# Cryptanalyse quantique de primitives symétriques

Xavier Bonnetain

# Cryptanalyse quantique de primitives symétriques

Xavier Bonnetain
Inria équipe-projet SECRET
directrice : María Naya-Plasencia

21 mars - 2 septembre 2016

## Le contexte général

La cryptanalyse quantique est une branche de la cryptanalyse, qui étudie la sécurité de systèmes cryptographiques face à un attaquant disposant d'un ordinateur quantique. Le résultat le plus connu concerne le système RSA dont la sécurité repose sur la difficulté de la factorisation. Ce problème devient facile avec un ordinateur quantique, grâce à l'algorithme de Shor. Cela fait une vingtaine d'années que la sécurité dans un cadre quantique de systèmes classiques est étudiée, et la plupart des résultats concernent la cryptographie asymétrique. Le premier article concernant un chiffrement symétrique date de 2012.

## Le problème étudié

L'intérêt pour la cryptanalyse quantique symétrique est très récent, et les attaques que l'on connaît à l'heure actuelle concernent des systèmes dont la structure est très spécifique. Mon stage s'est focalisé sur le développement d'outils et de méthodes de cryptanalyse quantique de chiffrements symétriques en essayant d'attaquer Treyfer, qui est un chiffrement cassé classiquement, mais dont la structure se rapproche de celle des chiffrements actuels. Ces travaux sont parmi les premiers sur ce sujet, et s'inscrivent dans le prolongement d'un article écrit par Marc Kaplan, Gaëtan Leurent, Anthony Leverrier et María Naya-Plasencia [15] et présenté cette année à la conférence CRYPTO.

## La contribution proposée

La cryptanalyse de Treyfer n'a pour l'instant pas abouti, mais trois résultats en sont ressortis.
- la détermination du taux d'erreur acceptable d'un algorithme de test lors d'une recherche exhaustive quantique,
- la généralisation d'algorithmes quantiques à un cadre potentiellement intéressant en cryptanalyse symétrique,
- la découverte de nouvelles attaques contre des systèmes de chiffrement ayant une structure particulière.

## Les arguments en faveur de sa validité

Les attaques concrètes en cryptographie symétrique très souvent des énumérations exhaustives dont le but est de chercher des éléments ayant une propriété donnée, et le premier résultat permet de calibrer explicitement un algorithme de test probabiliste quantique pour obtenir une probabilité de réussite fixée. Les algorithmes généralisés résolvent un problème un peu spécifique (certaines instances du problème du sous-groupe caché), et ont donc potentiellement moins d'applications. Cependant, une attention particulière a été portée sur le coût de ces algorithmes. Les nouvelles attaques sont des exemples d'applications des algorithmes précédents.

## Le bilan et les perspectives

Ces contributions constituent des briques de base pouvant aider à construire des attaques élaborées, ou, en dehors du domaine de la cryptanalyse, permettre de calculer plus finement les coûts et les taux de réussite. Tout cela constitue entrée en matière pour ma thèse qui débute en septembre, dont le sujet

est la formalisation de méthodes de cryptanalyse symétrique quantique, comme il en existe pour la cryptanalyse symétrique classique.

### Les remerciements

Je tiens à remercier toute l'équipe SECRET, et en particulier ma tutrice María Naya-Plasencia pour son aide et son temps en dépit d'un emploi du temps extrêmement contraint, ainsi qu'André Chailloux et Anthony Leverrier pour leur expertise en informatique quantique.

## Table des matières

# 1 Introduction

## 1.1 Cryptographie

L'objectif premier de la cryptographie est la protection de l'information. Cela se fait typiquement au moyen d'algorithmes de chiffrement, dont on distingue deux grandes familles : symétrique et asymétrique. Un chiffrement symétrique permet à plusieurs personnes partageant un secret commun de communiquer sans qu'un adversaire puisse apprendre d'information sur les messages envoyés (les clairs) ou le secret (la clé). Un chiffrement asymétrique permet de partager une information sécrète entre plusieurs personnes, en ne partageant que des informations publiques. Les chiffrement asymétriques étant bien plus coûteux en temps de calcul, on utilise généralement, pour le transfert d'informations, un système hybride, avec une partie initiale asymétrique pour transmettre une clé symétrique, puis un chiffrement symétrique pour transmettre les données. On utilise aussi des chiffrements symétriques dans des environnements très contraints (téléphonie, matériel embarqué), où l'on ne peut pas se permettre de faire des calculs trop lourds.

## 1.2 Cryptanalyse

La cryptanalyse est la science de l'attaque des systèmes cryptographiques. Elle est fondamentale, car c'est la seule façon de vérifier qu'une primitive respecte bien le niveau de sécurité annoncé. En cryptographie asymétrique, la sécurité est basée sur la difficulté de problèmes mathématiques bien connus, comme le problème du logarithme discret ou de la factorisation. Ce n'est pas le cas en cryptographie symétrique, où la sécurité est définie de façon plus heuristique : une primitive est considérée comme sûre si elle a été largement analysée et qu'aucune faiblesse n'a été trouvée.

De nombreux modèles d'attaque existent, selon l'objectif (récupérer de l'information sur les clairs, récupérer la clé, chercher des biais...) et selon ce que l'on peut faire (attaques à clairs connus, à clairs choisis (adaptatifs ou non), à clé connue...). Ces attaques sont à comparer avec les méthodes génériques (souvent l'énumération exhaustive des possibilités), et une primitive n'est sûre que si la meilleure attaque connue est l'attaque générique. Les attaques trouvées ne sont pas nécessairement réalisables, et peuvent nécessiter des ressources bien trop importantes. Elles montrent néanmoins des propriétés indésirables (et non prévues par les concepteurs) de la fonction. De plus, les attaques ne font que s'améliorer avec le temps, et il est courant de voir une attaque théorique évoluer pour devenir applicable ; cela a par exemple été le cas pour les attaques pratiques sur MD5, RC4 ou SHA-1. On considère donc qu'une primitive pour laquelle de telles attaques existent est cassée.

## 1.3 Cryptanalyse quantique

L'ordinateur quantique est un modèle de calcul qui a commencé par être une curiosité, puis qui a attiré beaucoup d'attention lorsque des algorithmes présentant des gains en complexité sur des problèmes utiles ont été trouvés. S'il n'existe à l'heure actuelle que des prototypes dans des laboratoires, de nombreuses institutions investissent pour développer un ordinateur quantique suffisamment puissant pour effectuer des calculs trop coûteux pour des ordinateurs classiques.

L'éventuelle apparition d'un ordinateur quantique au cours des prochaines décennies a des conséquences actuelles en cryptographie [5]. En effet, le jour où cet ordinateur arrivera, on devra disposer d'algorithmes qui sont sûrs. De plus, on souhaite que les secrets d'aujourd'hui restent secrets à l'avenir, et il est donc important d'anticiper les problèmes de sécurité qui pourraient apparaître avec cet ordinateur.

En cryptographie asymétrique, l'ordinateur quantique a un effet important. L'algorithme de Shor [22] permet de résoudre efficacement les problèmes de factorisation et de logarithme discret, ce qui rend caduc le chiffrement RSA ou le système d'échange de clés de Diffie-Hellman. Un des enjeux majeurs de la cryptographie asymétrique aujourd'hui est de trouver des systèmes dits *post-quantiques*, où l'usage d'un ordinateur quantique n'apporterait pas un gain important. Diverses pistes existent pour cela, comme la cryptographie basée sur les codes, sur les réseaux, ou des choses plus exotiques, comme les isogénies sur des courbes elliptiques supersingulières.

En cryptographie symétrique, l'effet est différent. L'algorithme de Grover [13] permet de rechercher un élément parmi $N$ qui répond à un critère fixé en $\Theta(\sqrt{N})$ itérations, là où un algorithme classique a besoin de $\Theta(N)$ itérations. Cela donne une attaque générique permettant de récupérer une clé de $n$ bits en $\Theta(2^{n/2})$ chiffrements (à partir de quelques couples clair/chiffré, la bonne clé est la seule qui donne le chiffré correspondant à chaque clair, ce qui donne le critère). On a donc un gain quadratique, et la croyance populaire était qu'il suffirait de doubler la taille des clés pour obtenir un niveau de sécurité équivalent. Cependant, des travaux récents visant à mieux comprendre les possibilités de l'ordinateur quantique en cryptanalyse symétrique [18, 3, 15] ont montré que certaines constructions sûres classiquement sont attaquables par un ordinateur quantique.

En cryptanalyse quantique, comme en cryptanalyse classique, plusieurs modèles sont possibles : on peut se restreindre à des requêtes de chiffrement/déchiffrement classiques, ou autoriser des requêtes en superposition. Ce second modèle est plus puissant, et peut s'appliquer si l'algorithme opère sur des informations publiques, ou si l'on dispose, par exemple, d'une implémentation *white-box* du chiffrement (un programme effectuant le chiffrement, sans que l'on puisse directement en extraire les secrets). Il a l'avantage d'être simple à définir, et étant un modèle très fort, si une primitive y est sûre, elle sera sûre dans des modèles plus contraignants.

# 2 Notions d'algorithmes quantiques

Les algorithmes quantiques sont des circuits, opérant sur des qubits, via des portes (ou opérateurs) quantiques. Celles-ci sont l'analogue des portes logiques, la principale différence est qu'elles doivent être inversibles (et ont donc notamment autant d'entrées que de sorties). On dispose de plus d'une façon de revenir dans le monde classique, la mesure d'un qubit, qui permet d'obtenir un bit classique.

## 2.1 Qubits

Les qubits sont l'analogue des bits. Ils n'appartiennent pas à $\{0,1\}^n$, mais à l'espace hermitien $\mathbb{C}^{2^n}$ (que l'on munit donc du produit scalaire hermitien, et de la norme associée), dont une base est l'ensemble $\{0,1\}^n$. Les états possibles sont donc l'ensemble des combinaisons linéaires complexes (non nulles) de ces éléments. Ils sont de plus définis à un coefficient multiplicatif global près, et on impose généralement qu'ils soient unitaires.

On utilise la notation de Dirac ($|\ \rangle$) pour les décrire. La base de l'espace de Hilbert peut s'écrire $\{|b_1\rangle \dots |b_n\rangle | (b_1, \dots b_n) \in \{0,1\}^n\}$, ou $\{|b_1 \dots b_n\rangle | (b_1, \dots b_n) \in \{0,1\}^n\}$ pour être plus compact. On peut aussi utiliser aussi une notation en base 2 pour décrire les qubits. La base est alors $\{|x\rangle | x \in [0, 2^n - 1]\}$. On parle de registre pour décrire un bloc $|\ \rangle$. On utilisera par la suite une notation en blocs : $|0\rangle\,|0\rangle$ correspond à un état de deux registres (dont la taille est implicite, fixée selon le problème).

On utilisera aussi une notation tensorielle pour décrire des groupes de registres : $|x_1\rangle \dots |x_n\rangle$ pourra être noté $\bigotimes_i |x_i\rangle$.

Par la suite, lorsque cela ne prête pas à confusion, le coefficient de normalisation pourra être omis.

