privée sur les réseaux ». In : *Hermès La Revue* 73.3 (2015), p. 209–215. URL : https://www.cairn.info/revue-hermes-la-revue-2015-3-page-209.htm.

Thèses

Tréguer, Félix. « Pouvoir et résistance dans l'espace public : une contre-histoire d'Internet (XVe -XXIe siècle) ». Thèse de doct. Paris : EHESS, nov. 2017. 600 p. URL : https://halshs.archives-ouvertes.fr/tel-01631122.

Articles journalistiques

Appelbaum, Jacob et al. « Inside the NSA's War on Internet Security ». In : *Spiegel Online* (28 déc. 2014). URL : http://www.spiegel.de/international/germany/inside-the-nsa-s-war-on-internet-security-a-1010361.html.

Greenwald, Glenn et Ewen MacAskill. « NSA Prism program taps in to user data of Apple, Google and others ». In : *The Guardian* (7 juin 2013). URL : https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.

Poitras, Laura et James Risen. « N.S.A. Report Outlined Goals for More Power ». In : *The New-York Times* (22 nov. 2013). URL : http://www.nytimes.com/2013/11/23/us/politics/nsa-report-outlined-goals-for-more-power.html.

18. Musiani, « Edward Snowden, l'« homme-controverse » de la vie privée sur les réseaux ».

B. Démonstration complémentaire

Cette démonstration nous permet de montrer l'inégalité suivante :

$$\forall x \in [-0.5, +\infty[, \quad \log(1+x) \geqslant x - Cx^2 \quad \text{, avec } C \geqslant 2.$$

Cette dernière est nécessaire aux calculs de la section 2.3.2 portant sur la probabilité d'être inversible modulo n pour un entier relatif x.

Posons $f(x) = \log(1+x) - x + Cx^2$, avec $C \geqslant 2$. Nous devons démontrer que pour tout $x \geqslant -0.5$, $f(x) \geqslant 0$.

$$\forall x \in [-0.5, +\infty[, \quad f'(x) = \frac{1}{1+x} - 1 + 2Cx = \frac{x(-1+2C+2Cx)}{1+x}.$$

Sur l'intervalle $[-0.5, +\infty[$, $1+x > 0$. En utilisant l'hypothèse que $C \geqslant 2$, alors $2Cx + 2C - 1 > 0$. Le signe de la dérivée dépend donc du signe de x.

De plus, nous avons $f' < 0$ sur $[-0.5, 0[$ et $f' > 0$ sur $]0, +\infty[$. Par conséquent, f est décroissante sur $[-0.5, 0]$ puis croissante sur $[0, +\infty[$. Elle admet donc un minimum en $x = 0$. Comme $f(0) = 0$, nous obtenons l'inégalité $f(x) \geqslant 0$.

C. Algorithmes de résolution

Cette annexe regroupe l'ensemble des mesures effectuées sur nos algorithmes. Les tables C.1, C.2 et C.3 sur les algorithmes de résolution ont permis de créer les graphes des figures 2.1 et 2.2.

Ordre du groupe	Mémoire (en ko)	Temps (en secondes)
10	9522	0
16	9569	0
36	9521	0
66	9582	0
130	9565	0
256	9589	0
520	9575	0
1030	9586	0
2052	9507	0
4098	9559	0
8208	9511	0
16410	9578	0
32770	9564	0
65536	9516	0
131100	9538	0, 03
262146	9556	0, 05
524308	9549	0, 02
1048582	9515	0, 14
2097168	9537	0, 27
4194318	9517	0, 09
8388616	9564	0, 19
16777258	9566	2, 44
33554466	9515	6, 96
67108878	9530	8, 04
134217756	9591	21, 32
268435458	9527	33, 03
536870922	9534	34, 29
1073741826	9588	83, 55
2147483658	9551	697, 21
4294967310	9539	1534, 61
8589934608	9532	1334, 25
17179869208	9562	2259, 91
34359738420	9530	6309, 93
68719476766	9564	10631, 67
137438953480	9583	20871, 69

TABLE C.1 – Mesures du temps d'exécution (en secondes) et de l'utilisation de la mémoire (en ko) de notre implémentation de l'algorithme naïf en fonction de l'ordre du groupe dans lequel on travaille.

