HAL
open science

Chiffrement homomorphe appliqué au retrait d'information privé

Joris Barrier

To cite this version:

Joris Barrier. Chiffrement homomorphe appliqué au retrait d'information privé. Informatique [cs]. Institut National des Sciences Appliquées de Toulouse (INSA Toulouse), 2016. Français. NNT : . tel-01682332v1

HAL Id: tel-01682332

https://laas.hal.science/tel-01682332v1

Submitted on 12 Jan 2018 (v1), last revised 12 Feb 2018 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Fédérale

Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *INSA de Toulouse*

Présentée et soutenue le *13/12/2016* **par :**

Joris Barrier

Chifffrement Homomorphe appliqué au Retrait d'Information Privé

JURY

M. Phillipe Gaborit	Professeur	Rapporteur
Mme. Caroline Fontaine	Chargée de Recherche	Rapporteur
M. Gentian Jakllari	Maître de Conférence	Examinateur
Mme. Maryline Laurent	Professeur	Examinateur
M. Marc-Olivier Killijian	Directeur de Recherche	Directeur
M. Carlos Aguilar Melchor	Enseignant Chercheur	Directeur

École doctorale et spécialité :
Mathématiques, Informatique, Télécommunications de Toulouse (ED MITT)

Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique Français (LAAS-CNRS)

Directeur(s) de Thèse :
M. Marc-Olivier Killijian et *M. Carlos Aguilar Melchor*

Rapporteurs :
M. Phillipe Gaborit et *Mme. Caroline Fontaine*

Résumés de la thèse

Français

Le retrait d'information privé que nous nommons PIR, désigne un groupe de protocoles qui s'inscrit dans un ensemble plus vaste des technologies d'amélioration de la vie privée. Sa fonctionnalité principale est de dissimuler l'index d'un élément d'une liste accédée par un client au regard de son hôte. Sans négliger l'appart de leurs auteurs à la communauté scientifique, l'utilisabilité de ce groupe de protocoles semble limitée, car pour un client, télécharger l'intégralité de la liste est plus efficient. À ce jour, les PIR, se fondent sur des serveurs répliqués mutuellement méfiants, des périphériques de confiance ou bien des systèmes cryptographiques. Nous considérerons ici les retraits d'informations privés computationnels et plus particulièrement ceux reposant sur les réseaux euclidiens qui n'offrent des propriétés particulières, comme l'homomorphisme. Afin d'en démontrer l'utilisabilité, nous proposons un retrait d'information privé reposant sur un système cryptographique homomorphe performant et aisé d'utilisation.

English

Private information retrieval, named PIR, is a set of protocols that is a part of privacy enhancement technologies.Its major feature is to hide the index of a record that a user retrieved from the host.Without neglecting the scientific contributions of its authors, the usability of this protocol seems hard since that, for a user, it seems more and more efficient to receive all the records.Thus far, PIR can be achieved using mutually distrustful databases replicated databases, trusted hardware, or cryptographic systems.We focus on computational private information retrieval, and specifically on thus based on cryptographic systems.This decision is contingent to the spread of cryptographic systems based on lattices who provide specific properties.To demonstrate it usability, we offer an efficient and easy-to-use private Information retrieval based on homomorphic encryption.

Table des matières

Introduction Générale **1**

1 Réseaux d'idéaux et polynômes **3**

2 Chiffrement homomorphe **23**

3 Private Information Retrieval **35**

4 Bibliothèque NFL **49**

5 Private Information Retrieval for Everyone **61**

Conclusion Générale **87**

Introduction générale

L'omniprésence d'Internet dans les pays développés est factuelle. D'après l'Union Internationale des Télécommunications, en 2015 plus de quatre-vingt pour cent [28] des individus peuplant ces pays ont accès à internet de leur domicile. En 2016, les réseaux sociaux Facebook et Baidu, respectivement Américain et Chinois, totalisent près de 2 milliards d'utilisateurs actifs dont la principale source de revenus est issue de leurs régies publicitaires. Nous pouvons aussi citer les services de vidéos à la demande, tels que Netflix et Rutube, qui se popularisent ainsi que les objets connectés tels que les montres, thermostats, pèse-personne, etc. Et pour terminer, nous pensons à la numérisation des services publics comme marqueur de l'omniprésence d'internet dans ces pays.

De plus, l'acte même du choix, d'un service parmi plusieurs, d'une vidéo parmi plusieurs par exemple, génère des données discriminantes. Ainsi se posent des questions de contrôle de ces informations et par conséquent de vie privée dans le débat public. Et plus particulièrement depuis le scandale produit par les révélations d'écoute massive de la «*National Security Agency*» Étasunienne en 2013.

Puisque ce sont les choix des internautes qui révèlent leurs préférences, nous étudierons dans ce mémoire un protocole client/serveur qui permet d'échanger des informations en masquant les centres d'intérêt des utilisateurs non seulement du point de vue d'un tiers capable de lire les échanges, mais aussi du serveur. Ce protocole est nommé en français Retrait d'Information Privé que nous abrègeront PIR pour correspondre à la littérature anglo-saxonne. Autrement dit, le PIR donne la possibilité de masquer un choix parmi plusieurs. Néanmoins, nous verrons que, malgré ses possibilités, le PIR est fortement limité et ce sont ses limites que nous avons cherché à franchir durant cette thèse.

Dans la première partie de ce mémoire, nous introduisons les concepts mathématiques et techniques nécessaires à la compréhension de nos contributions. Nous discutons des polynômes utilisés en cryptographie fondée sur les réseaux euclidiens ainsi que des outils comme la transformée de Fourrier discrète et le théorème des restes chinois.

Dans la seconde partie, nous proposons un état de l'art des systèmes de chiffrement homomorphes. Nous y expliquons leurs particularités et leurs avantages dans le cas du PIR.

Dans la troisième partie, nous nous intéressons à l'état de l'art des protocoles PIR, et plus particulièrement des protocoles PIR construits à partir de fonctions issues de systèmes de chiffrement homomorphes.

Dans la quatrième partie, nous décrivons notre première contribution : une bibliothèque, nommée NFLlib, de manipulation de polynômes sur les réseaux euclidiens d'idéaux. Cette bibliothèque a plusieurs objectifs : le premier est la performance, le second est d'être conçu comme une brique de construction de systèmes cryptographiques et enfin d'être employée dans des protocoles plus complexes, typiquement les protocoles PIR. Afin de montrer les performances de NFLlib nous la comparons aux bibliothèques de manipulation de polynômes les plus répandues pour le même langage de programmation.

Dans la cinquième partie, nous décrivons notre seconde contribution : une application de retrait d'information privé nommé *XPIR : Private Information*

Retrieval for Everyone. Cette application dont le cryptosystème est écrit avec NFLlib nous permet de montrer la praticabilité du PIR dans des cas réalistes comme des services de vidéos à la demande. De plus, nous avons porté une attention particulière à sa réutilisabilité et cette dernière est fournie avec une interface de programmation.

Enfin, nous terminerons par la conclusion des travaux présentés dans ce mémoire.

Chapitre 1

Réseaux d'idéaux et polynômes

Table des matières

1.1 Introduction 4
1.2 Définitions 4
1.2.1 Polynôme 4
1.2.2 Polynômes dans les réseaux 4
1.3 Représentation dans un ordinateur 4
1.3.1 Représentation d'un coefficient 4
1.3.2 Représentation d'un polynôme 5
1.4 Opérations arithmétiques 5
1.4.1 Somme de polynômes 5
1.4.2 Produit de polynômes 5
1.5 Calculs de modulo 5
1.5.1 Somme modulaire 6
1.5.2 Produit modulaire 6
1.6 Représentation CRT : théorème des restes Chinois 11
1.6.1 Relèvement 12
1.6.2 Représentation des polynômes modulo un grand nombre 12
1.6.3 Somme CRT 12
1.6.4 Produit CRT 13
1.7 Caractérisation d'un polynôme 14
1.7.1 Coefficients 14
1.7.2 Valeurs 14
1.8 La transformée de Fourier 16
1.8.1 Introduction 16
1.8.2 Approche intuitive 17
1.8.3 Représentation sous forme de sommes 17
1.8.4 Racines n-èmes de l'unité 18
1.8.5 Notation matricielle 19
1.9 Transformée de Fourier discrète rapide 20
1.10 Transformée de Fourier rapide sur un corps fini 21
1.10.1 Produit de deux polynômes 21
1.11 Conclusion 22

Résumé : *Nous décrivons les différents outils mathématiques, tels que les anneaux de polynômes, en passant par la transformation de Fourier ou le théorème des restes chinois, nécessaires à la bonne compréhension de ce mémoire.*

1.1 Introduction

Depuis ces dernières années, la cryptographie basée sur les réseaux euclidiens est devenue incontournable dans le milieu universitaire. Ces systèmes cryptographiques ont la particularité de vivre dans des structures mathématiques qui utilisent des polynômes. C'est naturellement à ces polynômes que nous allons nous intéresser dans ce chapitre.

1.2 Définitions

Pour débuter, nous allons tout d'abord définir les polynômes puis nous verrons leurs particularités dans les réseaux euclidiens.

1.2.1 Polynôme

Une caractérisation possible d'un polynôme est une séquence de $n+1$ [1] valeurs $(a_0, a_1, \dots, a_{n-1}, a_n)$ ou bien une fonction P définie par une expression du type :

$$\mathrm{P}(x) = \sum_{i=0}^{n} a_i x^i$$

Ou les a_i sont appelés coefficients du polynôme. Le degré d'un polynôme est donné par la valeur i la plus grande pour laquelle $a_i \neq 0$. Par exemple le polynôme $4x^2 + 13x + 5$ a pour degré 2 et le polynôme $7x^5 + 2x^2 + 32$ a pour degré 5.

1.2.2 Polynômes dans les réseaux

Dans cette configuration, les coefficients seront représentés modulo un entier et les polynômes modulo un autre polynôme. La forme du polynôme de module sera toujours $\phi_n(\mathrm{X}) = \mathrm{X}^n + 1$ avec n une puissance de 2. Ainsi, le degré des polynômes manipulés est au maximum $n-1$. Le polynôme $\phi_n(\mathrm{X})$, appelé *polynôme cyclotomique*, est le polynôme unitaire - dont le coefficient du monôme dominant est 1 - dont les racines complexes sont les racines primitives n-ièmes de l'unité. Quand on travaille modulo un polynôme ayant la forme $\mathrm{X}^n + 1$, l'opération modulaire se simplifie comme suit : le coefficient X^{n+i} devient $-\mathrm{X}^i$.

1.3 Représentation dans un ordinateur

1.3.1 Représentation d'un coefficient

L'architecture des processeurs modernes donne l'opportunité d'effectuer des opérations très efficaces, typiquement sur 64 bits, qui utilisent les registres à mémoire présents sur les processeurs. Afin de maximiser les performances offertes, nous pouvons configurer les coefficients des polynômes pour remplir au maximum les registres disponibles.

À noter que, même sur une architecture 64 bits, les processeurs d'aujourd'hui proposent des opérations dites *Single Instruction Multiple Data* sur 32 voir 16 bits. Les jeux d'instructions les plus connus sont *Streaming SIMD Extensions (SSE)* et *Advanced Vector Extensions (AVX)*. C'est-à-dire qu'au lieu de stocker un seul entier de 64 bits dans un registre, un programmeur peut en stocker respectivement 2 ou 4 dans des registres dédiés de taille plus importante et d'effectuer des opérations parallèlement. Les compilateurs récents tels que `gcc` sont capables d'utiliser ce type d'opérations ; ce processus est appelé *vectorisation*. Cependant, toutes les instructions ne sont pas gérées et il peut être nécessaire de devoir faire appel à ces instructions directement dans le programme.

1. On commence le décompte à 0 et on termine à n

1.3.2 Représentation d'un polynôme

Un polynôme peut être représenté comme un tableau de n cases d'entiers de 64 bits. Cette configuration, illustrée ci-dessous, lie le i-ème coefficient à la i-ème case, simplifiant la conception et la lecture des algorithmes.

case 0	case 1	case 2	...	case $n-1$
a_0	a_1	a_2	...	a_{n-1}

1.4 Opérations arithmétiques

La cryptographie fondée sur les réseaux euclidiens nécessite l'utilisation d'opérations arithmétiques sur les polynômes telles que la somme et le produit de deux polynômes que nous allons présenter dans cette section.

1.4.1 Somme de polynômes

L'opération arithmétique de somme entre deux polynômes est simple. Elle consiste à calculer un nouveau polynôme, résultat de la somme des 2 premiers, coefficients à coefficients modulo un entier.

Soient A et B deux polynômes de degré n dont les i-ème coefficients seront écrits respectivement a_i et b_i. L'algorithme calculant les coefficients c_i du polynôme C, somme de A et B, s'écrit :

Algorithm 1 Somme de deux polynômes

```
1 : procedure SUM(A, B)
2 :     for i ← 0, n do                    ▷ Pour tous les coefficients
3 :         c_i ← a_i + b_i mod p
4 :     end for
5 :     return C
6 : end procedure
```

Concernant les registres, le résultat direct de la somme de deux coefficients peut éventuellement dépasser leur taille, faussant les données. Afin de pallier à tous débordements possibles, on utilise un module de 1 ou 2 bits plus petits que les registres, *i.e* 63-62 bits.

1.4.2 Produit de polynômes

L'opération arithmétique naïve de produit entre deux polynômes est plus complexe. La multiplication étant distributive, chaque coefficient du premier polynôme est multiplié par chaque coefficient du second. Cette méthode implique d'imbriquer deux boucles **for**.