**Mesure** On peut mesurer des qubits pour obtenir des bits classiques. Le carré de la norme des différents éléments de la base s'interprète alors comme la probabilité de mesurer cet élément. Par exemple, pour $|0\rangle$, on mesurera 0 avec probabilité 1. Pour $\frac{|01\rangle+2i|10\rangle}{\sqrt{5}}$, on mesurera 01 avec probabilité $\frac{1}{5}$, et 10 avec probabilité $\frac{4}{5}$. De plus, la mesure projette les qubits dans l'état mesuré : si on mesure 10, l'état devient $|10\rangle$.

On peut aussi ne mesurer qu'une partie des qubits, et on projette dans ce cas l'état du système global sur les cas compatibles avec ce que l'on a mesuré. Pour $\frac{|01\rangle+2i|10\rangle}{\sqrt{5}}$, si on mesure le premier qubit, on forcera le second à être dans l'état opposé. Ces états, où une mesure partielle influence ce que l'on n'a pas mesuré, ne peuvent pas être décomposés en sous-états indépendants pour chacun des qubits, et on parle d'intrication, ou d'état non-local.

## 2.2 Opérateurs quantiques

On utilise des opérateurs (ou circuits) quantiques pour transformer des qubits. On parle de porte quantique pour les briques élémentaires d'un circuit. Elles sont réversibles (pour un opérateur $P$, on note son inverse $P^\dagger$), et linéaires (dans l'espace hermitien), on peut donc décrire leur comportement à partir de leur effet sur une base. On peut les voir comme des matrices de taille $2^n \times 2^n$, si $n$ est le nombre de qubits sur lesquelles elles opèrent.

De nombreux opérateurs existent, ici sont présentés ceux qui nous intéresseront par la suite.

### 2.2.1 Inversion de phase

La porte

$$Z : |b\rangle \mapsto (-1)^b |b\rangle$$

permet d'inverser la phase d'un ensemble de qubits en fonction d'un qubit précis. Elle n'a pas d'analogue classique.

### 2.2.2 Porte de Hadamard

Cette porte n'a pas non plus d'analogue classique. Elle est définie par

$$H : |b\rangle \mapsto \frac{|0\rangle + (-1)^b |1\rangle}{\sqrt{2}}.$$

Elle est involutive ($H^\dagger = H$).

Cette porte est très importante, et a de nombreuses applications dans les algorithmes quantiques. Les qubits $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ et $H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$ sont respectivement notés $|+\rangle$ et $|-\rangle$.

### 2.2.3 CNOT

L'opérateur CNOT, pour *controlled not* est l'équivalent quantique du xor. Il opère sur 2 qubits, et effectue l'opération $|x\rangle |y\rangle \mapsto |x\rangle |x \oplus y\rangle$.

### 2.2.4 Transformée de Fourier quantique

Ce circuit opère sur n qubits, et effectue l'opération

$$F_n : |x\rangle \mapsto \sum_y \exp\left(2i\pi \frac{xy}{2^n}\right) |y\rangle .$$

Cette transformée de Fourier est « rapide », dans le sens où un nombre de portes linéaire en $n$ suffit pour l'implémenter.

**Remarque 1.** On peut la voir comme une généralisation de la porte de Hadamard : $F_1 = H$.

### 2.2.5 Les fonctions classiques

On peut transcrire une fonction $f$ qui opère sur des bits en un opérateur sur des qubits, à partir de son circuit logique, en construisant $F : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$. Cela est toujours faisable. De plus, ces opérateurs sont involutifs (appliquer F à $|x\rangle |y \oplus f(x)\rangle$ redonne $|x\rangle |y\rangle$. Cela ne dit rien sur l'inversibilité de $f$.) Si on souhaite transcrire un algorithme dont le temps d'exécution peut varier, on fixe un nombre d'opérations maximal, et on peut alors créer le circuit, où l'on transforme les cas prenant trop de temps en des cas d'échec.

### 2.2.6 Composition d'opérateurs

On utilise une notation tensorielle pour décrire l'application d'opérateurs à une partie des qubits. Par exemple, $F_n \otimes \text{CNOT}$ applique $F_n$ sur les $n$ premiers qubits, et CNOT sur les 2 derniers. On peut utiliser l'opérateur identité sur $n$ qubits noté $\mathbb{1}_n$ pour combler. Pour décrire un opérateur qui applique le même opérateur $P$ à une succession de $n$ registres, on utilise la notation plus compacte $P^{\otimes n}$.

La composition pour faire un circuit se note comme la composition matricielle, $F_{2n}\ \mathbb{1}_n \otimes H^{\otimes n}$ est l'application de $\mathbb{1}_n \otimes H^{\otimes n}$ suivie de celle de $F_{2n}$.

## 2.3 Contraintes

Il y a quelques restrictions concernant les opérateurs quantiques. Il n'est pas possible de dupliquer ou de supprimer un qubit. On peut copier le bit classique dans le qubit grâce à un CNOT, qui permet de passer de $\sum |x\rangle |0\rangle$ à $\sum |x\rangle |x\rangle$. Il est cependant impossible de faire un opérateur qui à partir de $\sum |x\rangle |0\rangle$ construit $(\sum |x\rangle) \otimes (\sum |x\rangle)$. Si on tient à créer un tel état, il faut utiliser 2 fois l'opérateur ayant permis de construire $\sum |x\rangle$. De plus, les portes étant réversibles, elles ont autant de qubits en entrées qu'en sortie.

On a donc un nombre de qubits constant au cours du calcul. On peut bien sur en rajouter au début pour avoir plus de place, mais pour le résultat, cela peut être problématique : on souhaite que l'état à la fin ne soit pas intriqué avec ces qubits supplémentaires, car cela peut avoir un impact sur la suite. De plus, il se peut que les qubits en entrée aient été modifiés par le circuit, ce qui empêche de les réutiliser par la suite. L'astuce pour régler ce problème est de faire de l'*uncomputing*. En effet, puisque toutes les portes sont réversibles, on peut facilement inverser l'opérateur : il suffit de le symétriser et de remplacer chaque porte par son inverse. On peut alors faire le calcul, copier les qubits qui nous intéressent, puis inverser la transformation. Si les qubits que l'on copie correspondent bien à une fonction (et ne dépendent que de $x$), cela donne un opérateur qui effectue $|x\rangle |0\rangle |y\rangle \mapsto |x\rangle |0\rangle |y \oplus f(x)\rangle$, qui se comporte exactement comme l'opérateur $|x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$.

## 2.4 Recherche de période

### 2.4.1 Problème général

La recherche de période (ou problème du sous-groupe caché) est une classe de problèmes où des algorithmes quantiques exponentiellement plus rapides que leurs pendants classiques ont été trouvés. Le problème est le suivant : Soit un groupe $(\mathbb{G}, *)$, $H$ un sous-groupe de $\mathbb{G}$, $\mathcal{E}$ un ensemble quelconque et $f : \mathbb{G} \to \mathcal{E}$ une fonction telle que, pour tout $x \in \mathbb{G}$, $f(x) = f(x * y)$ si $y \in H$, et $f(x) \neq f(x * y)$ sinon. L'objectif est de retrouver la période de la fonction, c'est à dire une base de $H$. Les algorithmes de Simon [23] et de Shor [22] sont des exemples d'algorithmes quantiques résolvant ce problème en temps polynomial, dans des groupes précis.

### 2.4.2 Algorithme de Simon

L'algorithme de Simon résout la recherche de période dans le groupe $(\mathbb{Z}_2^n, \oplus)$. Il fonctionne comme suit :

On dispose d'un oracle $O_f : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$, avec la promesse que :

1. $\exists a : \forall x, f(x) = f(x \oplus a)$ (a peut être nul)
2. $\forall b \notin \{0, a\}, \forall x, f(x) \neq f(x \oplus b)$

L'objectif est de retrouver $a$. On itère alors $m$ fois le processus suivant :

- On part de l'état $|0\rangle |0\rangle$.
- On applique $H^{\otimes n}$ sur le premier registre, pour obtenir $\sum_i |i\rangle |0\rangle$
- On applique $O_f$, pour obtenir $\sum_i |i\rangle |f(i)\rangle$
- (optionnel) On mesure le second registre, on récupère alors $f(x_0)$ pour un $x_0$ inconnu, et le premier registre devient $|x_0\rangle + |x_0 \oplus a\rangle$
- On applique $H^{\otimes n}$ sur le premier registre, pour obtenir l'état $\sum_i \left((-1)^{x_0 . i} + (-1)^{(x_0 \oplus a) . i}\right) |i\rangle$

– On mesure le registre, pour récupérer un vecteur $v$

L'état avant la mesure peut se réécrire $\sum_i (-1)^{x_0.i} \left(1 + (-1)^{a.i}\right) |i\rangle$. On remarque que l'amplitude de l'élément $|i\rangle$ est proportionnelle à $(1+(-1)^{a.i})$, et est donc nulle si $a.i = 1$. On ne récupère ainsi que des vecteurs $v$ normaux à $a$, et on peut récupérer $a$ si la famille obtenue est de rang maximal. Chaque mesure étant faite indépendamment, on peut estimer le nombre d'itérations nécessaires.

**Proposition 1.** Une famille de $m \geq n$ vecteurs tirés uniformément dans $\mathbb{Z}_2^n$ est de rang $n$ avec probabilité

$$P(m,n) = \prod_{i=0}^{n-1} \left(1 - \frac{1}{2^{m-i}}\right).$$

*Démonstration.* Soit $v_i$ une famille de $m \geq n$ vecteurs indépendants de $\mathbb{Z}_2^n$. On considère la matrice $m \times n$ $\left(v_1 \middle| \cdots \middle| v_m\right)$. La famille est de rang n si et seulement si les lignes de la matrice sont des vecteurs formant une famille libre. Si on considère que l'on tire ces vecteurs au fur et à mesure, cela impose qu'ils ne sont pas dans l'espace vectoriel formé par les $i$ vecteurs précédents. Cet espace vectoriel contient $2^i$ éléments, et laisse donc $2^m - 2^i$ choix possibles, sur $2^m$. □

**Proposition 2.** $P(n,n) \geq \frac{1}{4} + \frac{1}{2^{n+1}}$

*Démonstration.* Par récurrence, avec $P(n+1,n+1) = P(n,n)\left(1 - \frac{1}{2^{n+1}}\right)$. □

Si $a = 0$, on génère $\mathbb{Z}_2^n$, et sinon, on génère un hyperplan de $\mathbb{Z}_2^n$, isomorphe à $\mathbb{Z}_2^{n-1}$ (et on tire nos vecteurs uniformément dedans). Il suffit donc d'un nombre de vecteurs linéaire en $n$ (moins de $4n$) pour espérer avoir toute l'information sur $a$, que l'on peut ensuite récupérer par un post-traitement classique, en temps polynomial en $n$.

**Remarque 2.** Si la fonction n'est pas parfaite, c'est à dire qu'il existe des couples $(x_0, b)$ pour lesquels $f(x_0 \oplus b) = f(x_0)$, et $b \neq a$, cela ne change pas la correction de l'algorithme. En effet, lors de la première mesure, on obtient le qubit $|x_0\rangle + |x_0 \oplus a\rangle + |x_0 \oplus b\rangle + |x_0 \oplus a \oplus b\rangle$. L'étape suivante donne $\sum_i \left((-1)^{x_0.i}(1 + (-1)^{a.i}) + (-1)^{(x_0 \oplus b).i}(1 + (-1)^{a.i})\right) |i\rangle$, et on mesurera un vecteur normal à $a$ et à $b$. Cela peut ralentir l'algorithme si un certain $b$ est très présent, car on veut une famille de vecteurs de rang maximal.