Ordre du groupe	Mémoire (en ko)	Temps (en secondes)
10	9837	0
16	9858	0
36	9883	0
66	9825	0
130	9822	0
256	9897	0
520	9866	0
1030	9813	0
2052	9821	0
4098	9889	0
8208	9883	0
16410	9877	0
32770	9875	0
65536	9833	0
131100	9864	0
262146	9849	0
524308	9822	0
1048582	9861	0,02
2097168	9857	0,02
4194318	9823	0,02
8388616	9831	0,03
16777258	9806	0,06
33554466	9858	0,11
67108878	9822	0,13
134217756	9896	0,2
268435458	9940	0,27
536870922	10100	0,3
1073741826	11300	0,52
2147483658	16520	1,2
4294967310	16850	1,92
8589934608	22688	2,22
17179869208	28024	3,77
34359738420	35628	7
68719476766	46308	11,6
137438953480	61432	20,71
274877906950	82924	37,91
549755813910	95321	70,56
1099511627790	110547	125,98
2199023255578	142016	254,53
4398046511118	215516	529,7
8796093022236	302672	1042,21
17592186044422	422268	2187,84
35184372088890	595680	4672,99
70368744177678	838400	9826,11
140737488355332	1181624	20370,26

Table C.2 – Mesures du temps d'exécution (en secondes) et de l'utilisation de la mémoire (en ko) de notre implémentation de l'algorithme Baby-Step Giant-Step en fonction de l'ordre du groupe dans lequel on travaille.

Ordre du groupe	Mémoire (en ko)	Temps (en secondes)	Nombre de mesures effectuées
10	9829	0	100
16	9848	0	100
36	9815	0	100
66	9882	0	100
130	9843	0	100
256	9848	0	100
520	9861	0	100
1030	9864	0	100
2052	9818	0	100
4098	9852	0	100
8208	9892	0	100
16410	9869	0	100
32770	9828	0	100
65536	9870	0,01	100
131100	9811	0	100
262146	9806	0,03	100
524308	9849	0	100
1048582	9849	0	100
2097168	9890	0	100
4194318	9858	0,02	100
8388616	9862	0	100
16777258	9897	0,01	100
33554466	9816	0,99	100
67108878	9872	0,05	100
134217756	9818	0,08	100
268435458	9884	0,13	100
536870922	9866	0,1	100
1073741826	9837	0,21	100
2147483658	9878	0,21	100
4294967310	9866	0,54	100
8589934608	9830	0,11	100
17179869208	9820	0,17	100
34359738420	9826	1,32	100
68719476766	9820	1,15	100
137438953480	9860	0,74	100
274877906950	9825	2,34	100
549755813910	9816	4,69	100
1099511627790	9827	13,17	100
2199023255578	9858	3,45	100
4398046511118	9828	6,41	100
8796093022236	9882	14,89	100
17592186044422	9853	12,22	100
35184372088890	9836	40,31	100
70368744177678	9806	25,47	100
140737488355332	9886	34,68	100
281474976710676	9835	53,04	100
562949953421381	9870	148,24	50
112589990684268	9834	166,81	25
2251799813685312	9812	240,8	25
4503599627370550	9874	208,4	26

Ordre du groupe	Mémoire (en ko)	Temps (en secondes)	Nombre de mesures effectuées
9007199254741048	9836	84, 25	25
18014398509482142	9845	954, 28	14
36028797018964018	9876	795, 39	14
72057594037928016	9812	276, 33	14
144115188075855946	9874	1560, 28	12
288230376151711812	9832	2204, 6	10
576460752303423618	9825	4140, 2	11
1152921504606847066	9862	4150, 68	9
2305843009213694008	9847	2624, 5	10
4611686018427388038	9852	7904, 56	9
9223372036854775906	9863	7409, 57	10
18446744073709551666	9812	12467, 2	9
36893488147419103362	9841	10963, 25	10
73786976294838206548	9823	14478, 23	9
147573952589676412990	9874	10963, 25	10
295147905179352825912	9866	11931, 18	6
590295810358705651816	9910	21587, 62	2

TABLE C.3 – Mesures moyennes du temps d'exécution (en secondes) et de l'utilisation de la mémoire (en ko) de notre implémentation de l'algorithme Rho de POLLARD en fonction de l'ordre du groupe dans lequel on travaille.

D. Résultats expérimentaux sur nos protocoles de cryptographie à clé publique

Cette annexe regroupe le graphe et les tableaux précisant nos mesures évoquées dans la section 3.3.4.

Nombre de caractères	Chiffrement (secondes)	Déchiffrement (secondes)
1	0	0
10	0	0
100	0	0
1000	0	0,03
10000	0,1	0,43
100000	9,22	17,01
1000000	1899,2	2481,86

TABLE D.1 – Temps nécessaire pour chiffrer puis déchiffrer un message avec notre implémentation deu cryptosystème de ELGAMAL en fonction de la taille du message à chiffrer.