Comme précédemment, soient A et B deux polynômes de degré n dont les i-ème coefficients seront écrits respectivement a_i et b_i. L'algorithme calculant les coefficients c_i de C, produit de A et B, s'écrit :

Dans le cas du produit de deux coefficients, le résultat devra être stocké dans un registre deux fois plus grand afin d'éviter tout débordement. Par exemple, le résultat de la multiplication de deux entiers de 64 bits doit être stocké sur 128 bits.

1.5 Calculs de modulo

L'opération de réduction modulaire d'un premier entier par un second, le module, est obtenue en calculant le reste de la division euclidienne de l'entier à réduire par le module. Dans l'opération de division, contrairement à la multiplication ou l'addition, il faut «deviner» combien de fois le diviseur (le module) se trouve dans le dividende (l'entier à réduire) afin de déterminer le reste.

Algorithm 2 Produit naïf de deux polynômes

```
procedure PRODUCT(A, B)
    for i ← 0, n do
        c_i ← 0
    end for
    for i ← 0, n do
        for j ← 0, n do
            α ← (a_i × b_j) mod p                    ▷ 64 × 64 → 128 bits
            if i + j >= n then
                i' ← i + j − n
                c_i' ← c_i' − α mod p                ▷ mod X^n + 1
            else
                c_i+j ← c_i+j + α mod p
            end if
        end for
    end for
    return C
end procedure
```

1.5.1 Somme modulaire

Les ressources processeurs requises pour une opération de modulo sont prohibitives (opération «devinette») dans le cadre d'une recherche de performances. Pour l'algorithme 1 ligne 3, nous pouvons remplacer l'opération par la soustraction conditionnelle suivante :

```
if c_i > p then
    c_i = c_i − p
end if
```

1.5.2 Produit modulaire

Concernant l'algorithme de produit, la réduction modulaire ne peut pas se simplifier aussi trivialement que pour la somme, car le nombre d'occurrences du module dans l'entier à réduire ne peut pas être connu à l'avance. Autrement dit, on ne sait pas de « combien » a été dépassé le module après une multiplication de deux entiers.

Nous pouvons envisager d'optimiser le module afin de diminuer le cout calculatoire de l'opération modulaire et plus particulièrement de la division. Cette réduction revient à faire un décalage, une multiplication par une constante et une somme.

Paul Barrett propose dans [2] une méthode d'accélération du calcul de réduction dont le principe est le suivant :

Réduction de P. Barrett : On veut calculer $x = a \bmod n$ avec des mots machine de k bits, $a < n^2$ étant représenté sur 2 mots, et n sur 1 mot. P. Barrett exprime la réduction comme suit : $x = a \bmod n \Leftrightarrow x = a - \left\lfloor \frac{a}{n} \right\rfloor \cdot n$. Si le module n est fixe, il est possible de précalculer $1/n$ dont la valeur est inférieure à 1 et doit être stockée dans un flottant. Rappelons que l'on veut réduire un entier a de deux mots de k bits. Afin de travailler uniquement avec des entiers, on va décaler la mantisse de $1/n$ pour la représenter dans un entier de 2 mots, ce qui revient à faire un décalage de $2k$ bits ou une multiplication de 2^{2k} ou encore 4^k. On obtient une valeur intermédiaire $m = \left\lfloor \frac{4^k}{n} \right\rfloor$ qui permettra d'approcher le quotient $\lfloor a/n \rfloor$ en multipliant a et m et en divisant par 4^k afin « d'annuler » le premier décalage utilisé pour représenter $1/n$ dans un entier de 2 mots. Ceci donne : $q = \left\lfloor \frac{a \cdot m}{4^k} \right\rfloor$. Maintenant, il reste à calculer $x = a - n \cdot q$. Mais une soustraction conditionnelle doit être

réaliser afin d'obtenir le résultat :

$$x = \begin{cases} x - n & si \quad x \geq n \\ x & sinon \end{cases}$$

En effet, lorsque l'on calcule $m = \left\lfloor \frac{4^k}{n} \right\rfloor$, la partie au-delà des 4^k bits, que l'on appellera par la suite «queue», est tronquée. Ainsi on aura $\frac{4^k}{n} = \left\lfloor \frac{4^k}{n} \right\rfloor + \epsilon$, avec $\epsilon < 1$ représentant la queue du flottant. Pour calculer la valeur exacte de $\lfloor a/n \rfloor$, il aurait fallu calculer : $a \cdot (m + \epsilon)/4^k > a \cdot m/4^{\mathrm{K}}$.

Toutefois, dans la méthode présentée la valeur ϵ n'intervient pas dans le calcul, ce qui biaise potentiellement le résultat d'une fois la valeur de module n. Comme illustré en vert sur la figure 1.1, l'erreur ϵ est très inférieure à 1. Cependant, lorsque l'on somme cette erreur afin de calculer la valeur exacte de $\lfloor a/n \rfloor$ on voit qu'elle se propage. Et plus particulièrement jusqu'aux bits en jaune sur le schéma ce qui, finalement, ajoute au maximum 1 à $\lfloor a/n \rfloor$. En effet, on a $\epsilon < 4^{-k}$ et $a < n^2 < 4^k$ et donc $a \cdot \epsilon < 1$ bornant $a \cdot \epsilon$. À noter que plus la bande jaune est large (plus il y a de 1 à la suite), plus la probabilité de propagation de $a \cdot \epsilon$ au-dessus de la virgule est élevée. Ainsi, comme on ne connaît pas à l'avance la valeur de a, il faut toujours tester si ce phénomène s'est produit, c'est-à-dire que l'on a calculé $\lfloor a/n \rfloor - 1$ au lieu de $\lfloor a/n \rfloor$, et ce indépendamment de la précision de ϵ.

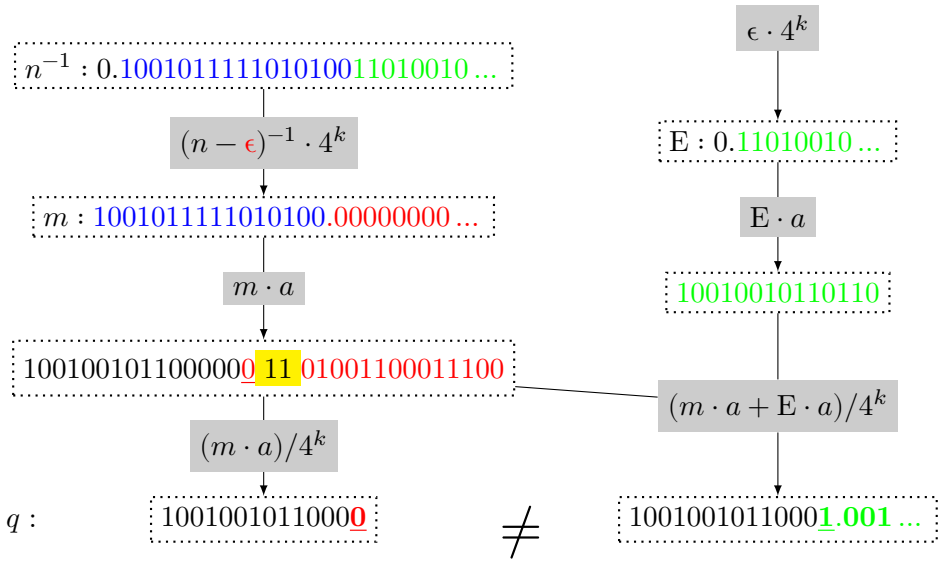


Figure 1.1 – Exemple représenté en binaire de l'effet produit par l'arrondi de $1/n$.

Algorithm 3 Réduction de Barrett

Require : $a < n^2$

1 : **procedure** BarrettReduction(a, m)

2 : $\quad m \leftarrow \left\lfloor \frac{4^k}{n} \right\rfloor$ $\quad \triangleright$ 4^k, 3 mots

3 : $\quad q \leftarrow \left\lfloor \frac{a \cdot m}{4^k} \right\rfloor$ $\quad \triangleright$ $128 \times 128 \rightarrow 256$ bits

4 : $\quad x \leftarrow a - n \cdot q$

5 : $\quad$ **if** $x \geq n$ **then**

6 : $\quad\quad x \leftarrow x - n$

7 : $\quad$ **end if**

8 : $\quad$ **return** x

9 : **end procedure**

Attention, cet algorithme peut poser des problèmes techniques lors d'une réduction pour deux raisons :

1. Le calcul de la ligne 2 de l'algorithme 3, revient à faire un décalage sur la gauche des bits de n^{-1}. Pour calculer 2^{2k} et le diviser par n il faut passer par un nombre sur trois mots car deux mots peuvent contenir les entiers entre 0 et $2^{2k}-1$, ce qui n'est pas géré nativement si ces derniers sont de 64 bits.
2. La multiplication de la ligne 3 de l'algorithme 3, se trouve entre deux entiers de deux mots ou entre un entier de deux mots et un entier d'un seul. De nouveau, utilisé avec des mots de 64 bits, il faudrait stocker le résultat dans des registres de plus de 128 bits qui n'existent pas encore nativement dans les processeurs d'ordinateurs de bureau. Le traitement de ces nombres engendre donc un surcoût significatif, ce qui ternit l'avantage obtenu en évitant la division.

Multiplication modulaire de V. Shoup : On veut calculer $r = a \cdot b \mod n$ avec des mots machine de k bits, n $< 2^k$, a et b $\in [0, n)$. Cet algorithme tente de répondre au même problème que celui de Barrett en supprimant l'une des contraintes : des opérations sur des données correspondant à trois ou quatre mots machine. Ainsi, V. Shoup propose dans [51] un algorithme qui réduit modulo un entier le résultat d'une multiplication en passant par des données qui s'exprime en au plus deux mots machine. Cependant, la technique introduit un précalcul sur l'une des opérandes (les coefficient dits «de Newton»), que l'on verra par la suite.

En premier lieu, on va écrire $a \cdot b \mod n = a \cdot b - q \cdot n$, avec q le quotient de la division euclidienne de $a \cdot b$ par n. Il faut alors trouver une méthode pour calculer q sans utiliser de flottants et sans avoir de résultats intermédiaires supérieurs à 2^{2k} (plus grand que 2 mots machine). Naïvement, on peut calculer $q = \dfrac{a \cdot b}{n}$ mais cette méthode n'a pas d'intérêt dans le cas présent, car elle oblige, soit de faire une division par n à chaque appel (rappel : les divisions sont lentes), soit de calculer $1/n$ puis de faire $a \cdot b \cdot 1/n$ pour retrouver les mêmes difficultés qu'avec l'algorithme de Barrett.

Afin de résoudre une partie du problème, l'idée est la suivante : on peut «sortir» une partie du calcul comme suit : $q = \dfrac{a \cdot b}{n} = a \cdot \dfrac{b}{n}$. Comme on ne peut toujours pas calculer directement q sans utiliser de flottant sous cette forme, on va d'abord précalculer indépendamment une approximation de $\frac{b}{n}$ avec pour objectif de se séparer des flottants. Cependant $\frac{b}{n} < 1$ car $b < n$, le résultat de cette division ne tient pas dans un registre d'entier. On va alors décaler $\frac{b}{n}$ de k bits afin de faire tenir le résultat sur un entier, revenant à multiplier par 2^k et tronquer le reste, ce qui donne $b' = \left\lfloor \dfrac{2^k \cdot b}{n} \right\rfloor$. Puis, à partir de b' on peut obtenir le quotient q. En effet, si on développe $a \cdot b'$ on a : $a \cdot b' \approx \dfrac{a \cdot b \cdot 2^k}{n}$, on voit qu'il reste 2^k, donc pour trouver q il faut le supprimer en divisant par 2^k. Ceci énoncé, on peut exprimer q tel que

$$q = \frac{\left(\dfrac{a \cdot b \cdot 2^k}{n}\right)}{2^k}$$

Toutefois, si on calcule $\left\lfloor \dfrac{a \cdot b'}{2^k} \right\rfloor$ nous avons une approximation de q car $b' = \left\lfloor \dfrac{2^k \cdot b}{n} \right\rfloor = \dfrac{2^k \cdot b}{n} - \epsilon$ pour un ϵ inférieur à un.

Ainsi, si on calcule $r = a \cdot b - \left\lfloor \dfrac{a \cdot b'}{2^k} \right\rfloor \cdot n$, il faudra, à l'instar de la réduction de Barrett, corriger avec une soustraction conditionnelle r :

$$r = \begin{cases} r - n & si \quad r \geq n \\ r & sinon \end{cases}$$

L'effet d'amplification de l'erreur d'arrondi est d'ailleurs proche de celui de l'algorithme de

Barrett, et si on exprime les deux erreurs comme suit :

$$(1) : 0 \leq \frac{b \cdot 2^k}{n} - b' < 1 \quad \text{et} \quad (2) : 0 \leq \frac{a \cdot b'}{2^k} - q < 1$$

Avec (1) représentant l'erreur d'arrondi produite lors du calcul de b' et (2) celle produite lors de l'arrondit du calcul de q. Puis, que l'on cherche exprimer r à partir de (1) et (2), on va multiplier (1) par $\frac{a \cdot n}{2^k}$ et (2) par n, ce qui donne :

$$(3) : 0 \leq a \cdot b - \frac{a \cdot b' \cdot n}{2^k} < \frac{a \cdot n}{2^k} \quad et \quad (4) : \frac{a \cdot b' \cdot n}{2^k} - q \cdot n < n$$

Puis on additionne (3) et (4) :

$$\begin{aligned} 0 \leq \quad & a \cdot b - \frac{a \cdot b' \cdot n}{2^k} + \frac{a \cdot b' \cdot n}{2^k} - q \cdot n \quad < n + \frac{a \cdot n}{2^k} \\ \Leftrightarrow \quad 0 \leq \quad & a \cdot b - q \cdot n \quad < n + \frac{a \cdot n}{2^k} \end{aligned}$$

Toutefois, $\frac{a \cdot n}{2^k} < n$ car $a < n < 2^k$, donc :

$$0 \leq a \cdot b - q \cdot n < 2n$$

Par conséquent, si on approche q par $\left\lfloor \frac{a \cdot b'}{2^k} \right\rfloor$, il faut vérifier s'il faut ajouter un à ce calcul intermédiaire pour bien obtenir q.