### 2.4.3 Décalage caché

Le problème du décalage caché (ou *hidden shift*) est une variante de la recherche de période, qui s'exprime comme suit : Dans un groupe $(\mathbb{G}, *)$, on dispose de deux fonctions, $f_0$ et $f_1$, injectives, telles qu'il existe un unique $y$ tel que pour tout $x$, $f_0(x) = f_1(x * y)$. Le but est de retrouver $y$.

Cela correspond à une recherche de période de la fonction $g$ définie par $g(b,x) = f_b(x)$, sur le groupe $\mathbb{Z}_2 \ltimes \mathbb{G}$ (le produit semi-direct de $G$ par $\mathbb{Z}_2$), qui est le groupe $\mathbb{Z}_2 \times \mathbb{G}$ muni de l'opération de groupe $\boxplus : (b_0, x_0) \boxplus (b_1, x_1) = (b_0 \oplus b_1, x_0 * (-1)^{b_0} x_1)$. Le sous-groupe que l'on veut récupérer devient alors $\langle (1,y) \rangle = \{(0,0), (1,y)\}$.

Si $\mathbb{G} = \mathbb{Z}_2^n$, $\mathbb{Z}_2 \ltimes \mathbb{Z}_2^n \simeq \mathbb{Z}_2^{n+1}$, et on peut appliquer l'algorithme de Simon. Dans le cas général, le groupe n'est pas abélien, et des algorithmes pour résoudre ce problème sont présentés en section 5.

## 2.5 Algorithme de Grover

### 2.5.1 Principes

L'algorithme de Grover [13] pour la recherche non structurée est un autre algorithme emblématique de l'ordinateur quantique.

Le problème est le suivant : On dispose d'une fonction de test $f$, qui permet de déterminer si un élément répond à un critère donné (on parle d'élément marqué par $f$). L'objectif est de retrouver un élément vérifiant ce critère, soit un $x$ tel que $f(x) = 1$.

**Remarque 3.** Si la fonction de test s'exécute (classiquement) en temps polynomial, obtient la classe des problèmes NP.

L'algorithme de Grover permet de résoudre ce problème, en $\Theta(\sqrt{\frac{N}{M}})$ tests si l'ensemble de départ a $N$ éléments, dont $M$ vérifient le critère, là où un algorithme classique, sans hypothèses supplémentaires, ne peut que tester successivement des valeurs, et donc aura besoin de $\Theta(\frac{N}{M})$ itérations pour trouver un élément. La version la plus simple et la plus efficace a besoin de connaître au préalable le nombre d'éléments solutions, mais une variante sans connaissance préalable du nombre de solution existe, et a une complexité sensiblement plus grande. Si $M \geq \frac{N}{4}$, une autre variante existe, qui permet de récupérer une superposition de toutes les solutions en une seule requête [8]. On se restreint donc au cas $M < \frac{N}{4}$

Il fonctionne comme suit :

A partir de la fonction de test, on crée un oracle de test $\mathcal{O}_f : |x\rangle \mapsto (-1)^{f(x)} |x\rangle$, qui inverse l'amplitude des éléments marqués.

On utilise de plus l'opérateur de diffusion $H^{\otimes n} Z_n H^{\otimes n}$, avec $Z_n : |0\rangle \mapsto -|0\rangle, |x \neq 0\rangle \mapsto |x\rangle$. Cet opérateur a pour effet de symétriser l'amplitude des éléments autour de la moyenne des amplitudes : $H^{\otimes n} Z_n H^{\otimes n} \sum_x c_x |x\rangle = \sum_x (c + (c - c_x)) |x\rangle$, avec $c$ la moyenne (arithmétique) des $c_x$.

On peut ainsi créer l'opérateur de Grover $G = H^{\otimes n} Z_n H^{\otimes n} \mathcal{O}_f$. Il va inverser l'amplitude des éléments marqués (en minorité), puis faire une symétrie autour de la moyenne. Si on part d'un état où tous les éléments ont la même amplitude, les éléments marqués étant peu nombreux, la moyenne sera proche de l'amplitude des éléments non marqués, et cela aura donc pour effet de diminuer l'amplitude des éléments non marqués, tout en augmentant celle des éléments marqués. On peut alors itérer cela, et tant que l'amplitude des éléments non marqués n'est pas trop faible, les éléments marqués étant minoritaire, la moyenne diminue lors de l'inversion d'amplitude, mais ne change pas de signe. On finit donc par obtenir une superposition d'états marqués, avec un très faible bruit dû au résidu d'amplitude des états non marqués.

**Remarque 4.** Cet opérateur fait des cycles (c'est en fait une rotation) : si on l'applique trop, la symétrie fera baisser l'amplitude des éléments marqués, pour finir par revenir à l'état équisuperposé initial. C'est pourquoi il importe de connaître le nombre d'éléments marqués.

Le nombre optimal d'itérations est $\left\lfloor \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rfloor$ pour $N \gg M$ (voir section 2.5.2).

L'algorithme de Grover est donc :
- Partir de l'état $\sum_x |x\rangle$,
- appliquer $\left\lfloor \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rfloor$ fois l'opérateur $H^{\otimes n} \mathcal{O}_0 H^{\otimes n} \mathcal{O}_f$,
- mesurer.

Pour $M = 1$, cet algorithme permet donc de rechercher une valeur en $\frac{\pi}{4}\sqrt{N}$ appels à un oracle de test, là où un algorithme classique a besoin de $\frac{N}{2}$ appels en moyenne à une fonction de test. De plus, sans hypothèse supplémentaire sur la fonction de test, cet algorithme est optimal [24].

### 2.5.2 Analyse de complexité

L'algorithme de Grover consiste en l'itération de l'opérateur de Grover, $H^{\otimes n} \mathcal{O}_0 H^{\otimes n} \mathcal{O}_f$.

Si on regarde cet opérateur dans la base orthonormale formée par les deux vecteurs $\left|y^\perp\right\rangle = \frac{1}{\sqrt{N-M}} \sum_x^{f(x)=0} |x\rangle$ et $|y\rangle = \frac{1}{\sqrt{M}} \sum_x^{f(x)=1} |x\rangle$, on obtient la matrice

$$\begin{pmatrix} 1 - \frac{2M}{N} & -2\frac{\sqrt{M(N-M)}}{N} \\ 2\frac{\sqrt{M(N-M)}}{N} & 1 - \frac{2M}{N} \end{pmatrix}$$

On remarque que c'est une matrice de rotation, d'angle $\theta = 2 \arcsin\left(\sqrt{\frac{M}{N}}\right) \simeq 2\sqrt{\frac{M}{N}}$ (on suppose $M \ll N$), et que l'état équisuperposé $\frac{1}{\sqrt{N}} \sum_x |x\rangle = \sqrt{\frac{M}{N}} |y\rangle + \sqrt{\frac{N-M}{N}} \left|y^\perp\right\rangle$ est le vecteur $\left|y^\perp\right\rangle$ après une rotation d'angle $\frac{\theta}{2}$ dans ce plan.

*Démonstration.*

$$\begin{pmatrix} \sqrt{\frac{N-M}{N}} & -\sqrt{\frac{M}{N}} \\ \sqrt{\frac{M}{N}} & \sqrt{\frac{N-M}{N}} \end{pmatrix}^2 = \begin{pmatrix} 1-\frac{2M}{N} & -2\frac{\sqrt{M(N-M)}}{N} \\ 2\frac{\sqrt{M(N-M)}}{N} & 1-\frac{2M}{N} \end{pmatrix}$$

□

On peut donc calculer le nombre de rotations $k$ à appliquer à cet état pour obtenir $|y\rangle$. On doit avoir $k\theta + \frac{\theta}{2} = \frac{\pi}{2}$, soit $k \simeq \frac{\pi}{4}\sqrt{\frac{N}{M}}$. En pratique, ce nombre n'étant pas nécessairement entier, on obtient un état très proche de $|y\rangle$ (et on mesurera donc un élément marqué, avec une probabilité d'erreur négligeable).

L'erreur angulaire entre le vecteur voulu et le vecteur obtenu est plus petite que $\theta/2$, on mesurera donc la bonne valeur avec une erreur plus petite que

$$\sin^2(\theta/2) = M/N.$$

# 3 Cryptanalyse quantique symétrique

La cryptanalyse utilise les outils algorithmiques avec des objectifs spécifiques. Notamment, la comparaison d'un attaque sur un système donné se fait par rapport au coût de la meilleure méthode générique. Ce coût est un nombre, dont l'unité est généralement le coût d'un chiffrement (ou d'un déchiffrement, ou d'une application de la fonction de hachage, selon le problème). On ne peut donc se contenter d'un $O(f(n))$ pour évaluer l'intérêt d'une attaque, et il faut pouvoir donner un coût précis, au moins pour la partie dominante de l'algorithme.

Des exemples d'utilisation des algorithmes quantiques de la section précédente en cryptanalyse sont présentés ci-dessous.

## 3.1 Utilisation de Grover

L'oracle de test utilisé par l'algorithme de Grover étant quelconque, il on peut chercher à savoir si un algorithme quantique donné peut facilement être utilisé dedans. En effet, la plupart des algorithmes minimisent la partie effectuée quantiquement, en faisant parfois d'important calculs classiques, ou en utilisant une mémoire classique conséquente. Ce genre de compromis ne peut généralement pas être fait dans l'oracle. Cela importe notamment en cryptanalyse, où de nombreuses attaques génèrent des candidats en fonction de divers paramètres, puis testent ces candidats. Il est donc intéressant de connaître le comportement d'une version purement quantique des algorithmes, qui génèrent des qubits dans un état (ou une superposition) répondant au critère voulu.

L'algorithme de Grover, par exemple, ne fait intervenir aucune mesure (sauf à la toute fin), et est donc purement quantique. L'algorithme de Simon utilise un post-traitement classique [1] (la résolution d'un système linéaire), et il faudrait donc rajouter un circuit quantique de résolvant ce système pour en faire une version purement quantique. De plus, on doit passer d'un temps variable à une probabilité d'erreur donnée, puisque sans mesure, le temps est nécessairement constant. Il faut donc bien choisir les paramètres (pour Simon, le nombre d'échantillons) en fonction de la probabilité de réussite souhaitée. Ce genre de transformation est toujours faisable pour un algorithme, mais cela empêche de faire des optimisations (tous les calculs se faisant quantiquement), et peut nécessiter une mémoire quantique beaucoup plus importante.

## 3.2 Even-Mansour

Even-Mansour [10] est une construction de chiffrement, à partir d'une bijection $P$ (publique) aléatoire. il est paramétré par deux clés, $k_1$ et $k_2$, et est défini par $EM_{k_1,k_2}(m) = P(m \oplus k_1) \oplus k_2$.