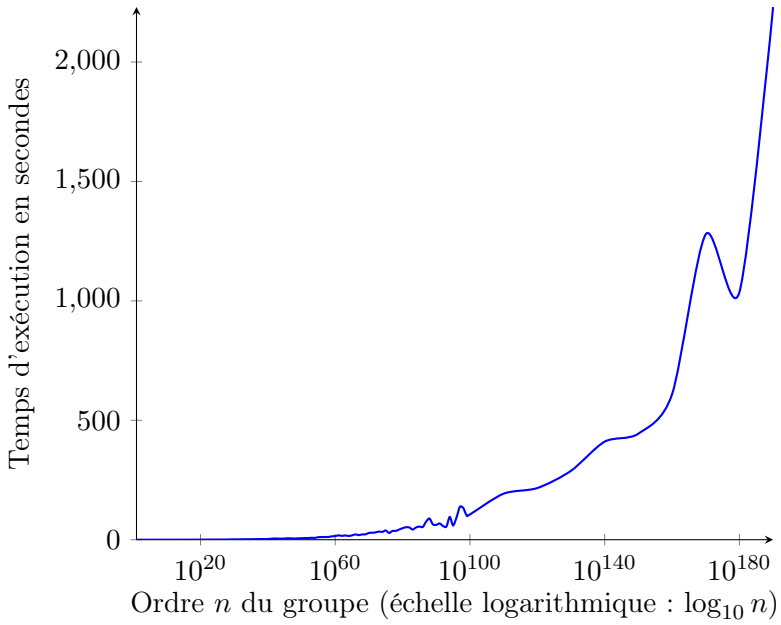

FIGURE D.1 – Temps moyen d'exécution de notre implémentation de génération d'un groupe en fonction de l'ordre de ce dernier

Ordre	Temps de construction (secondes)	Nombre de mesures
10^1	0	100
10^2	0	100
10^3	0	100
10^4	0,01	100
10^5	0,01	100
10^6	0,01	100
10^7	0,02	100
10^8	0,03	100
10^9	0,04	100
10^{10}	0,05	100
10^{11}	0,07	100
10^{12}	0,08	100
10^{13}	0,09	100
10^{14}	0,15	100
10^{15}	0,15	100
10^{16}	0,18	100
10^{17}	0,21	100
10^{18}	0,26	100
10^{19}	0,33	100
10^{20}	0,44	100
10^{21}	0,44	100
10^{22}	0,49	100
10^{23}	0,55	100
10^{24}	0,7	100
10^{25}	0,55	100
10^{26}	0,72	100
10^{27}	0,83	100
10^{28}	0,94	100
10^{29}	1,41	100
10^{30}	1,43	100
10^{31}	1,56	100
10^{32}	1,65	100
10^{33}	2,06	100
10^{34}	1,9	100
10^{35}	2,08	100
10^{36}	2,28	100
10^{37}	2,51	100
10^{38}	2,85	100
10^{39}	2,67	100
10^{40}	2,68	100
10^{41}	4,22	100
10^{42}	5,14	100
10^{43}	4,57	100
10^{44}	4,42	100
10^{45}	5,02	100

Ordre	Temps de construction (secondes)	Nombre de mesures
10^{46}	$6,11$	100
10^{47}	$5,54$	100
10^{48}	$5,13$	100
10^{49}	$5,71$	100
10^{50}	$6,31$	100
10^{51}	$6,84$	100
10^{52}	7	100
10^{53}	$8,03$	100
10^{54}	$7,69$	100
10^{55}	$10,65$	100
10^{56}	$11,03$	100
10^{57}	$11,33$	100
10^{58}	$11,56$	100
10^{59}	$14,11$	100
10^{60}	$15,16$	100
10^{61}	$18,38$	100
10^{62}	$16,16$	100
10^{63}	$17,87$	100
10^{64}	$15,68$	100
10^{65}	$18,14$	100
10^{66}	$22,29$	100
10^{67}	$19,31$	100
10^{68}	$22,25$	100
10^{69}	$22,53$	100
10^{70}	$28,42$	100
10^{71}	$29,19$	100
10^{72}	$30,74$	100
10^{73}	$34,06$	100
10^{74}	$32,84$	100
10^{75}	$38,72$	100
10^{76}	$28,36$	100
10^{77}	$36,63$	100
10^{78}	$36,89$	100
10^{79}	$42,75$	100
10^{80}	$48,2$	100
10^{81}	$52,4$	100
10^{82}	$51,33$	50
10^{83}	$42,62$	50
10^{84}	$51,33$	50
10^{85}	$54,99$	50
10^{86}	$53,4$	50
10^{87}	$75,86$	50
10^{88}	$88,64$	20
10^{89}	$64,43$	20
10^{90}	$62,07$	20