Cet algorithme a un avantage face à la réduction de Barrett : il n'y a pas de cas où 3 mots machine sont nécessaires. En revanche, il faut un précalcul pour chaque opérande b. Mais le coût de ce précalcul peut être amorti si réutilisé, ce qui est particulièrement intéressant lors d'un PIR lorsque le nombre b représente un élément d'une base de données.

Algorithm 4 Multiplication modulaire de V. Shoup

Require : $a, b \in [0, n)$, avec $n < \beta/2$, et $b' = \lfloor 2^k \cdot b/n \rfloor$ précalculé.
Ensure : $r = a \cdot b \bmod n$

```
procedure MulMod(a, b, b')
    q ← ⌊b' · a/2^k⌋                    ▷ Partie haute du produit b'a
    r ← (a · b) − (q · n)
    if r ≥ n then
        r ← r − n
    end if
    return r
end procedure
```

Réduction rapide : Il existe également d'autres algorithmes de réduction plus sophistiqués tels que celui que nous avons présenté dans [1], reproduit ici par l'algorithme 5. On utilisera une notation supplémentaire pour un nombre sur deux mots $x = \langle x_1, x_0 \rangle$ avec x_0 le premier mot de x et x_1 le second. Dans cette proposition, on cherche à calculer efficacement $a \bmod n$ avec $a = \langle a_1, a_0 \rangle$ et n vérifiant l'inéquation 1.1 dont la nécessité sera expliquée par la suite.

$$(1 + 1/2^{3\ell_0}) \cdot \beta/(2^{2\ell_0} + 1) < n < \beta/2^{2\ell_0} \tag{1.1}$$

Avec ℓ_0 un certain nombre de bits de marge tel que $0 < \ell_0 < \ell$.

Comme pour l'algorithme de P.Barrett, on cherche à déterminer le quotient de la division euclidienne de a par n en calculant $1/n$. Comme on ne souhaite pas utiliser de flottant on va décaler l'inverse de n afin de le faire tenir sur un entier de 2 mots ce qui donne : $m = \left\lfloor \frac{\beta^2}{p} \right\rfloor$. C'est précisément ici que joue la contrainte de l'inéquation 1.1 appliquée à n. En effet, cette dernière assure que la forme de m sera la suivante $m = \langle 2^{\ell_0}, m_0 \rangle$.

Ceci nous amène au calcul de q. Comme on connait à l'avance la forme de m, on peut se permettre de conserver seulement m_0. Cependant, on ne peut pas calculer directement une approximation de q par $a \cdot m$ car a, et m sont sur deux mots, et leur produit sur quatre. Étudions ce produit en le développant pour comprendre comment éviter de passer par quatre mots machine :

$$a \cdot m = a_0 \cdot m_0 + \beta(a_1 \cdot m_0) + \beta(a_0 \cdot 2^{\ell_0}) + \beta^2(a_1 \cdot 2^{\ell_0}).$$

Pour calculer q nous devons diviser $a \cdot m$ par β^2 pour «annuler» le premier décalage réalisé en calculant m :

$$\frac{a \cdot m}{\beta^2} = \underbrace{\frac{a_0 \cdot m_0}{\beta^2}}_{<1} + \frac{(a_1 \cdot m_0)}{\beta} + \frac{(a_0 \cdot 2^{\ell_0})}{\beta} + (a_1 \cdot 2^{\ell_0}).$$

On remarque que dans le résultat final $(a_0 \cdot m_0)/\beta^2$ sera très peu significatif, car fractionnaire, et on peut se permettra donc de l'ignorer dans le calcul. Toutefois, il reste un grand nombre de divisions peu pratiques. Ainsi nous allons dans un premier temps noter $q = a \cdot m/\beta$ et l'approcher par :

$$a_1 \cdot m_0 + a_0 \cdot 2^{\ell_0} + \beta(a_1 \cdot 2^{\ell_0}) = a_1 \cdot m_0 + 2^{\ell_0} \cdot a$$

car $2^{\ell_0}(a_0 + \beta a_1)i = 2^{\ell_0} \cdot a$.

À l'instar des précédents algorithmes il faut vérifier si lors du calcul de r on a utilisé $\lfloor a/n \rfloor - 1$ en lieu et place de $\lfloor a/n \rfloor$. Pour le prouver, on va d'abord chercher à exprimer r avec les différentes parties que l'on tronque au cours de l'algorithme.

$$r = a - \left\lfloor \frac{q}{\beta} \right\rfloor \cdot n \bmod \beta$$

Puis on développe q avec la formule utilisée dans la ligne 2 de l'algorithme :

$$r = a_0 - \left\lfloor \frac{m_0 \cdot a_1 + 2^{\ell_0} \cdot a}{\beta} \right\rfloor \cdot n \bmod \beta$$

On a substitué a par a_0 pour la même raison que celle décrite pour la réduction de V. Shoup, car on sait que le résultat de la soustraction vérifie $r < \beta$. Maintenant, on va chercher à ajouter les deux erreurs à la formule de r.

La première erreur apparait dans le calcul de m lors de la troncature, c'est la partie fractionnaire de m. On nomme cette erreur ϵ et on a $\epsilon = \beta^2 - m \cdot n$. Afin d'introduire plus aisément ϵ dans l'expression de r, on détermine : $m_0 \cdot n = \beta^2 - \epsilon - \beta 2^{\ell_0} n$.

La seconde s'exprime lors de la division q effectuée en ligne 3. Comme $q = q_0 + \beta q_1$, $\lfloor q/\beta \rfloor = q_1$, on omet q_0, ce qui permet d'écrire $\lfloor q/\beta \rfloor = (q - q_0)/\beta$. On obtient alors pour r :

$$r = a_0 - \frac{a_1 \cdot \beta^2 - a_1 \cdot \epsilon - a_1 \cdot \beta 2^{\ell_0} n + a \cdot 2^{\ell_0} \cdot n - q_0 \cdot n}{\beta} \bmod \beta$$

On inverse le signe devant la fraction et l'on fait disparaitre $a_1 \cdot \beta^2/\beta$ car $a_1 \cdot \beta^2/\beta = a_1 \cdot \beta \equiv 0 \bmod \beta$; on a alors :

$$r = a_0 + \frac{a_1 \cdot \epsilon + a_1 \cdot \beta 2^{\ell_0} n - a \cdot 2^{\ell_0} \cdot n + q_0 \cdot n}{\beta} \bmod \beta \tag{1.2}$$

Maintenant, on réorganise 1.2 afin de faire apparaitre deux parties distinctes qui simpli-

fient la recherche d'une borne supérieure, ce qui donne :

$$r = \left(a_0 - a_0 \cdot \frac{2^{\ell_0} \cdot n}{\beta}\right) + \frac{a_1 \cdot \epsilon + q_0 \cdot n}{\beta}$$

car

$$(a_1 \cdot \beta 2^{\ell_0} n - a \cdot 2^{\ell_0} \cdot n)/\beta = (a_1 \cdot \beta 2^{\ell_0} n - (a_0 \cdot 2^{\ell_0} \cdot n + a_1 \beta \cdot 2^{\ell_0} \cdot n))/\beta = -a_0 \frac{2^{\ell_0} \cdot n}{\beta}.$$

À partir de cette expression de r on voit clairement se définir la borne supérieure de r qui est de $2 \cdot n$. En se rappelant que $a \leq \beta^2$, on a :

$$0 \leq \underbrace{\left(a_0 - a_0 \cdot \frac{2^{\ell_0} \cdot n}{\beta}\right)}_{<n,\ \text{car}\ \frac{2^{\ell_0} \cdot n}{\beta} < 1} + \underbrace{\frac{a_1 \cdot \epsilon + q_0 \cdot n}{\beta}}_{<n,\ \text{car}\ a_1 \epsilon\ \text{et}\ q_0 n < n^2} < 2 \cdot n$$

Ce qui montre pourquoi une soustraction conditionnelle finale est nécessaire et suffisante pour obtenir $r = a \bmod n$.

Avantages : Cette méthode de réduction modulaire est très performante si l'on respecte la contrainte imposée sur le module. De plus, on peut précalculer la valeur m ce qui est particulièrement avantageux par rapport à la multiplication modulaire de V. Shoup. Effectivement, lorsque l'on réalise des opérations arithmétiques sur les polynômes, le module utilisé est le même pour tous les coefficients. De plus, l'algorithme n'étant pas dépendant de l'opérande, il est tout à fait utilisable en dehors d'un protocole PIR par exemple.

Algorithm 5 Réduction modulaire

Require : $a = a_1, a_0$ et n vérifie (1.1), $1 \leq s_0 \leq \ell - 1$ bits de marge et $m_0 = \lfloor \beta^2/n \rfloor \bmod \beta$

```
procedure REDUCTION(a)
    q ← m_0 · a_1 + 2^{s_0} · a mod β^2          ▷ ≈ (v · a)/β
    r ← a − ⌊q/β⌋ · n mod β                      ▷ mullow
    if r ≥ n then
        r ← r − n
    end if
    return r
end procedure
```

1.6 Représentation CRT : théorème des restes Chinois

Jusqu'à maintenant, nous supposions qu'un registre pouvait contenir entièrement un module. Dans le cas contraire, deux possibilités s'offrent à nous.

La première possibilité est d'utiliser des bibliothèques spécialisées dans la gestion d'entiers de taille arbitraire. Cependant, ces bibliothèques proposent de très nombreuses fonctionnalités qui ne sont pas nécessaires lors d'opérations arithmétiques comme celles présentées précédemment. De plus, elles sont orientées vers un usage générale et donc ne peuvent pas se permettre le même niveau d'optimisation qu'en se focalisant sur le cas des réseaux de polynômes.

La seconde possibilité est de décomposer le module de grande taille en un ensemble de sous modules pouvant être contenus chacun dans un seul registre. Une fois les opérations arithmétiques effectuées, on assemble ces sous-modules afin de reconstruire le résultat. Cette approche est une application du théorème des restes chinois que l'on peut énoncer comme suit :

Définition 1. *Soient k nombres entiers $p_1, p_2, \dots, p_k$ premiers entre eux deux-à-deux et $p = \prod_{i=1}^{k} p_i$. Il existe un isomorphisme d'anneau entre l'anneau des entiers modulo p et le produit des anneaux des entiers modulo p_i pour i allant de 1 à k.*

La notion d'isomorphisme implique :
- qu'il existe une fonction bijective permettant de passer d'un entier x modulo p à un vecteur d'entiers $(r_1, r_2, \dots, r_k)$ modulo les premiers $p_1, p_2, \dots, p_k$ et inversement ;
- qu'il est possible de réaliser des sommes et des produits dans une situation (avec des nombres modulo p) comme dans l'autre (avec des vecteurs de nombres modulo $p_1, p_2, \dots, p_k$) et de passer d'une représentation à l'autre quand cela nous arrange.

Pour passer d'un grand nombre x modulo p à un vecteur de nombres il suffit de calculer les restes modulo les petits premiers $(r_1, r_2, \dots, r_k) = (x \bmod p_1, x \bmod p_2, \dots, x \bmod p_k)$. Le passage inverse est plus compliqué.

1.6.1 Relèvement

L'opération de reconstruction du résultat modulo un grand entier à partir des restes modulo des facteurs de cet entier est appelée relèvement.

La solution est donnée par la formule $x = r_1 \cdot (p/p_1) \cdot y_1 + \dots + r_k \cdot (p/p_k) \cdot y_k$ avec $y_i = (p/p_i)^{-1} mod p_i$ l'inverse de (p/p_i) modulo p_i.
Le calcul de $(p/p_i)^{-1}x$ se fait en combinant l'algorithme étendu d'Euclide et l'identité de Bézout.

Identité de Bézout : *Soient a et b deux entiers relatifs et d leur PGCD alors, il existe deux entiers u et v tels que $a \cdot u + b \cdot v = d$.*

Comme les premiers $p_1, p_2, \dots, p_k$ sont premiers entre eux, p/p_i et p_i sont premiers entre eux et l'identité de Bézout nous assure donc qu'il existe (u, v) tel que $(p/p_i) \cdot u + p_i \cdot v = 1$. Comme $(p/p_i) \cdot u = 1 \mod p_i$, on a u est l'inverse recherché. L'obtention de u et v se fait par l'algorithme d'Euclide étendu.

1.6.2 Représentation des polynômes modulo un grand nombre

Les polynômes avec lesquels nous travaillerons ont des coefficients réduits modulo un grand nombre p qui est le produit de premiers $p_1, p_2, \dots, p_k$. Les polynômes eux-mêmes, comme déjà dit, seront réduits modulo un polynôme de la forme $\mathrm{X}^n + 1$. Ainsi les coefficients d'un polynôme A pourront être représentés comme un vecteur de nombres modulo p, $(a_0, \dots, a_{n-1})$, mais aussi comme un vecteur de vecteurs $((a_{0,0}, \dots, a_{k-1,0}), \dots, (a_{0,n-1}, \dots, a_{k-1,n-1}))$ en utilisant la représentation CRT pour chacun des coefficients. Ainsi $a_{i,j}$ représente le j-ème coefficient du polynôme A, réduit modulo p_i.