1. La routine présentée en section 2.4.2 utilise deux mesures, la première ne sert qu'à rendre les calculs plus lisibles, et la seconde à revenir dans le monde classique

Classiquement, si $P$ (ainsi que $k_1$ et $k_2$) sont choisies aléatoirement, la théorie de l'information donne une borne sur la complexité d'une attaque : si $E$ est le nombre de couples clair/chiffrés et $n$ le nombre de bits d'un message, le temps d'une attaque de récupération des clés est au minimum de $2^n/E$, l'unité étant le coût d'une application de $P$. Cela donne, si on considère que récupérer un couple a un coût, une complexité minimale en temps de $2^{n/2}$. Une variante à une seule clé (avec $k_1 = k_2$) existe, et offre la même sécurité.

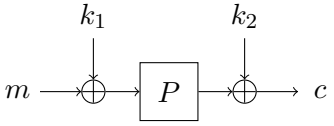



FIGURE 1 – Le chiffrement Even-Mansour.

Cependant, les choses sont différentes quantiquement. Kuwakado et Morii [18] ont montré une attaque quantique, avec la fonction

$$\begin{array}{rccl} f: & \{0,1\}^n & \to & \{0,1\}^n \\ & x & \mapsto & EM_{k_1,k_2}(x) \oplus P(x) = P(x \oplus k_1) \oplus P(x) \oplus k_2. \end{array}$$

Cette fonction satisfait $f(x) = f(x \oplus k_1)$, et on peut appliquer l'algorithme de Simon pour récupérer $k_1$. Une fois cela fait, on obtient $k_2$ en une requête classique, car pour tout $x$, $k_2 = EM_{k_1,k_2}(x) \oplus P(x \oplus k_1)$. Le coût est celui de l'algorithme se Simon, soit $4n$ échantillons qui coûtent chacun 2 permutations, ce qui revient à $8n$ chiffrements, et la résolution classique d'un système linéaire qui peut se faire en $O(n^3)$ opérations pour une petite constante, ce que l'on peut se permettre de négliger.

### 3.3 Slide attacks

Les *slide attacks* sont une forme de cryptanalyse introduite en 1999 par Biryukov et Wagner [6]. Elle s'applique à des chiffrements qui ont une structure itérative simple, typiquement, l'application successive de la même fonction $F_k$, dite fonction de tour (qui est paramétrée par une clé $k$, mais ne change pas en fonction du tour). Le chiffrement peut ainsi s'écrire $E_k = F_k^r$ (pour la composition). L'idée derrière ces systèmes est d'itérer une fonction simple à calculer, qui ne possède pas des propriétés

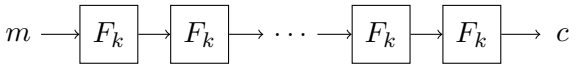



FIGURE 2 – Chiffrement itératif vulnérable à une slide attack.

très fortes, en espérant qu'un nombre suffisant de tours rendra toute attaque impraticable.

Le principe de cette attaque est de rechercher deux messages $m_1, m_2$ tels que $m_2 = F(m_1)$. On aura alors les chiffrés associés, $c_1$ et $c_2$, qui vérifient aussi $c_2 = F(c_1)$. Si la fonction $F$ est suffisamment simple, on peut facilement vérifier si les deux égalités $m_2 = F(m_1)$ et $c_2 = F(c_1)$ sont compatibles (ce qui donne un critère pour trouver la paire), puis extraire la clé. Le grand intérêt est que contrairement à la plupart des attaques, elle ne dépend pas du nombre de tours. Des généralisations à des systèmes où la clé n'est pas toujours la même, mais suit un motif régulier ont été développées par la suite [7]. La complexité est celle de la recherche et de l'identification de tels messages, et dépend entièrement de la fonction $F$. En faisant une recherche exhaustive sur les messages, on peut espérer trouver une telle paire avec $2^{n/2}$ messages, ce à quoi il faut ajouter le coût de test. La recherche peut parfois se faire sur un espace plus restreint (typiquement dans le cas d'un réseau de Feistel, où à chaque tour, seule la moitié des bits est affectée par la fonction), ce qui permet d'abaisser encore la complexité de l'attaque.

Une variante quantique de ces attaques a été trouvée en 2016 [15], si la fonction $F$ se décompose en $F(x) = P(x \oplus k)$, avec $P$ une bijection connue. Le chiffrement s'écrit alors $E_k(x) = F(x)^r \oplus k$ (La dernière itération de $P$ pouvant toujours être inversée, on l'enlève, car elle n'apporte rien). Classiquement, si $P$ est aléatoire, la théorie de l'information donne la même borne que pour Even-Mansour.

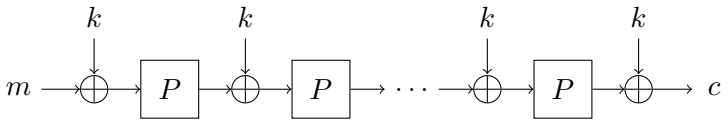



FIGURE 3 – Chiffrement itératif vulnérable à une slide attack quantique.

On peut donc créer la fonction

$$f : \begin{array}{ccccc} \{0,1\} & \times & \{0,1\}^n & \to & \{0,1\}^n \\ b & , & x & \mapsto & \begin{cases} P(E_k(x)) \oplus x & \text{si } b = 0 \\ E_k(P(x)) \oplus x & \text{si } b = 1 \end{cases} \end{array} .$$

Le chiffrement vérifie, pour tout $x$, $P(E_k(x)) \oplus k = E_k(P(x \oplus k))$. $f$ vérifie donc $f(0,x) = P(E_k(x)) \oplus x = E_k(P(x \oplus k)) \oplus k \oplus x = f(1, x \oplus k)$. Elle a pour période $(1,k)$. Cependant, il se peut que deux indices vérifient $f(b_1, x_1) = f(b_2, x_2)$, avec $(b_1 \oplus b_2, x_1 \oplus x_2) \neq (1,k)$. Comme indiqué dans la remarque 2, ce n'est pas un problème, mais cela peut ralentir l'algorithme si certains écarts sont trop présents. De tels cas sont extrêmement rares pour des fonctions aléatoires, et en cryptographie, les fonctions sont construites pour éviter d'avoir de telles propriétés, car elles peuvent servir de base à une attaque.

Une analyse précise du comportement de l'algorithme de Simon avec ces périodes parasites est faite dans le papier original[15], dont la conclusion est que si quelque soit $b$, la proportion d'éléments vérifiant $f(x \oplus b) = f(x)$ est plus petite que $p_0$, l'algorithme de Simon utilisant $cn$ échantillons échoue avec une erreur plus petite que $\left(2\left(\frac{1+p_0}{2}\right)^c\right)^n$. En prenant $c = 4$, ce qui est une borne très large, la complexité est la même que pour l'attaque précédente, soit $8n$ chiffrements, plus une résolution classique d'un système linéaire.

### 3.4 Cryptanalyse de Treyfer

Treyfer est un chiffrement par bloc conçu pour avoir une implémentation très compacte, et est un des premiers chiffrements à avoir été victime des slide attacks (classiques). Ce chiffrement transforme 8 blocs de 8 bits($t_i$), utilise une clé de 8 fois 8 bits ($k_i$), et une *S-box*, une bijection publique, aléatoire sur 8 bits ($S$). L'algorithme utilise l'addition sur 8 bits, une rotation des bits vers la gauche ($\lll$) et un registre auxiliaire, $t_9$.

Le chiffrement Treyfer

**for** r := 1 to 32 **do**
  $t_9 \leftarrow t_1$
  **for** i := 1 to 8 **do**
    $t_{i+1} \leftarrow (t_{i+1} + S(k_i + t_i)) \lll 1$
  **end for**
  $t_1 \leftarrow t_9$
**end for**

L'algorithme consiste en 32 applications de la même fonction, qui utilise une addition modulo $2^8$ de la clé. L'objectif initial était de trouver deux fonctions utilisant le chiffrement et ayant pour décalage une partie des $k_i$ (fonctions pouvant éventuellement dépendre de certains paramètres), l'idée étant de créer un oracle qui, en fonction de paramètres, génère des fonctions candidates pour un algorithme résolvant une instance de sous-groupe caché dont le résultat serait une partie des clés, que l'on peut ensuite tester à partir de quelques couples clair/chiffré. On peut enfin mettre le tout dans l'algorithme de Grover pour trouver les bons paramètres, ce qui donnerait la clé.

Il y a donc trois points importants pour monter une telle attaque quantique :
- évaluer l'efficacité d'un algorithme quantique pour résoudre le problème du décalage caché dans un groupe de la forme $\mathbb{Z}_{2^8}^r$, pour $r \leq 8$,
- ces algorithmes étant probabilistes, évaluer l'effet d'un oracle imparfait sur l'algorithme de Grover,

– créer deux fonctions répondant au mieux aux critères voulus.

Les deux premiers points sont adressés dans les sections suivantes, et le dernier point n'a pour l'instant pas abouti. Des exemples de constructions sûres classiquement mais attaquables quantiquement sont néanmoins présentées dans la partie 6.

# 4 Oracles de Grover

## 4.1 Oracles standard

L'oracle utilisé dans l'algorithme de Grover est un système de test, qui à un ensemble d'arguments associe un entier, 0 ou 1. Cependant, certains systèmes de test classiques ne sont pas déterministes, et peuvent échouer. C'est aussi le cas de systèmes quantiques comme l'algorithme de Simon, qui renvoie la bonne réponse avec une très forte probabilité. On peut donc s'intéresser au comportement de l'algorithme de Grover, quand l'oracle est bruité.

Il faut tout d'abord formaliser ce bruit. Pour cela, on doit rentrer plus en détail sur la façon de le construire. Si on dispose d'un circuit quantique $O_f : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$ pour une certaine fonction de test $f$, la manière canonique de construire un oracle de Grover est d'appliquer $O_f$ à $|x\rangle |-\rangle$. En effet, si $f(x) = 0$, l'état reste inchangé, et si $f(x) = 1$, le dernier qubit devient $\frac{|0\oplus1\rangle - |1\oplus0\rangle}{\sqrt{2}} = \frac{|1\rangle - |0\rangle}{\sqrt{2}} = -|-\rangle$. On obtient bien $-|x\rangle |-\rangle$, et les deux registres étant séparables, l'opérateur $|x\rangle |-\rangle \mapsto O_f |x\rangle |-\rangle$ a bien l'effet escompté.

On peut ne pas avoir de circuit faisant $|x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$, mais simplement un circuit faisant $A_f : |x\rangle |0\rangle |0\rangle \mapsto |x\rangle |z(x)\rangle |f(x)\rangle$ ($y(x)$ correspond aux variables intermédiaires calculées pour obtenir $f(x)$). On peut se ramener au cas précédent en utilisant l'*uncomputing* : on utilise l'opérateur $A_f^\dagger (\mathbb{1}_{n-1} \otimes Z) A_f$, qui inverse la phase des éléments tels que $f(x) = 1$, ce qui est bien ce que l'on veut.

## 4.2 Oracles probabilistes

Un algorithme probabiliste va partir de paramètres initiaux ($|x\rangle$), tirer un certain nombre de valeurs (typiquement en utilisant des $H^{\otimes n} |0\rangle$ pour un tirage uniforme), puis calculer un résultat à partir de ces éléments. Un circuit typique fera donc l'opération $B_f : |x\rangle |0\rangle |0\rangle \mapsto |x\rangle \sum_i |y_i(x)\rangle |g(x, y_i(x))\rangle$, avec les $y_i(x)$ dépendant de $x$ et de valeurs tirées, et $g(x, y_i(x))$ valant souvent 0 si $f(x) = 0$, et souvent 1 si $f(x) = 1$. On note $P_e(x)$ la probabilité que $g(x, y_i(x)) \neq f(x)$.