Ordre	Temps de construction (secondes)	Nombre de mesures
10^{91}	$68,04$	20
10^{92}	$57,53$	20
10^{93}	$54,83$	20
10^{94}	$95,42$	20
10^{95}	$59,92$	20
10^{96}	$92,67$	20
10^{97}	$137,36$	20
10^{98}	$133,09$	20
10^{99}	$102,75$	20
10^{100}	$105,95$	20
10^{110}	$192,87$	20
10^{120}	$216,38$	20
10^{130}	$288,52$	20
10^{140}	$410,72$	20
10^{150}	$444,09$	20
10^{160}	$608,79$	20
10^{170}	$1278,71$	20
10^{180}	$1036,53$	20
10^{190}	$2230,26$	20

Table D.2 – Mesures moyennes du temps (en secondes) nécessaire à la génération d'un groupe en fonction de l'ordre de ce dernier.

E. Algorithmes

Sur le wiki des projets CPBx se trouvent les algorithmes que nous avons implémentés sur Python3. Comme nous l'avons expliqué dans le mémoire, les implémentations ont été réalisées pour un travail dans des groupes $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$.

Tout d'abord, les fonctions communes à plusieurs algorithmes comme le calcul du pgcd ou de l'inverse modulaire sont regroupées dans le fichier `Fonctions.py`. Il contient aussi les différentes fonctions appelées par les fonctions principales de nos algorithmes pour une meilleure lecture du code. De plus, nous n'avons pas codé la fonction `millerRabin` : nous avons utilisé une version disponible sur Internet. La source se trouve dans les commentaires de notre implémentation. Les implémentations des trois algorithmes d'attaques étudiés — algorithme exhaustif, Baby-Step Giant-Step et Rho de Pollard — se trouvent dans les fichiers `.py` éponymes. Le fichier `Baby-Step Giant-Step.py` comporte différentes version de l'algorithme montrant ainsi les multiples améliorations présentées dans le mémoire. Nous avons utilisé la version 4 pour nos mesures expérimentales de temps de calcul et d'utilisation de mémoire.

Nous avons inclus dans chacunes des implémentations une partie dédiée pour effectuer différents tests de résolution. Cette partie fait appel à un autre fichier — `exemples.txt` — qui contient des triplets de paramètres pour des problèmes du logarithme discret. Dans ce fichier, les données sont donc sous la forme (p, g, h). Pour rappel, le but est de déterminer l'entier x qui vérifie $h \equiv g^x \mod p$, $0 \leqslant x < n = p - 1$.

Les trois algorithmes prennent en entrée les entiers n, h, g, p définis ci-dessus.

Les fichiers `Diffie-Hellman.py` et `ElGamal.py` contiennent l'implémentation des protocoles éponymes. Pour effectuer des tests avec le système ElGamal, le message à chiffrer doit être contenu dans la variable `text` et ne comporter que les caractères autorisés par notre implémentation. Ceux-ci sont contenus dans une liste située au-dessus de cette variable.

Pour les deux protocoles, la fonction `groupe` permet la génération d'un groupe respectant les conditions détaillées dans le mémoire. Cette fonction prend en argument deux entiers formant l'intervalle dans lequel est compris l'entier p, qui permet de former le groupe $((\mathbb{Z}/p\mathbb{Z})^\times, \times)$.

Pour le bon fonctionnement de nos différents algorithmes, il est nécessaire de placer le fichier `Fonctions.py` — ainsi que le fichier `exemples.txt` pour les algorithmes de résolution — dans le même dossier que nos algorithmes.

F. Gestion de projet

Tout au long de ce projet, nous avons dû gérer une organisation à trois personnes. Tout d'abord, nous avons eu recours à l'utilisation d'un éditeur de texte collaboratif (`Framapad`). Ainsi, il nous a été possible de rédiger des documents en commun plus rapidement et surtout plus facilement. Ce service nous a permis de plus de communiquer rapidement sur le travail à faire, en cours et celui restant. Par ailleurs, nous avons choisis de réaliser tous les livrables liés à ce projet avec le langage LaTeX. L'une des personne du groupe était déjà expérimentée dans l'utilisation de ce langage. Pour travailler plus efficacement, nous nous sommes initiés à son usage. De plus, LaTeX étant la norme dans de nombreux domaines scientifiques, cet apprentissage nous sera utile dans le futur.