De façon complètement équivalente, en réordonnant les coefficients, on peut représenter le polynôme par le vecteur $((a_{0,0}, \dots, a_{0,n-1}), \dots, (a_{k-1,0}, \dots, a_{k-1,n-1}))$, c'est à dire la juxtaposition de A mod p_1, A mod p_2, ..., A mod p_k. En pratique ceci équivaut à considérer que A est formé de k sous-polynômes qui sont les version réduites de A modulo $p_1, \dots, p_k$.

Pour cette représentation, nous devons définir de nouveaux algorithmes de somme et produit de polynômes.

1.6.3 Somme CRT

Pour la somme, l'algorithme calcule les additions pour chaque sous-module. Modifié, l'algorithme prend la forme suivante :

Soient A et B deux ensembles de k sous-polynômes de degrés $n-1$ dont les j-ème coefficients du i-ème module seront écrits respectivement $a_{i,j}$ et $b_{i,j}$. L'algorithme calculant C la somme de A et B s'écrit :

Algorithm 6 Somme de deux polynômes CRT

```
procedure SUM(A, B)
    for i ← 0, k do                                  ▷ Pour tous les modules
        for j ← 0, n do                              ▷ Pour tous les coefficients
            c_{i,j} ← a_{i,j} + b_{i,j} mod p_i
        end for
    end for
    return C
end procedure
```

Remarque.

Les sommes de polynômes étant indépendantes entre elles cet algorithme est facilement parallélisable.

1.6.4 Produit CRT

De la même façon que pour la somme, les multiplications sont effectuées entre les sous-polynômes correspondant à un même module. L'algorithme de produit devient :
Soient A et B deux ensembles de k sous-polynômes de degrés $n-1$ dont les j-ème coefficients du i-ème module seront écrits respectivement $a_{i,j}$ et $b_{i,j}$. L'algorithme calculant C le produit de A et B s'écrit :

Algorithm 7 Produit CRT de deux polynômes

```
procedure PRODUCT(A, B)
    for i ← 0, k do                                  ▷ Pour tous les modules
        for j ← 0, n do
            c_{i,j} ← 0
        end for
    end for
    for i ← 0, k do                                  ▷ Pour tous les modules
        for j ← 0, n do
            for l ← 0, n do
                if j + l < n then
                    c_{i,j+l} ← c_{i,j+l} + a_{i,j} × b_{i,l} mod p_i     ▷ 64 × 64 → 128bits
                end if
                if j + l >= n then
                    c_{i,j} ← c_{i,j} − a_{i,j} × b_{i,l}                 ▷ mod X^n + 1
                end if
            end for
        end for
    end for
    return C
end procedure
```

Remarque.

Comme pour la somme, cet algorithme peut aisément être parallélisé.

1.7 Caractérisation d'un polynôme

Dans les sections précédentes nous avons représenté les polynômes par leurs coefficients. Représentation utilisée intuitivement pour, par exemple, comparer deux polynômes. Cependant il existe d'autres représentations caractéristiques de ces objets mathématiques comme les racines ou même une séquence de valeurs obtenue par des évaluations successives d'un polynôme.

De ces différentes représentations découlent différents moyens de réaliser des opérations entre polynômes.

1.7.1 Coefficients

Les polynômes utilisés dans la cryptographie fondée sur les réseaux euclidiens demande qu'une représentation par coefficients soit utilisée à certains moments. En effet, elle donne la possibilité de faire la différence entre les bits de poids fort et les bits de poids faible des coefficients.

1.7.2 Valeurs

Une représentation par valeurs signifie que l'on va évaluer le polynôme en un ensemble de points et c'est l'ensemble des valeurs calculées suite à ces multiples évaluations qui caractérise le polynôme.

Cependant il faut être attentif au nombre d'évaluation ainsi qu'aux valeurs sur lesquelles ces évaluations sont faites.

En effet, on veut que l'ensemble des valeurs obtenues permette de caractériser à coup sûr le polynôme, c'est-à-dire qu'il nous faut une bijection entre la représentation par coefficient et la représentation par évaluation.

1.7.2.1 Évaluation

Passer de la représentation par coefficient à la représentation par valeurs est intuitif. Supposons que l'on souhaite autant de valeurs évaluées que de coefficients, on peut écrire l'algorithme suivant :

Algorithm 8 Évalue le polynôme P sur n entiers de 0 à $n-1$

```
function EVAL(P)
    for s_i ← 1, n − 1 do
        s_i ← 0
        for j ← 0, n − 1 do
            s_i ← p_j · i^j + s_i
        end for
    end for
    return s_i
end function
```

Il est important de noter que cette la représentation d'un polynôme de degré $n-1$ par n valeurs en n points distincts constitue une caractérisation (i.e. il y a un unique polynôme donnant ces n valeurs en ces n points). Il est facile de prouver ceci car si P et P' ont les n mêmes valeurs alors $\mathrm{P}-\mathrm{P}'$ vaut 0 en n points différents. Comme $\mathrm{P}-\mathrm{P}'$ est de degré au plus $n-1$, s'il admet n racines c'est alors le polynôme nul et $\mathrm{P}=\mathrm{P}'$, et donc il y a bien un seul polynôme pour un ensemble de n valeurs.

1.7.2.2 Interpolation

L'opération inverse, passer de la représentation par valeurs à la représentation par coefficients, est nommée interpolation. L'interpolation se fera en deux étapes, la première consiste à trouver des polynômes dits polynômes de Lagrange, notés L_i, pour chacun des

points. Autrement dit, pour interpoler un polynôme sur n points, on calcule n polynôme de Lagrange. Puis, en utilisant les polynômes de Lagrange on détermine la fonction polynomiale $\mathrm{L}(\mathrm{X})$ passant par ces points.

Ainsi les polynômes de Lagrange L_i sont donnés par :

$$\mathrm{L}_i(\mathrm{X}) = \prod_{j=0, j\neq i}^{n-1} \frac{\mathrm{X} - x_j}{x_i - x_j} = \frac{\mathrm{X} - x_0}{x_i - x_0} \dots \frac{\mathrm{X} - x_{i-1}}{x_i - x_{i-1}} \dots \frac{\mathrm{X} - x_{i+1}}{x_i - x_{i+1}} \dots \frac{\mathrm{X} - x_n}{x_i - x_n}$$

Puis le polynôme interpolé ayant les valeurs $(y_0, \dots, y_{n-1})$ est donné par :

$$\mathrm{L}(\mathrm{X}) = \sum_{i=0}^{n-1} y_i \mathrm{L}_i(\mathrm{X})$$

EXEMPLE 1.

Afin d'illustrer le processus d'interpolation, on va chercher à déterminer le polynôme de degré 2 passant par les points : $((x_0 = 1, y_0 = 2), (x_1 = 2, y_1 = -1), (x_2 = -1, y_2 = 1))$

1. On calcule les polynômes de Lagrange

$$l_0 = \frac{(\mathrm{X}-1)(\mathrm{X}+1)}{(1-2)(1+1)} = \frac{1}{2}\mathrm{X}^2 + 2\mathrm{X} + 1$$
$$l_1 = \frac{(\mathrm{X}-1)}{\mathrm{X}+1}(2-1)(2+1) = \frac{1}{3}\mathrm{X}^2 - 1$$
$$l_2 = \frac{(\mathrm{X}-1)(\mathrm{X}-2)}{(-1-1)(-1-2)} = -\frac{1}{6}\mathrm{X}^2 - 3x - 2$$

2. On détermine le polynôme d'interpolation

$$\mathrm{L}(\mathrm{X}) = \mathrm{P}(\mathrm{X}) = 2l_0 + -1l_1 + l_2 = \frac{1}{2}\mathrm{X}^2 + \frac{5}{6}\mathrm{X} + \frac{2}{3}$$

L'interpolation de Lagrange nécessite $\mathcal{O}(n^2)$ opérations, puisqu'il faut sommer n polynômes de degré $n-1$. Ce coût, qui peut paraitre faible compte tenu de l'exemple donné précédemment, devient majeur lorsque le nombre de coefficients des polynômes augmente. Pour un polynôme de 1000 coefficients, 1 000 000 d'opérations seront nécessaires.

1.7.2.3 Multiplication et somme par valeurs

Si A et B sont deux polynômes de degré $n-1$ évalués en n points par $(a_0, \dots, a_{n-1})$ et $(b_0, \dots, b_{n-1})$ leur somme $\mathrm{A} \cdot \mathrm{B}$ aura en ces mêmes n points les valeurs $(a_0 + b_0, \dots, a_{n-1} + b_{n-1})$. Ainsi il est possible de sommer deux polynômes sans revenir à une représentation par coefficients.

Si on souhaite calculer leur produit $\mathrm{A} \cdot \mathrm{B}$, celui-ci aura en ces mêmes n points les valeurs $(a_0 \cdot b_0, \dots, a_{n-1} \cdot b_{n-1})$. Cependant $\mathrm{A} \cdot \mathrm{B}$ est un polynôme de degré $2n-2$ et pour être caractérisé par un vecteur de valeurs il faudrait avoir $2n-1$ valeurs.

Ainsi si on souhaite obtenir un vecteur de valeurs caractérisant $\mathrm{A} \cdot \mathrm{B}$ ils nous faut avoir les évaluations de A et B en $2n-1$ points. Supposons que nous disposions d'une telle évaluation et que nous calculions $(a_0 \cdot b_0, \dots, a_{2n-2} \cdot b_{2n-2})$. Cette multiplication est représentée dans le schéma 1.2.

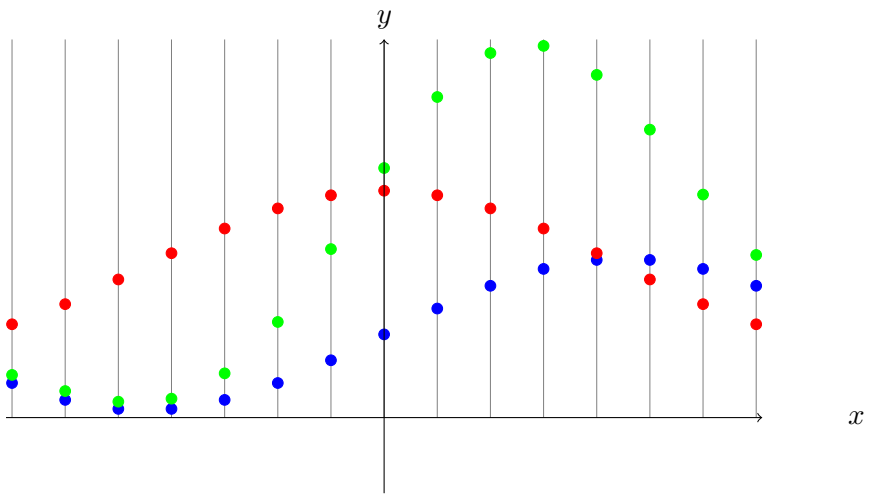


Figure 1.2 – Représentation de la multiplication de A en bleu et B rouge sous forme de points évalués. Le résultat est représenté par les points verts.

Le vecteur résultat caractérise $A \cdot B$. Il est remarquable de voir que pour passer de vecteurs caractérisant A et B par valeurs, à un vecteur caractérisant $A \cdot B$ il suffit de faire $O(n)$ multiplications. Si on avait procédé par coefficients il aurait fallu $O(n^2)$ opérations.

Bien sûr si on part d'une représentation de A et B par coefficients il faut $O(n^2)$ opérations pour transformer celles-ci en une représentation par valeurs, puis $O(n)$ opérations pour obtenir la représentation de $A \cdot B$ par coefficients, et enfin $O(n^2)$ opérations pour obtenir la représentation de $A \cdot B$ par coefficients (en faisant une interpolation). Du coup le coût total est inchangé asymptotiquement. Mais si on a plusieurs sommes et multiplications à réaliser on peut imaginer ne faire les opérations les plus coûteuses (évaluation et interpolation) qu'en début et fin de calcul et donc de les amortir.

1.8 La transformée de Fourier

1.8.1 Introduction

On a vu dans la section 1.7.2 que l'on pouvait calculer le produit de deux polynômes en les évaluant sur $2n$ points, en multipliant ces points 2 à 2 puis en interpolant le résultat afin de trouver le polynôme passant par ces points. Cependant les opérations d'évaluation et d'interpolation sont au moins aussi couteuses que la multiplication naïve des deux polynômes, ce qui rend la multiplication point à point inutilisable.

Toutefois, il existe une méthode appelée transformée de Fourier discrète (proche de notre évaluation) et son inverse (proche de notre interpolation) suffisamment efficace pour qu'au final la multiplication point à point soit plus rapide que la multiplication naïve. Dans notre contexte, où on travaille modulo un polynôme $X^n + 1$, un deuxième intérêt de la transformée de Fourier (plus précisément, des points d'évaluation dans la transformée de Fourier) est qu'il ne sera pas nécessaire d'avoir plus de n évaluation, même quand on fait des multiplications.

L'idée originale est qu'il est possible, sous certaines conditions, de décomposer une fonction périodique sous la forme d'une somme infinie de signaux sinuoidaux. En d'autres termes la transformée de Fourier transforme un signal temporel en une somme de sinus appelés fréquences. La fréquence peut se définir comme le nombre de fois que le sinus vaut 1 par unité de temps. Comme la fonction sinus est 2π-périodique, c'est-à-dire qu'elle vaut 1 en $\pi, 3\pi, 5\pi, \cdots$; chaque fois que l'axe des x prendra une de ces valeurs, on aura fait un tour du cercle trigonométrique.