Si on souhaite utiliser un algorithme probabiliste en tant qu'oracle, on peut utiliser la même construction qu'avant, $B_f^\dagger (\mathbb{1}_{n-1} \otimes Z) B_f$, mais le résultat ne sera pas tout à fait le même. C'est un modèle d'erreur dans lequel l'oracle donne la bonne réponse, à un bruit près. D'autres modèles d'erreurs considèrent une réponse ferme de l'oracle (0 ou 1, sans intrication avec des qubits auxiliaires), qui peut être erronée[2, 21]. Dans ces cas, on perd l'intérêt de cet algorithme, avec un temps de convergence linéaire en le nombre d'états possibles, au lieu d'une racine carrée.

**Proposition 3.** L'algorithme de Grover sur un espace de taille $N$ utilisant un oracle quantique marquant $M$ éléments dont l'erreur, pour chaque élément, est plus petite que $P_e$ a une probabilité de réussite plus grande que

$$\left(1 - \frac{M}{N}\right)\left(1 - 2P_e\right)^{\frac{\pi}{2}\sqrt{\frac{N}{M}}}.$$

*Démonstration.* L'algorithme part d'un état $|x\rangle |0\rangle |0\rangle$. On suppose que $f(x) = 1$. La première application de $B_f$ donne $|x\rangle \sum_i |y_i(x)\rangle |g(x, y_i(x))\rangle$. On peut décomposer cet état en

$$|x\rangle \sqrt{1 - P_e(x)} \sum_i^{g(x,y_i(x))=1} |y_i(x)\rangle |1\rangle = |x\rangle |y_{ok}(x)\rangle |1\rangle$$

(la partie pour laquelle l'algorithme donne le bon résultat) et

$$|x\rangle \sqrt{P_e(x)} \sum_i^{g(x,y_i(x))=0} |y_i(x)\rangle |0\rangle = |x\rangle |y_e(x)\rangle |0\rangle$$

(la partie problématique). On a donc l'état $|x\rangle\,(|y_{ok}(x)\rangle\,|1\rangle + |y_e(x)\rangle\,|0\rangle)$, avec $|y_{ok}(x)\rangle$ qui est de norme $\sqrt{1-P_e(x)}$ et $|y_e(x)\rangle$ de norme $\sqrt{P_e(x)}$.

L'application de $\mathbb{1}_{n-1}\otimes Z$ donne $-|x\rangle\,|y_{ok}(x)\rangle\,|1\rangle + |x\rangle\,|y_e(x)\rangle\,|0\rangle$. On ne peut pas directement donner le résultat de $B_f^\dagger$, mais on sait que $B_f^\dagger(|x\rangle\,(|y_{ok}(x)\rangle\,|1\rangle + |y_e(x)\rangle\,|0\rangle)) = |x\rangle\,|0\rangle\,|0\rangle$. On a donc $B_f^\dagger(-|x\rangle\,|y_{ok}(x)\rangle\,|1\rangle + |x\rangle\,|y_e(x)\rangle\,|0\rangle) = -|x\rangle\,|0\rangle\,|0\rangle + 2B_f^\dagger(|x\rangle\,|y_e(x)\rangle\,|0\rangle)$.

De même, si $f(x)=0$, on obtient

$$B_f^\dagger(\mathbb{1}_{n-1}\otimes Z)B_f\,|x\rangle\,|0\rangle\,|0\rangle = |x\rangle\,|0\rangle\,|0\rangle - 2B_f^\dagger(|x\rangle\,|y_e(x)\rangle\,|1\rangle).$$

On a donc un opérateur qui fait presque ce qu'il faut, et rajoute un bruit. Pour évaluer l'impact de ce bruit, on considère l'écart entre le comportement de l'algorithme de Grover avec cet oracle, et l'algorithme avec un oracle parfait. La seule différence est l'ajout d'un vecteur dont on connaît la norme (car elle ne dépend que de celle des $|x\rangle$, de $P_{fp}$ et de $P_{fn}$). Les autres opérateurs de l'algorithme étant linéaires, ils ne vont pas affecter le bruit, qui restera un vecteur normal au vecteur théorique idéal. On peut ainsi calculer à chaque étape quelle proportion du vecteur que l'on obtient se comporte correctement, et quelle proportion est inutilisable. De plus, si on fait une mesure finale, on peut détecter certaines erreurs, si les qubits auxiliaires ne sont pas tous à 0.

A une étape de l'algorithme, on obtient le vecteur $|u\rangle$ en sortie de l'oracle parfait. En sortie de l'oracle imparfait, on obtient le vecteur $|v\rangle = |u\rangle + |b\rangle$, avec $|b\rangle$ le petit vecteur de bruit. En décomposant $|v\rangle$ en une partie proportionnelle à $|u\rangle$ et une partie perpendiculaire, on peut calculer ce que l'on perd à chaque appel, et donc ce que l'on perdra au total.

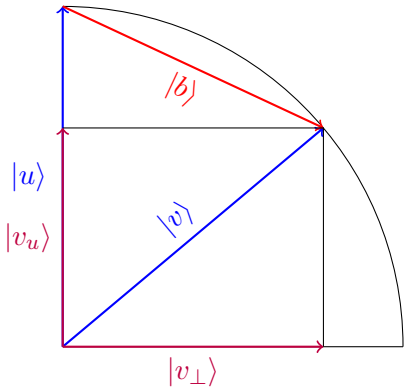



FIGURE 4 – Vecteur après un passage dans l'oracle

Si $b$ est la norme de $|b\rangle$, un peu de géométrie élémentaire nous indique que $|v_u\rangle$ a pour norme $1-b^2/2$, et donc que la norme de $v_\perp$ est de $b\sqrt{1-b^2/4}$ (on considère $b<\sqrt{2}$, ce qui est le cas lorsque pour tout $x$, $P_e(x)<1/2$). Il reste donc à calculer la norme de $b$ à chaque étape. $B_f^\dagger$ étant unitaire, $|b\rangle$ a la même norme que le vecteur avant transformation. Si on note $l_x$ l'amplitude de l'élément $x$, on a $b^2 = 4\sum_x l_x^2 P_e(x)$. Les opérateurs étant linéaires, si on part d'un état déjà bruité, le vecteur correct restant sera divisé de la même façon, et le vecteur de bruit, qui est normal au vecteur correct, n'interagit pas.

Si on note $N$ le nombre d'éléments, dont $M$ sont marqués, on a une probabilité de réussite finale de

$$P_r = \prod_{i=0}^{\frac{\pi}{4}\sqrt{N/M}} \left(1 - 2\sum_x \chi_i(x)P_e(x)\right)^2$$

avec

$$\chi_i(x) = \begin{cases} \frac{1}{N-M}\cos^2\left((2i+1)\arcsin\left(\sqrt{\frac{M}{N}}\right)\right) & \text{si} \quad f(x)=0 \\ \frac{1}{M}\sin^2\left((2i+1)\arcsin\left(\sqrt{\frac{M}{N}}\right)\right) & \text{si} \quad f(x)=1 \end{cases}$$

le carré de l'amplitude de chaque élément à l'étape $i$. Il faut rajouter à cela l'erreur de l'algorithme avec un oracle idéal (qui est en $M/N$), que l'on peut généralement négliger si l'objectif est d'obtenir une probabilité finale de réussite constante.

Tout d'abord, on peut remarquer que pour deux distributions d'erreur données $P_e$ et $P'_e$, si pour tout $x$, $P_e(x) < P'_e(x)$, alors $P_r > P'_r$. On peut donc se contenter de majorer les erreurs pour étudier le comportement.

Si cette formule permet de calculer explicitement les probabilités de réussite à partir de la distribution des $P_e(x)$, on peut regarder le comportement dans différents modèles.

Si $P_e(x) = P_e$ pour tout $x$, alors $b^2 = 4P_e$. On a donc une perte de $1 - 2P_e$ à chaque étape. Il y a $\frac{\pi}{4}\sqrt{\frac{N}{M}}$ étapes, ce qui donne une probabilité finale de réussite de $(1-2P_e)^{\frac{\pi}{2}\sqrt{\frac{N}{M}}}$, à quoi il faut rajouter le taux d'erreur de l'algorithme de Grover parfait, qui est de $\frac{M}{N}$. □

**Remarque 5.** Cela impose une probabilité d'erreur en $\sqrt{\frac{M}{N}}$ pour obtenir une probabilité de réussite constante.

On peut aussi regarder d'autres distributions d'erreur. Si $P_e(x) = P_e$ si $f(x) = 1$, et $P_e(x) = 0$ sinon (on n'a que des faux négatifs), $b^2$ vaut, à l'étape $i$ de l'algorithme, $4P_e \sin^2\left((2i+1)\arcsin\left(\sqrt{\frac{M}{N}}\right)\right)$, ce qui donne une probabilité finale en

$$\prod_{i=0}^{\frac{\pi}{4}\sqrt{\frac{N}{M}}} \left(1 - 2P_e \sin^2\left((2i+1)\arcsin\left(\sqrt{\frac{M}{N}}\right)\right)\right)^2.$$

Des calculs pénibles permettent de savoir que l'on obtient toujours une probabilité en $\sqrt{\frac{M}{N}}$ pour obtenir une probabilité constante, mais l'erreur initiale peut être plus grande, à une constante près. Intuitivement, durant la moitié de l'algorithme, la norme du vecteur des éléments marqués sera grande (plus grande que $1/\sqrt{2}$), et donc les erreurs venant le lui seront significatives. On obtient un résultat similaire si l'on n'a que des faux positifs.

On doit donc chercher des profils d'erreur pour les algorithmes dans l'oracle en $\sqrt{\frac{M}{N}}$, ce qui va générer un surcoût logarithmique (en $N/M$) par rapport à une probabilité d'erreur constante (qui est généralement ce que l'on choisit). On peut de plus calculer explicitement l'erreur acceptable pour l'oracle permettant d'obtenir une probabilité de réussite donnée. On peut donc calibrer un algorithme quantique probabiliste comme celui de Simon pour l'utiliser comme composant d'un oracle.

S'il l'on souhaite optimiser plus avant un tel algorithme, on peut faire évoluer l'oracle de test au fil des étapes. Par exemple, s'il n'a que des faux négatifs, il peut se permettre d'être moins précis au début, puis devenir de plus en plus précis lorsque l'amplitude des éléments marqués augmente.

# 5 Algorithmes pour le décalage caché

On souhaite créer des fonctions ayant un décalage caché pour cryptanalyser Treyfer. Ce chiffrement utilisant de nombreuses additions modulaires (et aucun xor), on peut s'attendre à ce que le décalage soit aussi une addition modulaire. Il convient donc d'étudier les algorithmes résolvant ce problème, et le cas échéant les adapter à la structure probable $\mathbb{Z}_{2^8}^r$ qu'auraient nos fonctions. Il faut de plus prendre en compte, comme pour l'algorithme de Simon, la possibilité que les fonctions ne soient pas injectives.