Grâce à ces deux outils, nous avons pu nous répartir le travail selon les envies et compétences de chacun. Nous pouvions aisément consulter les avancées de nos travaux, les corriger ainsi qu'effectuer des ajouts. Concernant les algorithmes programmés, nous nous les sommes aussi répartis pour la première implémentation avant de les mettre en commun en vue de les corriger et les améliorer. La plupart des différentes tâches (décrites dans la figure F.1) ont été réalisées en parallèles.

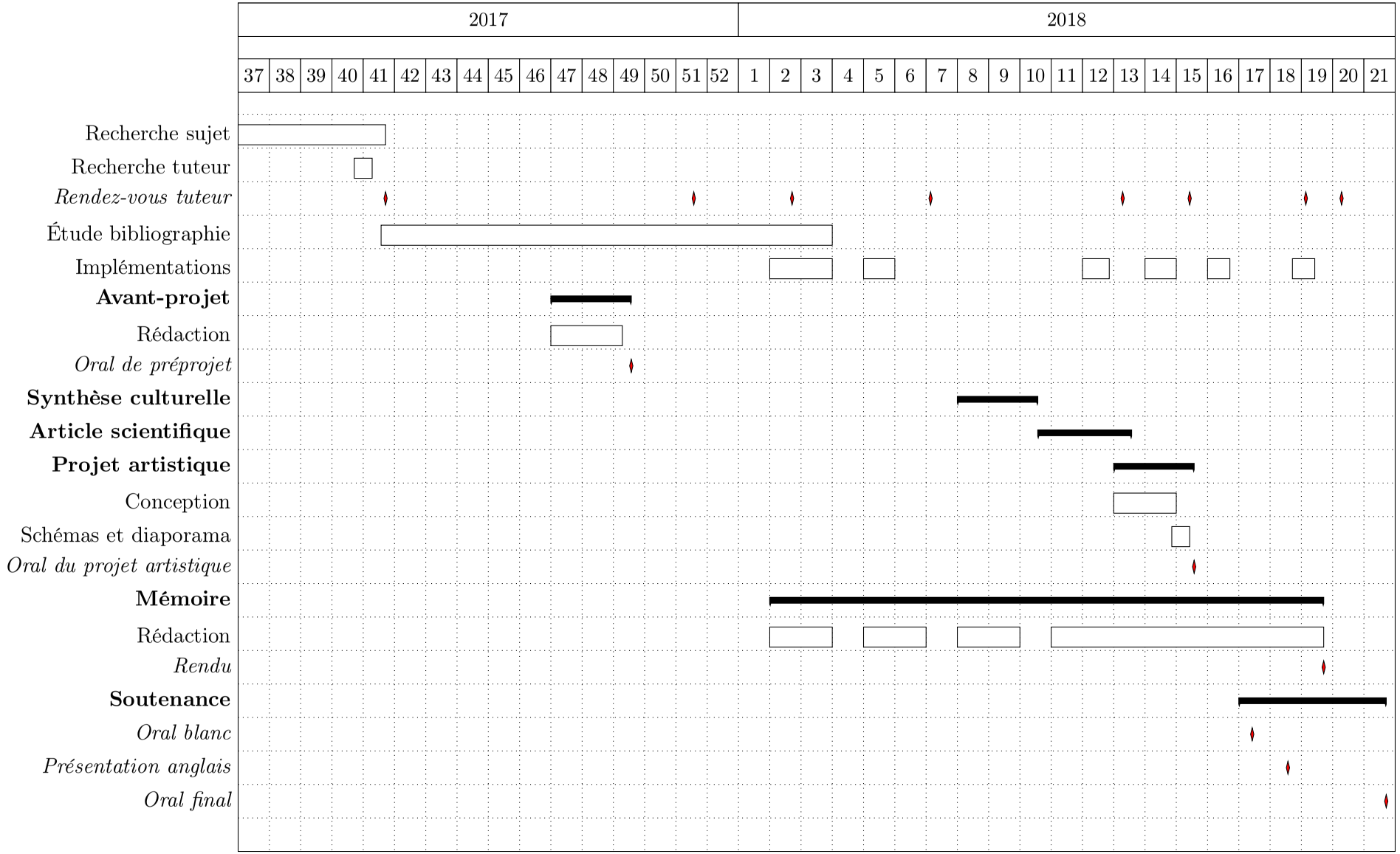

FIGURE F.1 – Diagramme de GANTT de l'organisation de notre projet final