Afin de donner un exemple prenons le domaine de la compression du son avec perte. Cette opération de compression transforme un son, typiquement de la musique, en un un autre son similaire à l'écoute dans lequel on aura réduit la précision de fréquences auxquelles

l'oreille humaine est peu sensible et supprimé les fréquences inaudibles. Le procédé simplifie de compression avec pertes est le suivant :

1. On transforme le signal original en somme de fréquences par la transformée de Fourier.
2. On approxime les fréquences peu sensibles pour l'oreille humaine. Par exemple si les données de la transformée tiennent sur 16 bits, on peut approximer certaines fréquences sur 4 bits plutôt que 16, que l'on multipliera par 4096 lors de la reconstruction du signal de son. Ainsi on réduit la taille du signal.
3. On supprime les fréquences inaudibles pour l'oreille, c'est-à-dire toutes les fréquences au dessus d'un certain seuil. Cette suppression est un filtre passe bas.

1.8.2 Approche intuitive

Supposons que l'on souhaite décomposer le signal de valeurs $[4, 0, 0, 0]$ en une somme de sinus. Comme on a 4 points à décomposer, on aura besoin de 4 fréquences : 0Hz, 1Hz, 2Hz, 3Hz. La fréquence 0Hz à une valeur fixe elle ne « tourne » pas sur le cercle trigonométrique, 1Hz correspond à 1 tour par unité de temps, 2Hz correspond à deux tours et n Hz correspond à n tours. Dans notre décomposition, si toute fréquence démarrent à 1, la valeur obtenue en 0 sera bien de 4 car $sin(0) = 1$. Cependant, comment se comportent ces fréquences pour les 3 unités restantes ? On peut construire le tableau suivant :

fréq \ temps	0	1	2	3
0 Hz	(1,0)	(1, 0)	(1, 0)	(1, 0)
1 Hz	(1,0)	(0, i)	(-1, 0)	(0, -i)
2 Hz	(1,0)	(-1, 0)	(1, 0)	(0, 1)
3 Hz	(1,0)	(0, -i)	(-1, 0)	(-1, 0)
$\sum$	**(4,0)**	(0,0)	(0,0)	(0,0)

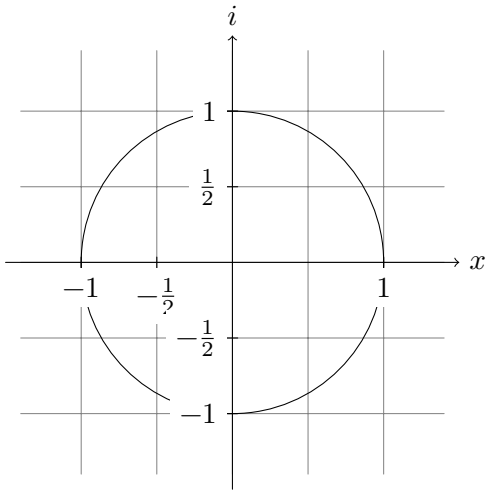


Table 1.1 – Décomposition en sinus d'un signal $[4, 0, 0, 0]$ en fréquence $1, 1, 1, 1$.

Le tableau 1.1 représente la décomposition en sinus du signal $[4, 0, 0, 0]$ en fréquences $1, 1, 1, 1$. Chaque ligne représente la position, de la fréquence durant 4 unités de temps. La dernière ligne est la somme des 4 fréquences par unité de temps. On remarque que dans ce tableau, faire démarrer les 4 fréquence à 1, permet de décomposer correctement le signal discret. En effet à t_0 les fréquences s'additionnent toutes et donnent la valeur 4. Puis elles s'annulent les unes par rapport aux autres ce qui donne les trois 0 restant du signal.

On peut voir par cet exemple qu'il est possible de représenter un signal comme une somme de sinus. Cette opération de décomposition que nous avons réalisée de façon intuitive peut s'écrire de manière formelle en utilisant la notation trigonométrique du cercle présentée plus tôt.

1.8.3 Représentation sous forme de sommes

Définition 2. *Soit un point p quelconque sur le cercle unité, on appelle θ la mesure de l'angle de sommet 0 formé par l'axe des abscisses, le point 0 et le point p. Les coordonnées du point p peuvent s'écrire* $(\cos\theta, \sin\theta) = (\cos(\theta + 2\pi), 2\pi \sin(\theta + 2\pi))$ *car les fonctions sinus et cosinus sont de période* 2π.

Évaluation : Dans la transformée de Fourier ont cherche à décomposer un signal discrétisé, c'est-à-dire une séquence d'entiers en une somme de fonctions trigonométriques. Pour trouver chacune des fréquences, on va projeter, pour une fréquence donnée, les points du signal sur le cercle trigonométrique afin d'obtenir pour chacun des points sa contribution à la fréquence, puis faire la somme de toutes les contributions obtenues. Ainsi on obtient, pour la fréquence donnée, sa valeur dans le signal discrétisé. Autrement dit, on peut voir le calcul de fréquences comme une forme particulière évaluation du polynôme sur le cercle unité. Soit un polynôme $\mathrm{P(X)}$ ayant pour coefficients $(c_0, \cdots, c_{n-1})$; on divise virtuellement le cercle en n points. Ainsi, déterminer la pondération d'une fréquence k dans la séquence d'échantillons, revient, à pondérer chaque point l du cercle par le nombre complexe qui lui est associé. Pour trouver ce complexe, on utilise la notation trigonométrique ou exponentielle de l'angle formé par l'axe des abscisses et le point l du cercle. Comme un tour complet du cercle donne l'angle 2π et que l'on divise le cercle en n points, la mesure de l'angle θ du $l-$ème point à la fréquence k est donné par $\theta = 2\pi k \frac{l}{n}$. Il est primordial de multiplier par k car plus la fréquence est élevée plus les points évalués « tournent rapidement » autour du cercle. Le nombre complexe correspondant est donné par $(\cos(-2\pi k \frac{l}{n}) + i \sin(-2\pi k \frac{l}{n})$. Comme on cherche la pondération totale de chacune des $n-1$ fréquences ; pour la $k-$ème fréquence la formule de la transformée de Fourier est :

$$f_k = \sum_{l=0}^{n-1} c_l \cdot (\cos(2\pi k \frac{l}{n}) + i \sin(2\pi k \frac{l}{n})), \quad k \in \mathbb{Z}$$

Ce qui donne en notation exponentielle :

$$f_k = \sum_{l=0}^{n-1} c_l \cdot e^{2i\pi k \frac{l}{n}} \tag{1.3}$$

Interpolation : Pour inverser la TFD il faut extraire les coefficients à partir des fréquences. Il faut donc, pour chaque correspondance retrouver la mesure de l'angle du point correspondant sur le cercle et le multiplier par la fréquence k pour extraire la valeur de la pondération du coefficient recherché dans la fréquence k. Ainsi en sommant la contribution du coefficient à chaque fréquence on retrouve le coefficient. La formule de la transformée de Fourier inverse pour retrouver le l-ème coefficient est :

$$c_l = \frac{1}{n} \sum_{k=0}^{n-1} f_k \cdot (\cos(-2\pi k \frac{l}{n}) + i \sin(-2\pi k \frac{l}{n})), \quad l \in \mathbb{Z}$$

Ce qui donne en notation exponentielle :

$$c_l = \frac{1}{n} \sum_{k=0}^{n-1} f_k \cdot e^{-2i\pi k \frac{l}{n}} \tag{1.4}$$

1.8.4 Racines n-èmes de l'unité

L'équation $z^n = 1$ admet n solutions complexes. Cela signifie que le nombre 1 admet n racines n-èmes. Celles-ci ont pour images dans le plan complexe les sommets d'un polygone régulier inscrit dans le cercle de centre le point d'affixe 0 et de rayon 1, avec un sommet de coordonnées (1, 0). Les coordonnées de ces points, ou encore les parties réelles et imaginaires des racines n-èmes sont $(\cos(2\pi l/n), \sin(2\pi l/n))$ pour l allant de 0 à $n-1$.

EXEMPLE 2.

On peut remarquer qu'en élevant au carré les huit racines 8 èmes de 1, on trouve les 4 racines 4 èmes de 1, chacune étant répétées deux fois :

$$\begin{aligned}
(\omega_8^0)^2 &= \omega_4^0 = 1 \\
(\omega_8^1)^2 &= \omega_8^2 = \omega_4^1 \\
(\omega_8^2)^2 &= \omega_8^4 = \omega_4^3 \\
(\omega_8^3)^2 &= \omega_8^6 = \omega_4^3 \\
(\omega_8^4)^2 &= \omega_8^8 = \omega_4^0 \\
(\omega_8^5)^2 &= \omega_8^2 = \omega_4^1 \\
(\omega_8^6)^2 &= \omega_8^4 = \omega_4^2 \\
(\omega_8^7)^2 &= \omega_8^6 = \omega_4^0
\end{aligned}$$

On note aussi que les quatre dernières racines 8-èmes sont les opposées des quatre premières, par exemple $\omega_8^5 = -\omega_8^1$, ce qui se généralise avec la formule :

$$\omega_n^{l+n/2} = -\omega_n^l \tag{1.5}$$

1.8.5 Notation matricielle

Si l'on reprend la notation exponentielle 1.3 de la TFD, on voit que l'on peut la réécrire comme suit :

$$f_k = \sum_{l=0}^{n-1} c_l \cdot \left(e^{\frac{2\pi i}{n}}\right)^{l*k} \tag{1.6}$$

Ce qui fait apparaitre la constante $\omega = e^{\frac{2\pi i}{n}}$.

Notons que la transformée de Fourier peut être vue comme le vecteur obtenu en évaluant un polynôme P(X) sur les n racines n-ème de l'unité, $\mathrm{TFD}(\mathrm{P(X)}) = (\mathrm{P}(\omega_n^0), \mathrm{P}(\omega_n^1), \cdots, \mathrm{P}(\omega_n^{n-1}))$. Si on écrit le calcul des différentes fréquences sous forme de matrice la liste des $e^{-2i\pi k\frac{l}{n}}$ pour chaque fréquence calculée, on obtient une matrice carrée de côté n que l'on multiplie par les échantillons, c'est-à-dire une ligne composée de c_l. On a alors :

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{n-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \cdots & w^{(n-1)^2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix}, w = e^{-\frac{2\pi i}{n}}$$

On note que l'on a utilisé les racines n-ièmes de 1 plutôt que d'autres nombres. En effet, le fait de faire intervenir des cosinus et des sinus, qui sont les coordonnées des racines de l'unité, rapproche la transformée de Fourier discrète de la transformée de Fourier classique, telle qu'on la définit dans le domaine du continu, et par la même occasion cela rend la transformation inverse très simple, puisqu'elle se ramène à une interpolation.

Sous cette forme, l'interpolation revient à multiplier les f_k par l'inverse de la matrice des racines n-èmes de l'unité. On a alors :

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{n-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \cdots & w^{(n-1)^2} \end{pmatrix}^{-1} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

Remarque.

On constate intuitivement que ces deux équations de TFD et de TFDi demandent $\mathcal{O}(n^2)$ opérations sur un ordinateur, ce qui ne donne pas d'avantage par rapport à une évaluation et une interpolation de polynôme classique sur des valeurs arbitraires. Cependant, on voit par l'écriture matricielle que de nombreuses puissances de ω se retrouvent de ligne en ligne. Il existe un algorithme pour calculer la TFD et la TFDi exploitant ces redondances qui permet de réduire grandement le nombre d'opérations nécessaires.

1.9 Transformée de Fourier discrète rapide

La TFDR est une méthode mathématique et algorithmique qui calcule la transformée de Fourier décrite dans la section précédente en $\mathcal{O}(n \log n)$ opérations.

Attention Dans notre contexte de travail n est une puissance de deux, ce qui simplifie l'algorithme.

Soit $\mathrm{P(X)} = (c_0 + c_1\mathrm{X} + \cdots + c_{n-1}\mathrm{X}^{n-1})$ un polynôme de degré $n-1$ (avec n une puissance de 2), on peut alors réorganiser ses coefficients en deux parties, d'un coté les coefficients pairs de l'autre les impairs. On a alors :

$$\mathrm{P(X)} = (c_0 + c_2\mathrm{X}^2 + \cdots + c_{(n-1)/2}\mathrm{X}^{n-1/2}) + \mathrm{X}(c_1 + c_3\mathrm{X}^2 + \cdots + c_{(n-2)/2}\mathrm{X}^{n-1/2})$$

Ce que nous noterons plus simplement :

$$\mathrm{P(X)} = \mathrm{P}_{pair}\mathrm{X}^2 + \mathrm{X} \cdot \mathrm{P}_{impair}\mathrm{X}^2$$

Maintenant remplaçons X par une racine n-ème de l'unité. On a vu dans la section 1.8.4 que le fait d'élever au carré transformait une racine n-ème en une racine $n/2$-ième et cela à deux reprises lorsque l'on prend toutes les racines n-èmes.

exemple 3.