## 5.1 Algorithmes existants

Le problème du sous groupe caché dans des groupes non abéliens a été très étudié, et de nombreux résultats existent sur les produits semi-directs, en particulier pour $\mathbb{Z}_2 \ltimes \mathbb{Z}_n$, noté $D_n$, qui est le groupe dihédral d'ordre $n$.

Pour le groupe dihédral, Ettinger et Høyer ont montré en 1999 dans [9] qu'un nombre linéaire (en $\log(n)$) de requêtes suffisait pour récupérer le sous-groupe. Cependant, un post-traitement classique en temps exponentiel restait nécessaire. Bacon et al. ont montré dans [4] qu'on ne pouvait pas faire mieux qu'un nombre linéaire de requêtes pour espérer récupérer un bit du générateur du sous-groupe.

Regev a donné, en 2003, une réduction de ce problème vers celui de la somme de sous-ensemble, et vers le problème du plus petit vecteur d'un réseau [19]. Cela laisse donc à penser qu'il n'existe pas d'algorithme à temps polynomial résolvant ce problème (ou s'il existe, ses conséquences seraient très importantes, notamment en cryptographie post-quantique). Kuperberg a présenté en 2003 un algorithme sous-exponentiel en temps et en mémoire, en $\tilde{O}(3^{\sqrt{n}})$ pour résoudre ce problème [16]. Regev en a ensuite proposé une variante polynomiale en mémoire quantique [20] (mais avec un temps et une mémoire classique sous-exponentielle). Kuperberg en a proposé une synthèse 10 ans plus tard, avec une version paramétrable en fonction de la quantité de mémoire quantique dont on dispose [17].

D'autres variations ont aussi été étudiées. Friedl et al. ont par exemple présenté en 2003 un algorithme pour résoudre le problème dans le groupe $\mathbb{Z}_2 \ltimes \mathbb{Z}_p^n$ avec $p$ premier, dont la complexité temporelle est polynomiale en $n$ (mais exponentielle en $p$) [11] . Un autre algorithme polynomial a été présenté en 2007 par Inui et Le Gall, pour le groupe $\mathbb{Z}_p \ltimes \mathbb{Z}_{p^r}^n$, avec $p$ premier impair [14].

La plupart de ces algorithmes reposent sur des principes similaires (et proches de ceux de l'algorithme de Simon), qui sont expliquées ci-dessous.

### 5.1.1 Échantillonnage

Pour plus de simplicité dans l'explication, on se place dans le cas du groupe $(\mathbb{Z}_{2^n}, +)$. Pour rappel, on dispose de deux fonctions $f_0, f_1 : \mathbb{Z}_{2^n} \to \mathcal{E}$ (l'ensemble d'arrivée n'a aucune importance, les 2 fonctions doivent simplement être injectives), telle que $f_0(x) = f_1(x + y)$, pour un certain $y$. On suppose disposer de plus d'un opérateur $O_f : |b\rangle |x\rangle |y\rangle \mapsto |b\rangle |x\rangle |y \oplus f_b(x)\rangle$.

Ces algorithmes reposent sur une routine dite d'échantillonnage, qui fonctionne ainsi :
- Partir de l'état $\sum_{b,x} |b\rangle |x\rangle |0\rangle$,
- appliquer $O_f$, ce qui donne $\sum_{b,x} |0\rangle |x\rangle |f_b(x)\rangle$
- (optionnel) mesurer le 3eme registre, pour obtenir un $f_0(x_0)$, et l'état $|0\rangle |x_0\rangle + |1\rangle |x_0 + y\rangle$
- appliquer la transformée de Fourier quantique sur le second registre, ce qui donne

$$\sum_l \exp\left(2i\pi \frac{x_0 l}{2^n}\right) |0\rangle |l\rangle + |1\rangle \exp\left(2i\pi \frac{(x_0+y)l}{2^n}\right) |1\rangle |l\rangle$$

- mesurer le second registre, pour obtenir un $l$, et un qubit dans l'état

$$\exp\left(2i\pi \frac{x_0 l}{2^n}\right) \left(|0\rangle + \exp\left(2i\pi \frac{yl}{2^n}\right) |1\rangle\right)$$

Cette routine permet donc de récupérer des états $|\psi_l\rangle = |0\rangle + \exp\left(2i\pi \frac{yl}{2^n}\right) |1\rangle$ pour un certain $l$ connu, à une phase globale près. On cherche alors à récupérer de l'information sur $y$, à partir de suffisamment d'échantillons $|\psi_l\rangle$.

**Remarque 6.** $|\psi_0\rangle = |0\rangle + |1\rangle$ ne contient pas d'information sur $y$, et n'a donc aucun intérêt ici.

**Remarque 7.** $|\psi_{2^{n-1}}\rangle = |0\rangle + \exp(i\pi y) |1\rangle = |0\rangle + (-1)^y |1\rangle$. Cet état permet de récupérer le bit de poids faible de $y$, $y_0$ ($H |\psi_{2^{n-1}}\rangle = |y \mod 2\rangle$).

### 5.1.2 Combinaisons d'éléments

On peut de plus combiner des échantillons, à l'aide d'un CNOT :

$$\text{CNOT} |\psi_{l_1}\rangle |\psi_{l_2}\rangle = |00\rangle + \exp(2i\pi \frac{y(l_1+l_2)}{2^n}) |10\rangle + \exp(2i\pi s \frac{l_2}{2^n}) \left(|01\rangle + \exp(2i\pi y \frac{l_1 - l_2}{2^n}) |11\rangle\right)$$

Le premier qubit devient, si l'on mesure 0 dans le second qubit, $|0\rangle + \exp(2i\pi \frac{y(l_1+l_2)}{2^n}) |1\rangle = |\psi_{l_1+l_2}\rangle$. Si l'on mesure 1, il devient $|0\rangle + \exp(2i\pi y \frac{l_1-l_2}{2^n}) |1\rangle = |\psi_{l_1-l_2}\rangle$. On a donc une méthode pour combiner les échantillons, afin d'en obtenir de plus intéressants, mais où l'on ne connaît l'opération effectuée ($+$ ou $-$) qu'après-coup. Les deux résultats de la mesure étant équiprobables, on a autant de chances de faire une addition qu'une soustraction.

A partir de cela, il y a deux approches : chercher à atteindre un élément qui contient toute l'information, comme $|\psi_1\rangle$, et utiliser diverses transformations et mesures pour en extraire $y$, ou chercher à atteindre $|\psi_{2^{n-1}}\rangle$ pour en extraire le bit $y_0$ et ensuite itérer l'algorithme, avec les fonctions $f'_0(x) = f_0(2x)$ et $f'_1(x) = f_1(2x + y_0)$, qui ont un décalage caché de $y' = (y - y_0)/2$ dans le groupe $(\mathbb{Z}_{2^{n-1}}, +)$. On se concentrera ici sur la seconde approche.

**Définition 1.** Pour $x$ dans $\mathbb{Z}_{2^n}$, on définit la hauteur de $x$ par

$$h(x) = \max\left\{i \in [0; n] \;\middle|\; 2^i \text{ divise } x\right\}.$$

Elle correspond au nombre de zéros consécutifs à droite dans l'écriture en base 2.

**Remarque 8.**

1. $h(x) = n \Rightarrow x = 0$
2. $h(x) = n - 1 \Rightarrow x = 2^{n-1}$

**Remarque 9.** Combiner deux éléments de même hauteur n'est jamais défavorable : si $l_1 = 2^i(1+2m_1)$ et $l_2 = 2^i(1 + 2m_2)$, $h(l_1 + l_2) = h(2^i(2 + 2m_1 + 2m_2)) = i + 1 + h(m_1 + m_2 + 1)$ et $h(l_1 - l_2) = h(2^i(2m_1 - 2m_2)) = i+1+h(m_1 - m_2)$. On augmente donc toujours la hauteur d'au moins 1, l'objectif étant d'atteindre l'élément de hauteur $n - 1$. De plus, si $m_1$ et $m_2$ ont les mêmes bits de poids faible (et donc si $l_1$ et $l_2$ ont plus d'un bit de poids faible en commun), la soustraction sera plus favorable encore, et la hauteur augmentera alors du nombre de bits égaux [2].

Le premier algorithme de Kuperberg[16] utilise la méthode de combinaison ci-dessus, et choisit les éléments à combiner en prenant, parmi les éléments de plus faible hauteur, ceux qui ont le plus grand nombre de bit de poids faible en commun.

Les deux algorithmes suivant[20, 17] utilisent des méthodes de combinaison plus complexes (qui nécessitent des résolutions classiques du problème de la somme de sous-ensemble) qui permettent de réduire les coûts en temps et en mémoire quantique. Ces algorithmes sont moins intéressants dans leur version purement quantique car tous les calculs doivent se faire par l'ordinateur quantique, et une partie des gains en mémoire quantique se fait en jetant les qubits inutiles pour les réutiliser, ce que l'on ne peut pas faire sans observation.

### 5.2 Promesse approchée

Si nos deux fonctions ne sont pas bijectives, on a des éléments $x_0$ et $x_1$ tels que $f_0(x_0) = f_0(x_1) = f_1(x_0 + s) = f_1(x_1 + s)$. Tout comme pour Simon, cela n'introduit pas d'erreur : Le qubit final sera de la forme

$$\left(\exp\left(2i\pi\frac{x_0 l}{2^n}\right) + \exp\left(2i\pi\frac{x_1 l}{2^n}\right)\right)\left(|0\rangle + \exp\left(2i\pi\frac{yl}{2^n}\right)|1\rangle\right).$$

Cela altère la distribution des $l$, qui n'est plus uniforme, et pourrait donc augmenter le nombre d'échantillons requis, en particulier si certains écarts reviennent trop souvent. La probabilité de mesurer un certain $l$ à $f(x_0)$ fixé est proportionnelle à $1 + \exp\left(2i\pi\frac{(x_0-x_1)l}{2^n}\right)$, ce qui va augmenter les chances d'obtenir $l = 0$, en particulier si $x_0 - x_1$ a une hauteur importante. Les autres éléments favorisés sont ceux pour lesquels $(x_0 - x_1)l$ est petit modulo $2^n$, ce qui est le cas de tous $l$ de hauteur plus grande que $n - h$ si $h$ est la hauteur de $x_0 - x_1$, qui auront autant de chances d'apparaître que 0. Dans le cas opposé, si $(x_0 - x_1)$ est de hauteur nulle, la moitié des éléments de $\mathbb{Z}_{2^n}$ seront plutôt favorisés (ceux pour qui $l(x_0 - x_1)$ est plus petit que $2^{n-2}$, ou plus grand que $2^n - 2^{n-2}$), et l'avantage qu'a l'élément nul est négligeable. Intuitivement, la présence de ce léger bruit sera négligeable.

Une autre approche, présentée dans [12], est de prendre un t-uplet $(m_1, \ldots, m_t)$ d'éléments distincts, et construire à partir des fonctions $f_0$ et $f_1$ les fonctions $f'_0 : x \mapsto (f_0(x + m_1), \ldots, f_0(x + m_t))$ et $f'_1 : x \mapsto (f_1(x+m_1), \ldots, f_1(x+m_t))$, qui ont le même décalage, mais auront moins de collisions. Cette construction multiplie le coût de chaque échantillon par $t$, mais peut permettre de rendre injective la fonction. Ce papier borne la probabilité qu'une fonction aléatoire ne devienne pas injective par ce procédé. Dans notre cas, en considérant que les espaces de départ et d'arrivée sont $\mathbb{Z}_{2^n}$, cette probabilité est de $2^{n(2-t/2)}$. On peut ainsi créer des fonctions ayant de très fortes chances d'être injectives.