On prend $\mathrm{N} = 8$:

$$\begin{aligned}
\mathrm{P}(\omega_8^0) &= \mathrm{P}_{pair}(\omega_4^0) + \omega_8^0\mathrm{P}_{impair}(\omega_4^0) \\
\mathrm{P}(\omega_8^1) &= \mathrm{P}_{pair}(\omega_4^1) + \omega_8^1\mathrm{P}_{impair}(\omega_4^1) \\
\mathrm{P}(\omega_8^2) &= \mathrm{P}_{pair}(\omega_4^2) + \omega_8^2\mathrm{P}_{impair}(\omega_4^2) \\
\mathrm{P}(\omega_8^3) &= \mathrm{P}_{pair}(\omega_4^3) + \omega_8^3\mathrm{P}_{impair}(\omega_4^3) \\
\mathrm{P}(\omega_8^4) &= \mathrm{P}_{pair}(\omega_4^0) + \omega_8^0\mathrm{P}_{impair}(\omega_4^0) \\
\mathrm{P}(\omega_8^5) &= \mathrm{P}_{pair}(\omega_4^1) + \omega_8^1\mathrm{P}_{impair}(\omega_4^1) \\
\mathrm{P}(\omega_8^6) &= \mathrm{P}_{pair}(\omega_4^2) + \omega_8^2\mathrm{P}_{impair}(\omega_4^2) \\
\mathrm{P}(\omega_8^7) &= \mathrm{P}_{pair}(\omega_4^3) + \omega_8^3\mathrm{P}_{impair}(\omega_4^3)
\end{aligned}$$

On constate que les répétitions des racines dans ces équations font qu'évaluer un polynôme sur n valeurs (avec n^2 opérations) revient à évaluer deux polynômes de longueur moitié sur $n/2$ valeurs (avec $2 * n^2/4$ opérations). Ainsi le coût est divisé par 2. Si n est une puissance de deux il est alors possible de recommencer $\log n$ fois cette subdivision et de gagner un facteur 2 sur les coûts à chaque fois. Le gain total est donc de $2^{\log n} = n$. En prenant en compte les multiplications hors évaluation polynomial et les sommes le coût final est de $\mathrm{O}(n \log n)$.

1.10 Transformée de Fourier rapide sur un corps fini

On a vu, dans la section précédente la transformée de Fourier d'une séquence de valeurs ou d'un polynôme P de degré $n-1$ dans le corps des complexes $\mathbb{C}$. Cette évaluation consiste à évaluer n racines n-èmes de l'unité dans $\mathbb{C}$ de la forme ω^i avec ω une racine primitive n-ème de l'unité. Le choix des racines de n-ièmes de l'unité s'explique par le fait qu'elles simplifient le calcul de la transformée de Fourier en faisant apparaître des symétries dans les calculs.

Dans cette partie, on va généraliser la transformée rapide dans un corps qui n'est plus celui des complexes, mais dans $\mathbb{Z}/p\mathbb{Z}$) avec p un nombre premier. Comme il est habituel en cryptographie on notera ce corps $\mathbb{Z}_p$. Cette structure est un corps fini, et son sous-groupe multiplicatif est cyclique d'ordre $p-1$. En d'autres termes il existe un g, appelé générateur, dont les puissances $(g, g^2, \dots, g^{p-1})$ modulo p engendrent tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ sauf 0 avec en particulier $g^{p-1} = 1$ et $g^i! = 1$ pour $i < p-1$ (g est donc une racine primitive $p-1$-ème de l'unité.

La transformée de Fourier dans ce corps est nommée en Anglais *Number Theoretic Transform* et elle utilise des racines primitives n-ièmes de l'unité qui sont dans $\mathbb{Z}_p$ et pas des nombres complexes.

Définition 3. *On dit que ω est une racine primitive n-ème de l'unité dans $\mathbb{Z}_p$ si :*

1. $\omega^n = 1$
2. $\omega^i \neq 1$ *pour i entre* 1 *et* $n-1$

Nous choisirons systématiquement p de la forme $1 + k * 2n$ avec k une constante quelconque et n le degré du polynôme quotient $\mathrm{X}^n + 1$ que nous souhaitons utiliser. Comme le générateur g engendre le groupe multiplicatif de taille $p - 1 = k * 2n$ il suffit de prendre g^k pour avoir une racine $2n$-ième de l'unité ou g^{2k} pour avoir une racine n-ième de l'unité.

EXEMPLE 4.

- Soit $p = 17$ et $\mathrm{N} = 8$ une puissance de 2 qui divise $p-1$. Un générateur $\mathcal{U}(17)$ est 3. Un élément d'ordre $\mathrm{N} = 8$ est $3^{16/8} = 9$. Son inverse 2 est aussi d'ordre 8, comme on peut le vérifier en prenant ses puissances successives 2,4,8,16,15,13,9,1.
- Soit $p = 17$ et $\mathrm{N} = 16$ une puissance de 2 qui divise $p-1$. À partir du raisonnement précédent, une racine primitive est $\omega = 3$ qui est un générateur, et son inverse 6 est aussi d'ordre 16.
- Soit $p = 97$ et $\mathrm{N} = 32$. Un générateur de $\mathcal{U}(97)$ est 5. On prend comme racine primitive 32-ème de l'unité $\omega = 5^{96/32} = 5^3 = 28$.

1.10.1 Produit de deux polynômes

Nous avons expliqué pourquoi la transformée de Fourier rapide est une évaluation de polynôme en n points en $\mathcal{O}(n \log n)$ opérations. La transformée inverse demande un nombre d'opérations du même ordre. Les performances de ces deux méthodes légitiment l'utilisation de la représentation évaluée des polynômes dans le dessin de calculer leurs produits.

À première vue, cette méthode dispose d'une contrainte équivalente à celle discutée dans la partie 1.7.2. Les polynômes dont on souhaite obtenir le produit sont tenus d'être évalués sur $2n$ points, en considérant comme nuls les coefficients entre $\mathrm{X}^n + 1$ et X^{2n} des

polynômes à multiplier initialement, pour ensuite interpoler le polynôme de degré $2(n-1)$, résultat du produit.

Soit la fonction `NTT` qui effectue la transformée de Fourier rapide en utilisant des racines $2n$-ièmes de l'unité pour un polynôme de degré $n-1$ et `iNTT` la fonction effectuant la transformée inverse, l'algorithme décrit la multiplication de deux polynômes F et G.

Algorithm 9 Multiplication de deux polynômes, F et G sous forme évaluée par la NTT.

```
1 : function MULTIPLY(F, G)
2 :     F' ← NTT(F)
3 :     G' ← NTT(G)
4 :     for i ← 0, 2(n − 1) do
5 :         r'_i ← f'_i × g'_i
6 :     end for
7 :     R ← iNTT(R')
8 :     return R
9 : end function
```

Cet algorithme retourne un polynôme de degré $2(n-1)$. Toutefois, nous avons vu dans le chapitre 1 que la structure utilisée dans la cryptographie fondée sur les réseaux implique de réduire les polynômes modulo un autre polynôme de la forme X^n+1. L'algorithme 9 retourne un polynôme de degré $2(n-1)$ qu'il faut alors réduire modulo X^n+1. Cette opération revient à soustraire aux n premiers coefficients les n suivants.

Ainsi, il est possible d'optimiser cette réduction modulaire en restreignant à n le nombre de points évalués pour les deux polynômes. Nous allons montrer que le polynôme de degré $n-1$ résultant de la réduction d'un polynôme P de degré $2n-1$ est égal au polynôme obtenu par la transformée de Fourier de P sur n points, suivi de sa transformée inverse. Soit P un polynôme de degré $2n-1$ dont le i-ème coefficient est noté c_i.

$$\mathrm{P}(\mathrm{X}) = c_0 + c_1\mathrm{X}^1 + \cdots + c_{n-1}\mathrm{X}^{n-1} + c_n\mathrm{X}^n + \cdots + c_{2n-1}\mathrm{X}^{2n-1}$$

En calculant $\mathrm{P}'_r(\mathrm{X}) = \mathrm{P}(\mathrm{X}) \bmod \mathrm{X}^n+1$, on obtient sans factoriser les coefficients :

$$\mathrm{P}'(\mathrm{X}) = c_0 + c_1\mathrm{X}^1 + \cdots - c_{n-1}\mathrm{X}^{n-1} - c_n + \cdots + c_{2n-1}\mathrm{X}^{n-1}$$

En outre, lorsque l'on évalue $\mathrm{P}(\mathrm{X})$ sur les racines $2n$-èmes de l'unité on remarque par la propriété (1.5) des racines que $\omega_{2n}^{k+n} = -\omega_{2n}^{k}$. Évaluer $\mathrm{P}(\mathrm{X})$ sur n racines de l'unité revient à évaluer directement $\mathrm{P}'(\mathrm{X})$ car les racines de l'unité transforment les coefficients au-delà du $(n-1)$-ème coefficient d'une manière semblable à réduction plus générale de P par X^n+1. Ainsi, lorsque l'on interpole l'ensemble de valeurs évaluées de $\mathrm{P}(\mathrm{X})$ sur n racines de l'unité, on obtient directement $\mathrm{P}'(\mathrm{X})$.

1.11 Conclusion

Nous avons vu dans ce chapitre les outils, et plus particulièrement les polynômes dans la structure particulière qu'est le réseau euclidien d'idéaux ainsi que les opérations d'additions et de multiplication, nécessaires à la bonne compréhension du reste de ce mémoire. De nombreuses références y seront faites tout au long des chapitres qui suivent afin de rappeler les notions abordées.

Chapitre 2

Chiffrement homomorphe

Table des matières

2.1 Introduction 23
2.2 Prémices 25
2.2.1 Hypothèse 25
2.2.2 Chiffrements homomorphes multiplicatifs 26
2.3 Chiffrement de Paillier et généralisation 28
2.3.1 Chiffrement de Paillier 28
2.3.2 Généralisation du cryptosystème 28
2.3.3 Inconvénients majeurs 28
2.4 Chiffrement fondé sur les réseaux euclidiens 28
2.4.1 SIS et cryptosystèmes 28
2.4.2 LWE, Ring-LWE et cryptosystèmes 30
2.5 Conclusion 33

Résumé : *Les systèmes de chiffrement homomorphes possèdent des propriétés particulièrement intéressantes dans le cadre de la protection de la vie privée. Nous faisons ici l'état de l'art d'un outil qui sera fondamental dans les constructions que nous proposons par la suite.*

2.1 Introduction

Le chiffrement est une tentative de réponse au problème suivant :

> *Comment assurer le secret d'une communication lorsque celle-ci est transmise sur un canal insécurisé ?*

Depuis plusieurs millénaires, les groupes gouvernants ont recherché des moyens de communication efficaces et sûrs afin de maitriser la diffusion d'informations. Que ce soit pour régenter un territoire ou commander une armée le danger inhérent à la capture de ce type de messages par des adversaires fut rapidement assimilé. On retrouve des pratiques de brouillage de l'information dès l'époque antique avec par exemple la Scytale – bâton autour duquel le message dissimulé doit être enroulé pour être lisible – chez les Spartiates [52]. Intuitivement, le problème du transfert d'informations apparait lorsque deux parties distinctes communiquent par le biais d'un moyen de transmission sensible à la fuite et/ou à l'interception d'informations. Les deux parties cherchent à échanger des informations en tentant de minimiser les fuites vers le potentiel adversaire corrompant le canal de transmission. La solution canonique moderne à ce problème est d'employer un système cryptographique de chiffrement ou cryptosystème. Sommairement, un cryptosystème est un protocole qui donne des garanties de secret des communications entre plusieurs parties distinctes.

L'un des algorithmes d'un système cryptographique est appelé *chiffrement* – dans un protocole cryptographique, il est exécuté par l'émetteur du message – tandis que son inverse

– exécuté par le receveur – est nommé *déchiffrement*. Ainsi avant d'envoyer un message l'émetteur applique l'algorithme de chiffrement avec en entrée son message. L'exécution de l'algorithme terminée, il envoie le résultat, nommé *chiffré* sur un canal non sécurisé. Le chiffré reçu, le receveur exécute l'algorithme de déchiffrement et récupère le message initial, appelé *texte clair*.

Pour que le protocole décrit précédemment assure la confidentialité d'un échange , le receveur se doit de connaitre une information — un secret — inconnue de l'adversaire. Dans le cas contraire, l'adversaire pourrait, comme le receveur, déchiffrer le message chiffré et lire le texte clair. La connaissance supplémentaire nécessaire au receveur peut prendre la forme de l'algorithme de déchiffrement ou d'un paramètre supplémentaire utilisé par l'algorithme de déchiffrement. Cette information supplémentaire est appelée *Clé de déchiffrement*.

Les systèmes de chiffrement «traditionnels» utilisent un secret partagé entre l'algorithme de chiffrement et de déchiffrement. Ces systèmes de chiffrement sont appelés *chiffrements symétriques* et impliquent que le secret partagé soit transmis sur un canal sécurisé. Une analogie, non exempte de défauts, peut prendre la forme suivante. L'individu A souhaite 1) envoyer un colis à l'individu B et 2) que le contenu du colis soit uniquement accessible à B. Les deux individus vont en premier lieu s'accorder sur le cadenas à utiliser pour fermer le colis et une fois le cadenas choisi, faire un double de la clé (une pour A et une pour B). Ainsi, lorsqu' A envoie un colis à B – souhait 1) – , A le ferme avec le cadenas choisi précédemment pour être certain que seul B pourra, une fois reçu l'ouvrir – souhait 2) –. Cependant, le problème d'échange de clés apparait lorsqu'on ne dispose pas d'un canal confidentiel pour transmettre la clé.

Le principe retenu pour résoudre ce problème est de créer une clé de chiffrement distincte qui, même si elle est récupérée par l'adversaire, garantit la confidentialité du message. De plus, cela implique que la clé de déchiffrement ne se dérive pas facilement de la clé de chiffrement. Le cryptosystème est alors nommé *asymétrique* car il utilise deux clés différentes. Reprenons l'analogie précédente avec les individus A et B. A souhaite envoyer un colis à B dans un protocole proche du chiffrement asymétrique. En premier lieu, B envoie un cadenas *ouvert* à A dont B est *seul* possesseur de la clé. A va remplir le colis destiné à B, le sécuriser en fermant le cadenas puis l'envoyer – ils respectent 1) et 2) –. On observe que B est le seul à pouvoir ouvrir le cadenas et donc accéder au contenu du colis. Le cadenas peut être vu comme la fonction de chiffrement et la méthode de fermeture comme la clé. Ainsi B peut distribuer publiquement son cadenas ainsi que sa méthode de fermeture sans compromettre la sécurité du contenu du colis. Les protocoles asymétriques, plus couteux en matière de calculs sont aujourd'hui principalement utilisés afin de partager un secret qui sera utilisé dans un deuxième temps comme clé partagée par un cryptosystème symétrique moins couteux.

En 2008 Craig Gentry introduisit une nouvelle possibilité cryptographique, parfois nommée «Graal» par les enthousiastes. Cette dernière consiste à évaluer une fonction arbitraire sur les chiffrés. Autrement dit, appliquer un algorithme, avec un surcout, aux données chiffrées sans pour autant avoir accès aux textes clairs serait possible. Ainsi, une entité peut délocaliser un calcul chez un tiers potentiellement pernicieux en étant assurée que la confidentialité de ses données est préservée. Une application pragmatique serait de chiffrer les données à traiter par un service distant, tel que celui proposé par Amazon Web Service. Ce sont les cryptosystèmes offrant cette fonctionnalité, dits *chiffrements homomorphes* que nous allons étudier dans ce chapitre.

Nous verrons en premiers lieux les prémices de ces systèmes puis le premier cryptosystème utilisable dans la pratique malgré ses limitations. Puis nous verrons une branche plus performante dont nous étudierons la principale limite. Enfin, nous terminerons sur les systèmes qui proposent un compromis entre l'étendue des fonctions calculables sur les données chiffrées et la performance.

2.2 Prémices

2.2.1 Hypothèse

R. Rivest, L. Adelman et M. Dertouzos émirent en 1978 dans [47] l'hypothèse qu'un système de chiffrement nommé dans le texte «*Privacy homomorphism*» était crédible et utile. Dans cet article les auteurs partent d'un problème pratique puis proposent différentes solutions. La première solution consiste en une modification du matériel afin d'y introduire des registres sécurisés – inaccessibles pour le propriétaire de la machine – dans lesquels les données de l'utilisateur sont stockées déchiffrées afin d'y appliquer des opérations arithmétiques. Ce type de construction, avec matériel sécurisé, est nommé plus tard dans la littérature « *Trusted Hardware*». Cette catégorie de matériel peut être rattaché aux coprocesseurs sécurisés comme l'IBM 4758 utilisé dans [27] afin d'augmenter la confidentialité d'un PKI ainsi qu'aux cartes à puces (aussi équipé de nombreux systèmes de sécurité physiques). La seconde solution proposée est d'utiliser une construction cryptographique dont l'opération de chiffrement se comporte comme un (homo)morphisme de groupe entre la représentation des textes clairs et la représentation des chiffrés. C'est-à-dire que le morphisme (l'opération de chiffrement ou d'encodage dans l'article) préserve la structure de départ et la structure d'arrivée. À titre d'exemple, les logarithmes – népérien et décimal — sont des homomorphismes de groupe.

Nous allons présenter 3 définitions qui seront nécessaires à la bonne compréhension de ce chapitre :

Définition 4. *Un système de chiffrement se définit par trois fonctions,* $\mathrm{K}(\cdot)$ *la fonction de génération des clés,* $\mathrm{E}(.)$ *la fonction de chiffrement,* $\mathrm{D}(.)$ *la fonction de déchiffrement.*

En utilisant cette définition pour un système de chiffrement, on obtient :

Définition 5. *Soit un cryptosystème* $(\mathrm{K}(\cdot), \mathrm{E}(\cdot), \mathrm{D}(\cdot))$ *avec* E *allant de* G *vers* H. *Soit* $+_{\mathrm{G}}$ *une opération interne sur* G *telle que* $(\mathrm{G}, +_{\mathrm{G}})$ *soit un groupe, et* $+_{\mathrm{H}}$ *une opération interne sur* H. *Ce cryptosystème est dit (simplement) homomorphe lorsque pour tout couple de messages* (m_1, m_2) *avec* $m_1, m_2 \in \mathrm{G}$ *on a*

$$\mathrm{E}(m_1) +_{\mathrm{H}} \mathrm{E}(m_2) = \mathrm{E}(m_1 +_{\mathrm{G}} m_2).$$

Selon que l'opération $+_{\mathrm{G}}$ soit une multiplication ou une addition modulaire on dit que le cryptosystème est un chiffrement homomorphe multiplicatif ou additif. On ne s'intéresse que très rarement au cas ou $+_{\mathrm{G}}$ n'est dans aucun de ces deux cas. L'opération $+_{\mathrm{H}}$ a une importance pour le coût des opérations réalisées sur les données chiffrées, mais n'est pas utilisée en général pour catégoriser les systèmes de chiffrement homomorphe.

Définition 6. *Soit un cryptosystème* $(\mathrm{K}(\cdot), \mathrm{E}(\cdot), \mathrm{D}(\cdot))$ *avec* E *allant de* G *vers* H. *Soient* $+_{\mathrm{G}}, \cdot_{\mathrm{G}}$ *deux opérations internes sur* G *telles que* $(\mathrm{G}, +_{\mathrm{G}}, \cdot_{\mathrm{G}})$ *soit un anneau, et* $+_{\mathrm{H}}, \cdot_{\mathrm{H}}$ *deux opérations internes sur* H. *Ce cryptosystème est dit complètement homomorphe lorsque pour tout couple de messages* (m_1, m_2) *avec* $m_1, m_2 \in \mathrm{G}$ *on a :*

1. $\mathrm{E}(m_1) +_{\mathrm{H}} \mathrm{E}(m_2) = \mathrm{E}(m_1 +_{\mathrm{G}} m_2)$ *et*
2. $\mathrm{E}(m_1) \cdot_{\mathrm{H}} \mathrm{E}(m_2) = \mathrm{E}(m_1 \cdot_{\mathrm{G}} m_2)$.

Les résultats des opérations sur l'espace des chiffrés donnant des chiffrés il est possible d'utiliser ces opérations pour réaliser un nombre d'opérations arbitraires sur les chiffrés. Cette propriété n'est pas toujours facile à obtenir et ces définitions sont souvent relaxées vers une version dite en Anglais *leveled homomorphic* ou *somewhat homomorphic* où nous avons des propriétés du type

— $\mathrm{D}(\mathrm{E}(m_1) +_{\mathrm{H}} \mathrm{E}(m_2)) = m_1 +_{\mathrm{G}} m_2$ et
— $\mathrm{D}(\mathrm{E}(m_1) \cdot_{\mathrm{H}} \mathrm{E}(m_2)) = m_1 \cdot_{\mathrm{G}} m_2$.

Ces propriétés ne déterminent pas de validité du déchiffrement si plus d'une opération est réalisée sur les chiffrées et un paramètre supplémentaire L est généralement ajouté. Ce paramètre est la limite du nombre d'opérations que l'on peut réaliser (généralement mesurée en profondeur d'un circuit) tout en conservant un résultat valide au déchiffrement.

2.2.2 Chiffrements homomorphes multiplicatifs

Le système de chiffrement asymétrique RSA [48] est un exemple concret d'homomorphisme multiplicatif. Dans ce cas précis le produit de deux chiffrés est égal au chiffré du produit des clairs modulo un nombre.

Algorithm 10 Principaux algorithmes du cryptosystème RSA

Génération d'une paire de clés RSA

1. Choisir p et q deux nombres premiers distincts.
2. Calculer $n = p \cdot q$.
3. Choisir un entier e premier avec $\phi(n)$ – avec ϕ la fonction indicatrice d'Euler – et $e < \phi(n)$.
4. Calculer $d = e^{-1} \bmod \phi(n)$ avec $d < \phi(n)$.

Ainsi, le couple (e, n) est la clé publique du système et d est la clé privée.

Algorithme de chiffrement

Soit $m < n$ un texte clair, l'opération de chiffrement est : $c \equiv m^e \bmod n$.

Algorithme de déchiffrement

Soit c un chiffré, le texte clair correspondant est obtenu par l'opération de déchiffrement suivante : $m \equiv c^d \bmod n$.

Maintenant que nous avons décrit le système de chiffrement RSA par l'algorithme 10, nous allons décrire la propriété de chiffrement homomorphe. Soient c_1 et c_2 deux chiffrés des messages m_1 et m_2 obtenus par l'algorithme de chiffrement de RSA avec la clé publique (e, N). Si nous nous plaçons dans la position d'un tiers qui a accès au module n mais pas à la clé de déchiffrement d, nous pouvons calculer c_3 tel que $c_3 \equiv c_1 \cdot c_2 \pmod{n}$, puis le propriétaire de d déchiffre c_3 tel que :

$$\begin{aligned} \mathrm{D}(c_3) &\equiv \mathrm{D}(c_1 \cdot c_2) \equiv (c_1 \cdot c_2)^d \pmod{n} \\ \Leftrightarrow (c_1^d \cdot c_2^d) \equiv (m_1 \cdot m_2)^{ed} &\equiv (m_1 \cdot m_2)^{1+k\phi(n)} \equiv m_1 \cdot m_2 \pmod{n} \\ \text{donc } \mathrm{D}(c_3) &\equiv m_1 \cdot m_2 \pmod{n} \end{aligned}$$

Il s'agit donc d'un chiffrement homomorphe multiplicatif. Néanmoins, ce système de chiffrement a un problème de fond au niveau de la confidentialité. En effet, deux chiffrés du même message sont équivalents, il n'y a pas de données aléatoires où pseudo aléatoires introduites dans les chiffrés. La fonction de chiffrement est dite déterministe. Ainsi selon le protocole, le tiers qui exécute les opérations homomorphes peut inférer aisément le contenu de certains chiffrés par une attaque à clairs choisis. L'adversaire, connaissant certains messages échangés selon le protocole, les reproduit en utilisant la clé publique de chiffrement. Il peut distinguer au moins une partie des chiffrés les uns des autres [1].

De fait, les chiffrés produits par un cryptosystème homomorphe moderne, contrairement aux chiffrés produits par RSA, doivent être indistinguables.

Indistingabilité Une notion centrale de la cryptographie moderne est la notion d'indistingabilité calculatoire. Le principe sous-jacent est de savoir si des différences sont observables par l'application d'un algorithme calculable en temps raisonnable (en temps polynomial ou moins). Dans le cas où la réponse serait négative, on peut considérer que les deux objets sont indistingables pour toutes applications pratiques. [2]

1. Les protocoles PIR traités dans le Chap. 3 illustrent ce cas.
2. Traduction libre et augmentée de la définition C.12 de l'ouvrage [16].

Un second système de chiffrement encore utilisé aujourd'hui et possédant une propriété d'indistingabilité est le chiffrement ElGamal présenté en 1985 par Taher Elgamal [14]. Ce système, dérivé de l'échange de clés proposé par Diffie-Hellman [12], est fondé sur le problème du logarithme discret. Ce problème part du principe qu'il est facile de calculer une exponentiation modulaire, mais qu'une fois l'exponentiation calculée il n'existe pas d'algorithme capable de retrouver l'exposant en temps raisonnable.

Algorithm 11 Principaux algorithmes du cryptosystème ElGamal

Génération d'une paire de clés ElGamal

1. Choisir un nombre premier p et une racine primitive g dans $\mathcal{Z}_p$ d'ordre p (c'est-à-dire que les puissances de g modulo p engendrent $\mathbb{Z}_p - 0$).
2. Choisir aléatoirement un entier a dans l'intervalle $[1, .., p-1]$ et calculer $\alpha = (g^a \bmod p)$.

La partie publique de la clé est le triplet (p, g, α) et la partie privée est a.

Algorithme de chiffrement

Soit $m \in \mathbb{Z}_p$ un texte clair, l'opération de chiffrement se déroule comme suit :

1. Choisir aléatoirement un entier b dans l'intervalle $[1, \cdots, p-1]$ et calculer $\beta = (g^b \bmod p)$.
2. Calculer c tel que $c = \alpha^b \cdot m \bmod p$.

Le chiffré est le couple (β, c).

Algorithme de déchiffrement

Soit (β, c) un chiffré, le texte clair correspondant est obtenu par l'opération de déchiffrement suivante :

1. Déterminer le nombre $x = p - 1 - a$.
2. Calculer $\beta^x \cdot c \bmod p$ et retrouver le message m initial.

Montrons qu'il s'agit d'un chiffrement homomorphe multiplicatif. Soit (β_1, c_1) et (β_2, c_2) deux chiffrés des messages m_1 et m_2 obtenus par l'algorithme de chiffrement – voir Alg. 11 – À partir d'une clé publique quelconque (p, g, α). L'homomorphisme multiplicatif peut s'exprimer comme suit :

Chiffrement de m_1 et m_2 :

$$\beta_1 = m_1 \cdot \alpha^{b_1}, \qquad \beta_2 = m_2 \cdot \alpha^{b_2}$$
$$c_1 = g^{b_1}, \qquad c_2 = g^{b_2}$$

Multiplication de (β_1, c_1) et (β_2, c_2) :

$$\beta_3 = \beta_1 \cdot \beta_2 = (m_1 \alpha^{b_1}) \cdot (m_2 \alpha^{b_2}) = m_1 \cdot m_2 \cdot \alpha^{(b_1+b_2)}$$
$$c_3 = c_1 \cdot c_2 = g^{b_1} \cdot g^{b_2}$$

Déchiffrement de (β_3, c_3) :

$$\frac{\beta_3}{c_3^a} = \frac{m_1 \cdot m_2 \cdot \alpha^{(b_1+b_2)}}{g^{a(b_1+b_2)}} = \frac{m_1 \cdot m_2 \cdot g^{a(b_1+b_2)}}{g^{a(b_1+b_2)}} = m_1 \cdot m_2$$

Remarque.

Le cryptosystème ElGamal, en plus d'être homomorphe multiplicatif, produit des chiffrés indistinguables.