2. Cela peut se généraliser si $N$ n'est pas une puissance de 2, la hauteur fait alors intervenir les différents diviseurs de $N$, et elle est au moins égale à celle des éléments combinés

## 5.3 Généralisation

On souhaite généraliser ces algorithmes opérant sur $\mathbb{Z}_N$ à des groupes de la forme $\mathbb{Z}_{2^n}^r$.

On a notre fonction $f(b, x_1, \ldots, x_r)$, qui vérifie que $f(0, \cdot)$ et $f(1, \cdot)$ sont injectives, et il existe $(y_1, \ldots, y_r)$ tel que $f(0, x_1, \ldots, x_r) = f(1, x_1 + y_1, \ldots, x_r + y_r)$. On dispose d'un opérateur $O_f$ : $|b\rangle |x_1\rangle \ldots |x_r\rangle |y\rangle \mapsto |b\rangle |x_1\rangle \ldots |x_r\rangle |y \oplus f(b, x_1, \ldots, x_r)\rangle$

### 5.3.1 Échantillonnage

Tout d'abord, il faut adapter la routine d'échantillonnage, qui devient :
- Partir de l'état $\sum_{b,x_1,\ldots x_r} |b\rangle \bigotimes_j |x_j\rangle |0\rangle$,
- appliquer $O_f$, pour obtenir $\sum_{b,x_1,\ldots x_r} |b\rangle \bigotimes_i |x_i\rangle |f(0, x_1, \ldots, x_r)\rangle$,
- (optionnel) mesurer le dernier registre, pour obtenir un $f(0, a_1, a_r)$ et l'état $|0\rangle \bigotimes_i |a_j\rangle + |1\rangle \bigotimes_j |a_j + y_j\rangle$
- appliquer $F_n^{\otimes r}$, ce qui donne

$$\sum_{l_1,\ldots,l_r} \exp\left(2i\pi \frac{\sum_j a_j l_j}{2^n}\right) |0\rangle \bigotimes_j |l_j\rangle + \exp\left(2i\pi \frac{\sum_j (a_j + y_j) l_j}{2^n}\right) |1\rangle \bigotimes_j |l_j\rangle$$

- mesurer les $r$ derniers registres, pour obtenir un vecteur $(l_1, \ldots, l_r)$ et l'élément $|\psi_{l_1,\ldots,l_r}\rangle \simeq |0\rangle + \exp\left(2i\pi \left(\sum_j y_j l_j\right)/2^n\right) |1\rangle$ (la phase globale étant ignorée).

On a alors une routine similaire, mais au lieu d'avoir une phase dépendant du produit $ly$, la phase dépend du produit scalaire $(l_1, \ldots, l_r) \cdot (y_1, \ldots, y_r)$.

### 5.3.2 Combinaisons d'éléments

**Définition 2.** On définit la hauteur de $(x_1, \ldots, x_r)$ vecteur de $\mathbb{Z}_{2^n}^r$ par le vecteur des hauteurs des différents éléments.

**Remarque 10.**

1. $\min h(x_1, \ldots, x_r) = n \Rightarrow \forall i, x_i = 0$
2. $\min h(x_1, \ldots, x_r) = n - 1 \Rightarrow \forall i, x_i \in \{0, 2^{n-1}\}$

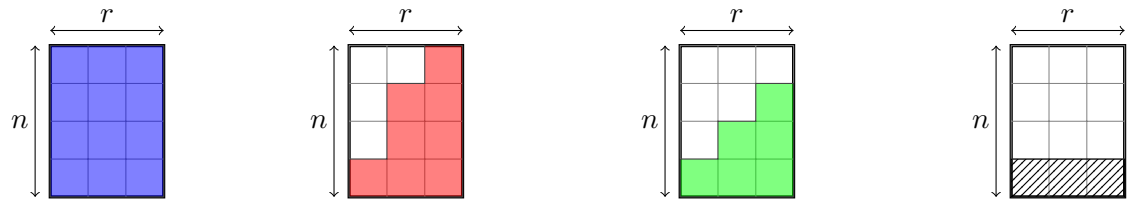
On peut utiliser le même système de combinaison d'éléments. Si deux vecteurs $x$ et $y$ ont les mêmes vecteurs de hauteur, les combiner fera augmenter toutes les valeurs de ce vecteur. De plus, si dans une certaine coordonnée, il y a plus de bits en commun, la hauteur à cette coordonnée augmentera d'autant.

Si $\min h(x_1 \ldots, x_r) = n - 1$, on a $|\psi_{x_1,\ldots,x_r}\rangle = |0\rangle + (-1)^{\sum_{j \in J(x_1,\ldots,x_r)} y_j} |1\rangle$, avec $J(x_1, \ldots, x_r)$ l'ensemble des indices non nuls, soit $\left\{i \in [1; r] \;\middle|\; x_i = 2^{n-1}\right\}$. Cet état permet donc d'obtenir une équation linéaire sur les $y_{0i}$, les bits de parité des $y_i$ : $\sum_{j \in J(x_1,\ldots,x_r)} y_j$, qui vaut 0 ou 1, selon ce que l'on mesure. On peut donc récupérer les valeurs de tous les bits de parité si l'on a suffisamment d'équations linéaires pour saturer le système. La probabilité que cela arrive est donnée par la proposition 1. De plus, si on combine deux éléments correspondant à ces équations entre eux, on obtient l'équation correspondant à la somme des deux équations. On ne peut donc pas gagner d'information comme cela, ce qui implique que l'on fera mieux que si l'on cherche à récupérer précisément une équation $x_i = \delta$ (avec la contrainte supplémentaire de la résolution d'un système linéaire).

L'heuristique du premier algorithme de Kuperberg (prendre, parmi les moins bons éléments, ceux qui collisionnent le mieux) fonctionne très bien ici. En effet, on peut voir les vecteurs de $\mathbb{Z}_{2^n}^k$ comme des vecteurs de $\mathbb{Z}_{2^{nk}}$ qui auraient été repliés. Si on l'applique à la lettre, le nombre de qubits annulés à chaque étape sera plus important, ce qui nous donne immédiatement une borne sur la complexité de cette méthode. Cependant, l'analyse précise des taux d'erreur et du coup est ardue, et la suite présente une méthode qui, si elle est asymptotiquement moins efficace, permet une évaluation précise de son comportement.

## 5.4 Découpage en blocs

La méthode utilisée par mon algorithme est de voir les éléments de $\mathbb{Z}_{2^n}^r$ comme des matrices booléennes de taille $n \times r$. On part d'éléments quelconques dans cet espace, et on souhaite arriver à des éléments dont toutes les cases sont à 0, sauf dans la ligne des bits de poids fort. Pour cela, on définit une suite de sous-groupes de $\mathbb{Z}_{2^n}^r$ imbriqués les uns dans les autres, et une étape de l'algorithme consistera à partir d'éléments d'un sous-groupe pour générer des éléments du sous-groupe suivant. Par exemple, la suite $E_0 = \mathbb{Z}_{2^n}^r$, $E_1 = \{x | \forall i, x_i \equiv 0 \mod 2\}$, $E_2 = \{x | \forall i, x_i \equiv 0 \mod 4\}$...est une suite possible. Les sous-groupes qui nous intéressent sont de la forme $\{(x_1, \ldots, x_r) | x_i \equiv 0 \mod 2^{\alpha_i}\}$, pour des $\alpha_i$ dans $[0; n]$ qui correspondent à la hauteur minimale des éléments dans l'ensemble (ce qui forme bien des sous-groupes, étant stables par l'addition et non vides). On peut alors représenter ces différents ensembles par des parties d'une matrice $n \times r$ imbriquées qui correspondent aux éléments pouvant valoir 1.




(a) Espace entier (b) Premier sous-groupe (c) Second sous-groupe (d) Sous-groupe objectif

FIGURE 5 – Suite de sous-groupes

On peut prendre une représentation plus compacte, mettant en évidence les différences entre ces groupes. En effet, c'est ce point qui importe : il représente l'ensemble des bits que l'on souhaite mettre à 0 durant une étape de l'algorithme. On obtient alors un découpage de la matrice $n \times r$ en blocs.

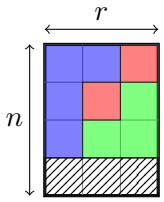



FIGURE 6 – Découpage en blocs de l'exemple précédent

## 5.5 Choix des blocs et complexité

L'annulation se fait en combinant des éléments qui sont égaux sur ledit bloc : si on obtient une soustraction, tout le bloc est annulé, sinon, seulement la première ligne du bloc l'est.

On peut choisir, pour cet algorithme, de prendre pour blocs les lignes de la matrice. Cela a l'avantage que chaque combinaison fonctionnera à tous les coups, et augmentera toutes les hauteurs d'au moins 1. Ces blocs sont de taille $2^r$. Cela signifie donc que si on part de $E$ éléments dans le premier bloc, on aura au plus $2^r$ termes qui ne collisionnent pas, et on obtiendra au moins $(E - 2^r)/2$ pour le bloc suivant, après les combinaisons. On peut donc calculer le nombre d'éléments à échantillonner pour être certain d'obtenir un certain nombre d'équations sur les bits de parité. Cela donne, pour obtenir $\delta$ équations,

$$E = 2^{n-1}\delta + 2^{n+r} + 2^{r+1}$$

échantillons.

On peut calibrer $\delta$ grâce à la proposition 1, mais il est peu utile de trop chercher à le réduire, et on peut le prendre de l'ordre de $2^r$ sans que cela soit le terme dominant du coût.

On peut aussi choisir des blocs de largeur variable. En effet, un gros bloc vers la fin de l'algorithme signifie qu'il faudra payer cher ses éléments qui ne collissionneront pas. Si $e$ est le nombre de blocs,

et $t_i$ leurs tailles respectives (le premier bloc est de taille $t_0$, et ainsi de suite), on obtient un coût en échantillons de

$$E = 2^e\delta + \sum_{i=0}^{e-1} 2^{i+t_i}.$$

Ce genre de formules est plus difficile à optimiser, mais on peut remarquer qu'il est toujours intéressant d'avoir des tailles de bloc décroissantes : si $t_{i+1} > t_i$, $2^{i+1+t_{i+1}} + 2^{i+t_i} \geq 2^{i+t_{i+1}} + 2^{i+1+t_i}$.

Pour les blocs de hauteur plus grande que 1, la situation est moins favorable : on a une chance sur 2 de gagner tout le bloc, et une chance sur 2 de ne gagner qu'une ligne. Si on considère que le bloc est traité ligne par ligne, cela donne $\frac{1}{4}$ des éléments qui sont directement annulés, plus $\frac{1}{4}$ des éléments dont seule la première ligne est annulée. On peut donc itérer cela pour réessayer d'annuler ces éléments, jusqu'à avoir parcouru toute la hauteur. La dernière opération fonctionnera à tous les coups. On obtient donc

$$\sum_{i=1}^{h} \left(\frac{1}{4}\right)^i + \left(\frac{1}{4}\right)^h = \frac{1}{3}\left(1 + \frac{2}{4^h}\right),$$

fois moins d'éléments parmi ceux que l'on peut collisionner. On va donc chercher à minimiser cette hauteur, et pour une taille de bloc $t$, faire le bloc tel que $h = \left\lceil \frac{t}{r} \right\rceil$.

Cela donne donc, pour un découpage en $e$ blocs de tailles $t_i$ et de hauteur $h_i$ et $\delta$ équations voulues un nombre requis d'échantillons qui est le terme $E_e$ de la suite

$$E_0 = \delta, E_{i+1} = 3\left(1 + \frac{2}{4^{h_{i+1}}}\right) E_i + 2^{t_{i+1}}.$$

Là aussi, il peut être complexe de trouver les meilleurs paramètres. On peut remarquer que les blocs doivent toujours être de taille décroissantes, et à partir de là faire une optimisation par ordinateur à partir des paramètres du problème.

### 5.6 Coût total

Une passe de l'algorithme récupère $r$ bits du décalage, et il faut l'appliquer $n$ fois (sur des espaces de plus en plus petits) pour tous les récupérer. Pour la dernière passe, le groupe est $\mathbb{Z}_{2^n}$, et on peut utiliser l'algorithme de Simon, qui est légèrement plus efficace. La partie précédente présentait le coût en requêtes, alors que le nombre d'opérations par échantillon n'est pas constant (il dépend du nombre de blocs et de leur hauteur). Si un échantillon est combiné, il a une chance sur 2 d'être annulé sur un bloc au premier coup, une chance sur 4 d'être annulé au second, et ainsi de suite. Cela donne un coût moyen par bloc plus petit que 2. On peut donc majorer le nombre d'opérations par 2 fois le nombre de blocs. Ce nombre va être de l'ordre de $\sqrt{nr}$, et on peut supposer que c'est un coût négligeable par rapport à celui d'une requête. Ce coût est plus important dans une version purement quantique si des blocs de hauteur plus grande que 1 sont présents. On peut aussi négliger la résolution du système linéaire, qui est en $r^3$.

Le coût en qubits est principalement celui des qubits que l'on cherche à faire collisionner durant la première passe, qui pour un découpage en blocs de taille $t_i$ est de $\sum_i 2^{t_i}$.

Si on considère un découpage en lignes avec $\delta$ équations à chaque étape, le coût total sera de

$$\sum_{i=1}^{n} 2^{i-1}\delta + 2^{i+r} + 2^{r+1} \simeq 2^n\delta + 2^{r+n+1}.$$

échantillons.

## 6 Applications

### 6.1 Variante d'Even-Mansour

On peut définir une variante d'Even-Mansour, avec $k_1$ dans $\mathbb{Z}_{2^n}^r$, $k_2$ dans $\mathbb{Z}_2^{rn}$ et $P$ une bijection publique de $\mathbb{Z}_{2^n}^r$ dans $\mathbb{Z}_{2^{n'}}^{r'}$ par $EM'_{k_1,k_2}(x) = P(x + k_1) \diamond k_2$ (avec deux addition terme-à-terme, +

dans le premier espace, $\diamond$ dans le second). $\diamond$ correspond typiquement à n'importe quelle opération d'un groupe de la forme $\mathbb{Z}_{2_1^n} \dots \mathbb{Z}_{2_m^n}$. On verra qu'elle n'a aucune importance (n'importe quelle opération commutative fonctionnant). Pour que $EM'$ soit un chiffrement, l'application $x \mapsto x \diamond k_2$ doit cependant être inversible.

La recherche exhaustive avec l'algorithme de Grover se fait en $\frac{\pi}{4} 2^{rn/2}$ opérations, dont le coût va être d'environ deux chiffrements.

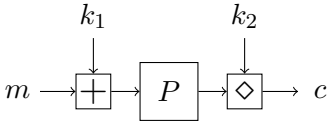



FIGURE 7 – Une variante d'Even-Mansour.

Cette variante est attaquable par notre algorithme, en construisant la fonction

$$\begin{array}{rccccc} f: & \{0,1\} & \times & \{0,1\}^{nr} & \to & \{0,1\}^{n'r'} \\ & b & , & x=(x_1,\dots,x_r) & \mapsto & \begin{cases} EM'_{k_1,k_2}(x) \diamond P(-x) & \text{si } b=0 \\ EM'_{k_1,k_2}(-x) \diamond P(x) & \text{si } b=1 \end{cases} \end{array}$$

Cette fonction vérifie $f(1, x+k_1) = P(-x-k_1+k_1) \diamond k_2 \diamond P(x+k_1) = P(x+k_1) \diamond k_2 \diamond P(-x) = f(0,x)$. Elle a donc $(1, k_1)$ comme période, et on peut appliquer l'algorithme pour le décalage caché pour récupérer $k_1$, puis récupérer $k_2$ en une requête classique. Le coût est de l'ordre de $2^{n+r}$ pour une version basique de l'algorithme, ce qui fait bien mieux que la recherche exhaustive si $r$ et $n$ sont du même ordre de grandeur.

### 6.2 Slide attack

On peut aussi attaquer des chiffrements dont la fonction de tour est $F_k(x) = P(x+k)$, toujours avec $k$ dans $\mathbb{Z}_{2^n}^r$, $+$ l'addition terme-à-terme et $P$ une bijection publique.

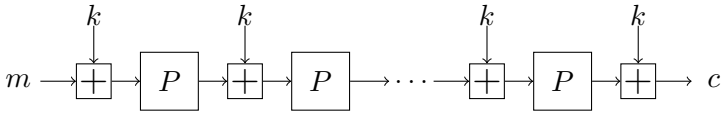



FIGURE 8 – Chiffrement itératif vulnérable à une slide attack quantique.

Ce chiffrement vérifie, pour tout $x$, $E_k(P(x+k)) = P(E_k(x)) + k$. On peut alors créer la fonction

$$\begin{array}{rccccc} f: & \{0,1\} & \times & \{0,1\}^{nr} & \to & \{0,1\}^{nr} \\ & b & , & x=(x_1,\dots,x_r) & \mapsto & \begin{cases} P(E_k(x)) - x & \text{si } b=0 \\ E_k(P(x)) - x & \text{si } b=1 \end{cases} \end{array}$$

qui vérifie $f(0,x) = f(1, x+k)$. Là aussi, le coût est directement celui d'une résolution d'un décalage caché, soit en $2^{n+r}$ pour une version basique.

## 7 Conclusion

Ces travaux sont un exemple de ce que l'on peut faire en cryptanalyse quantique symétrique, et constituent une base qui peuvent avoir divers prolongements. Le plus évident est la recherche d'une slide attack quantique sur Treyfer, mais d'autres pistes sont possibles, comme une analyse fine de l'impact de la non-injectivité sur les algorithmes de décalage caché, l'étude d'autres algorithmes résolvant ce problème, ou plus généralement l'adaptation des techniques actuelles de cryptanalyse symétrique à un cadre quantique. Cela sera le thème de ma thèse commençant en septembre.

## Références


[1] *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 1994.

[2] Andris Ambainis, Arturs Backurs, Nikolajs Nahimovs, and Alexander Rivosh. Grover's Algorithm with Errors. In Antonín Kucera, Thomas A. Henzinger, Jaroslav Nesetril, Tomás Vojnar, and David Antos, editors, *MEMICS*, volume 7721 of *Lecture Notes in Computer Science*, pages 180–189. Springer, 2012.

[3] Mayuresh Vivekanand Anand, Ehsan Ebrahimi Targhi, Gelo Noel Tabia, and Dominique Unruh. Post-Quantum Security of the CBC, CFB, OFB, CTR, and XTS Modes of Operation. In Tsuyoshi Takagi, editor, *PQCrypto*, volume 9606 of *Lecture Notes in Computer Science*, pages 44–63. Springer, 2016.

[4] Dave Bacon, Andrew M. Childs, and Wim van Dam. Optimal measurements for the dihedral hidden subgroup problem. *Chicago Journal of Theoretical Computer Science (2006), no. 2*, April 2005.

[5] Daniel J. Bernstein. Post-Quantum Cryptography. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 949–950. Springer, 2011.

[6] Alex Biryukov and David Wagner. Slide Attacks. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[7] Alex Biryukov and David Wagner. Advanced Slide Attacks. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000.

[8] Dong Pyo Chi and Jinsoo Kim. Quantum Database Searching by a Single Query. *Quantum Computing and Quantum Communications, First NASA International Conference, selected papers, QCQC'98, C. P. Williams Ed. (Palm Springs, California, USA, February 17-20, 1998), Lecture Notes in Computer Science, Vol. 1509, pp. 148-151, Springer-Verlag, 1999 ; "Quantum Database Search with Certainty by a Single Query", Chaos, Solitons, and Fractals 10 (1999), 1689-1693.*, August 1997.

[9] Mark Ettinger and Peter Høyer. *On Quantum Algorithms for Noncommutative Hidden Subgroups*, pages 478–487. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[10] Shimon Even and Yishay Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *J. Cryptology*, 10(3) :151–162, 1997.

[11] Katalin Friedl, Gábor Ivanyos, Frédéric Magniez, Miklos Santha, and Pranab Sen. Hidden translation and orbit coset in quantum computing. In *STOC*, pages 1–9. ACM, 2003.

[12] Mirmojtaba Gharibi. Reduction from non-injective hidden shift problem to injective hidden shift problem. *Quantum Information & Computation*, 13(3-4) :221–230, 2013.

[13] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *STOC*, pages 212–219. ACM, 1996.

[14] Yoshifumi Inui and François Le Gall. Efficient quantum algorithms for the hidden subgroup problem over semi-direct product groups. *Quantum Information & Computation*, 7(5) :559–570, 2007.

[15] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking Symmetric Cryptosystems using Quantum Period Finding. In *CRYPTO 2016*, Lecture notes in computer science. Springer, 2016.

[16] Greg Kuperberg. A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. *SIAM J. Comput.*, 35(1) :170–188, 2005.

[17] Greg Kuperberg. Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. In Simone Severini and Fernando G. S. L. Brandão, editors, *TQC*, volume 22 of *LIPIcs*, pages 20–34. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[18] Hidenori Kuwakado and Masakatu Morii. Security on the quantum-type Even-Mansour cipher. In *ISITA*, pages 312–316. IEEE, 2012.

[19] Oded Regev. Quantum Computation and Lattice Problems. In *FOCS*, pages 520–529. IEEE Computer Society, 2002.

[20] Oded Regev. A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space, June 2004.

[21] Oded Regev and Liron Schiff. Impossibility of a Quantum Speed-Up with a Faulty Oracle. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 773–781. Springer, 2008.

[22] Peter W. Shor. Algorithms for Quantum Computation : Discrete Logarithms and Factoring. In *FOCS* [1], pages 124–134.

[23] Daniel R. Simon. On the Power of Quantum Cryptography. In *FOCS* [1], pages 116–123.

[24] Christof Zalka. Grover's quantum searching algorithm is optimal. *Phys.Rev. A60 (1999) 2746-2751*, December 1999.