2.3 Chiffrement de Paillier et généralisation

2.3.1 Chiffrement de Paillier

Le système de chiffrement élaboré par P.Paillier dans [43] que l'on nommera Paillier dans la suite de ce mémoire, est fondé, à l'instar du cryptosystème ElGamal, sur le problème du logarithme discret. Il s'agit d'un chiffrement homomorphe additif, assurant l'indistingabilité des clairs. La communauté scientifique intéressée par le respect de la vie privée et la sécurité a trouvé de nombreuses applications à un cryptosystème offrant de telles possibilités.

2.3.2 Généralisation du cryptosystème

Damgård et Jurik présentent dans [10] une généralisation de Paillier. Dans ce nouveau cryptosystème, les chiffrés deviennent *surchiffrables*. Dans Paillier, pour chiffrer de nouveau des données préalablement chiffrées, ces dernières doivent être considérées comme le message clair à chiffrer. Ce message doit alors être segmenté, car un chiffré ne peut pas contenir l'intégralité d'un autre chiffré (la taille maximale des clairs, $\log n$, est plus petite que celle des chiffrés, $2 \log n$). Dans le cas de la généralisation, les chiffrés sont directement utilisables tels quels, c'est la taille du module avec lequel on travaille qui grandit. Si le module utilisé est n, on obtiendra successivement des chiffrés modulo $n^2, n^3, \ldots$. Le facteur d'expansion du au surchiffrement, c'est-à-dire l'accroissement de la taille des chiffrés, est réduit par rapport à Paillier.

2.3.3 Inconvénients majeurs

Le système de chiffrement est lent du fait de sa construction fondée sur le logarithme discret. Ainsi de nombreuses opérations et plus particulièrement la multiplication d'un chiffré par un entier, impliquent le calcul d'exponentiations modulaires avec de grands modules (2048 bits pour une sécurité de 80 bits). On notera que malgré l'existence d'algorithmes performants d'exponentiation modulaire, telle que l'exponentiation rapide (*square and multiply*), ces derniers ne réduisent pas suffisamment le coût de ces opérations. En effet, même si pour calculer x^n cet algorithme effectue $\mathrm{O}(\log n)$ multiplications à la place des $\mathrm{O}(n)$ de l'algorithme naïf, les multiplications se font dans le cas de Paillier, entre des entiers de grande taille pour laquelle il n'existe pas de registres suffisamment grands dans les processeurs d'ordinateurs actuels [49] pour effectuer ces multiplications en un seul tour d'horloge.

2.4 Chiffrement fondé sur les réseaux euclidiens

2.4.1 SIS et cryptosystèmes

Le premier cryptosystème notable fondé sur les réseaux idéaux, nommé NTRU, fut présenté en 1996 par Jeffrey Hoffstein, Jill Pipher et Joseph H. Silverman à Crypto'96 dans l'article [26] fondé sur le problème du vecteur le plus plus court (*shot vector problem* (SVP)) traité depuis les années 80 [32].

La communauté scientifique montra rapidement que le problème sur lequel était fondé ce cryptosystème peut être exprimé comme un problème fondé sur les réseaux euclidiens. Ainsi à la conférence ANTS de 1998 les auteurs de NTRU y présentèrent une version améliorée comprenant une évaluation approfondie de sa sécurité pratique contre les attaques utilisant les réseaux euclidiens.

Algorithm 12 Principaux algorithmes du cryptosystème Paillier

Génération d'une paire de clés Paillier

1. Choisir p et q deux premiers de même taille.
2. Calculer $n = pq$ et $\lambda = \phi(n)$
3. Calculer $g = n + 1$ un générateur de $\mathbb{Z}^*_{n^2}$
4. Calculer μ tel que $\mu = \lambda^{-1} \mod n$.

La partie publique de la clé et le couple (n, g) et la partie privée sont le couple (λ, μ).

Algorithme de chiffrement

Soit $m \in \mathbb{Z}_n$ un texte clair, l'opération de chiffrement est :

1. Choisir aléatoirement un entier r tel que $r \in \mathbb{Z}^*_n$
2. Calculer c tel quel $c = g^m \cdot r^n \mod n^2$

Le chiffré est l'entier c.

Algorithme de déchiffrement

Soit $c \in \mathbb{Z}^*_{n^2}$ un chiffré, le texte clair correspondant est obtenu par l'opération de déchiffrement suivante :

1. Calculer le clair $m = (c^\lambda \mod n^2) \cdot \mu \mod n$

Addition de deux clairs

Soient deux chiffré c_1 et c_2 des clairs m_1 et m_2 respectivement :

1. Calculer c_3 tel que $c_3 = c_1 \cdot c_2 \mod n^2$

On a $c_3 \equiv c_1 \cdot c_2 \equiv r_1^n \cdot r_2^n \cdot g^{m_1} \cdot g^{m_2} \equiv (r_1 \cdot r_2)^n \cdot g^{m_1+m_2} \mod n^2$ qui, une fois déchiffré donne $m_1 + m_2$.

Multiplication d'un chiffré par une constante

Soit c le chiffré de m et $k \in \mathbb{Z}_n$, la multiplication d'un chiffré par une constante se fait en effectuant le calcul suivant :

1. Calculer c' tel que $c' = c^k \mod n^2$.

On a $c' \equiv (r^n \cdot g^m)^k \equiv (r^k)^n \cdot g^{k \cdot m} \mod n^2$ qui, une fois déchiffré donne $k \cdot m$.

En 2011 Damiens Stehlé et Ron Steinfield proposèrent dans [54] une version performante de NTRUencrypt avec une sécurité fondée sur la difficulté de problèmes reconnus dans les réseaux euclidiens. Ce cryptosystème sera présenté dans la suite du document.

Les chiffrés sont représentés par des polynômes de degré n dont les coefficients se trouvent dans l'anneau commutatif $\mathbb{Z}_p$. L'anneau polynomial est quotienté par le polynôme $x^n + 1$ avec n une puissance de 2. Les éléments de ce système de chiffrement sont représentés par des polynômes de la forme :

$$\mathrm{P(X)} = a_0 + a_1\mathrm{X} + a_2\mathrm{X}^2 + \cdots + a_{n-1}\mathrm{X}^{n-1}, \text{ avec } a_i \text{ les coefficients du polynôme.}$$

NTRUencrypt est un cryptosystème à clé publique bâti autour de trois paramètres (n, p, q) avec n une puissance de 2 qui représente le degré maximal $n - 1$ des polynômes du système, et de deux modules p et q premiers entre eux et $q > p$. Ces principaux algorithmes sont présentés ci-dessous.

Algorithm 13 Principaux algorithmes du cryptosystème NTRU

Préliminaires

Soit $\mathrm{R}_q^\times = \mathbb{Z}_q[\mathrm{X}]/(\mathrm{X}^n - 1)$ un anneau de polynômes avec n une puissance de deux et q premier.

Génération de la paire de clés NTRU

1. Choisir deux premiers p et q avec $p < q$.
2. Générer aléatoirement les polynômes f et g de degré au plus $n-1$ avec des coefficients dans $\{-1, 0, 1\}$ tels que f soit inversible modulo p et modulo q.
3. Noter f_p, f_q les polynômes tels que $f \cdot f_p \equiv 1 \bmod p$ et $f \cdot f_q \equiv 1 \bmod q$.
4. Calculer h tel que $h = p \cdot f_q \cdot g \bmod q$.

La partie publique de la clé est le polynôme h et la partie privée est le couple de polynômes (f, f_p).

Algorithme de chiffrement

Soit $m \in \mathbb{Z}$ un texte clair, l'opération de chiffrement est :

1. Transformer m en polynôme M tel que les coefficients de M soient des entiers dans $[0, p-1]$.
2. Tirer aléatoirement un polynôme r avec de petites coefficients.
3. Calculer $c = r \cdot h + m \bmod q$.

Le chiffré est le polynôme $c \in \mathrm{R}_q^\times$.

Algorithme de déchiffrement

Soit c un chiffré le texte clair correspondant est obtenu par l'opération de déchiffrement suivante :

1. Calculer c' tel que $c' = f \cdot c$
2. Afin d'enlever les multiples de p calculer $\zeta = f \cdot m \bmod p = c' \bmod p$
3. Pour supprimer f il suffit maintenant de multiplier ζ par l'inverse de $f \bmod p$, c'est-à-dire f_p : $\mathrm{M} = \zeta \cdot f_p \bmod p$.

Remarque.

NTRUEncrypt offre une opération arithmétique sur les chiffrés, l'addition. Il est possible de l'étendre pour permettre aussi de faire des multiplications, mais nous ne présenterons pas ceci dans ce manuscrit.

2.4.2 LWE, Ring-LWE et cryptosystèmes

Parallèlement à la naissance et montée du cryptosystème NTRU, un problème fondé sur les réseaux euclidiens fut démocratisé par Regev. En 2005, il présenta le problème nommé aujourd'hui *Learning With Errors* (LWE) dans l'article [45]. Ce problème est suffisamment difficile et flexible pour en dériver des systèmes de chiffrement. Le problème «*learning with errors*» est une généralisation par Regev d'un problème nommé «*learning parity with noise*» à de plus grands modules. Ce problème est constitué de trois paramètres variables, $n \geq 1$, un module $q \geq 2$, et une distribution gaussienne aléatoire sur $\mathbb{Z}_q$. Regev a ensuite proposé un cryptosystème dans [46] fondé sur ce problème.

En 2010, Lyubashevsky et al. propose dans [37] une version de LWE fondée sur un anneau d'idéaux, intitulée Ring-LWE, lançant l'élaboration de cryptosystèmes plus naturels et performants que les précédents car il permet de manipuler directement des polynômes plutôt que des matrices. Brakerski et Vaikutanathan proposent, dans un articl publié dans la conférence CRYPTO'11, différents cryptosystèmes reposants sur ce problème [6]. C'est la version symétrique que nous allons présenter dans la suite de ce chapitre.

Dans ce cryptosystème, fondé sur le problème Ring-LWE, les différentes opérations font appel à des opérations arithmétiques faciles à transposer dans un langage informatique. Ce sont ces facilités d'implémentations qui nous ont poussés à programmer ce système cryptographique fondé sur les réseaux afin de construire un cPIR - voir Chap. 3 - performant.

Algorithm 14 Principaux algorithmes d'un cryptosystème fondé sur Ring-LWE

Génération de la clé secrète du cryptosystème

Entrée : un entier t module pour les coefficients dans l'espace des clairs

1. En suivant les paramètres de sécurité énoncés dans [35] déterminer un entier q qui servira de module, n une puissance de 2, et une distribution aléatoire χ.
2. Constuire un polynôme s dont les coefficients suivent χ.

La clé sécrète est le polynôme s.

Algorithme de chiffrement

Soit $m \in \mathbb{Z}$ un texte clair, l'opération de chiffrement est :

1. Transformer m en polynôme M tel que les coefficients de M soient des entiers dans $[\mathrm{O}, t-1]$.
2. Générer des polynômes a, e tel que leurs coefficients suivent χ.
3. Calculer le polynôme $e' = e \otimes t + \mathrm{M}$ avec $\otimes t$ représentant la multiplication de tous les coefficients par l'entier t.
4. Calculer $b = (a * s) + e'$.

Le chiffré est le couple de polynômes (a, b).

Algorithme de déchiffrement

Soit (a, b) un chiffré tel que $a, b \in \mathrm{R}_q$ et s la clé secrète, le clair M correspondant est obtenu par l'opération de déchiffrement suivante :

1. Calculer $e' \in \mathrm{R}_q$ par $e' = b - (a * s)$.
2. Calculer $\mathrm{M} \in \mathrm{R}_q$ par $\mathrm{M} = e' \mod t$.

Addition de deux chiffrés

Soit deux couples de chiffrés (a_1, b_1) et (a_2, b_2) de, respectivement, le message m_1 et le message m_2.

1. Calculer le couple (a_3, b_3) tel que $(a_3, b_3) = (a_1 + a_2, b_1 + b_2)$

On a $a_3 = a_1 + a_2$ et $b_3 = (a_1 + a_2)s + (e_1 + e_2)t + (\mathrm{M}_1 + \mathrm{M}_2)$ et pour le déchiffrement de (a_3, b_3) on calcule :
$e'_3 = (a_1 + a_2)s + (e_1 + e_2)t + (\mathrm{M}_1 + \mathrm{M}_2) - (a_1 + a_2)s$ puis $\mathrm{M}_3 = (e_1 + e_2)t + (\mathrm{M}_1 + \mathrm{M}_2) \mod t = (\mathrm{M}_1 + \mathrm{M}_2) \mod t$.

Multiplication d'un chiffré par une constante

Soit un chiffré (a, b) et un entier k, le résultat de la multiplication entre un chiffré et une constante est obtenu comme suit :

1. Calculer (a', b') tel que $(a', b') = (k \cdot a, k \cdot b$

On a $a' = k \cdot a$ et $b' = k \cdot (a * s) + k \cdot e \otimes t + k \cdot \mathrm{M}$ et pour le déchiffrement de (a', b') on calcule $e = k \cdot (a * s) + k \cdot e \otimes t + k \cdot \mathrm{M} - k \cdot (a * s) = k \cdot e \otimes t + k \cdot \mathrm{M}$ puis $\mathrm{M}' = k \cdot e \otimes t + k \cdot \mathrm{M} \mod t = k \cdot \mathrm{M} \mod t$.

2.5 Conclusion

Les principes du chiffrement homomorphe sont posés, comme nous l'avons montré, depuis de nombreuses années. Cette idée parait avoir été laissée de coté par la communauté même si des cryptosystèmes bien connus tel que RSA et ElGamal possédaient déjà des propriétés homomorphes sans avoir été construits dans cette optique. Nous nous sommes concentrés sur des systèmes plus récents, fondés sur le problème Ring-LWE, qui nous permettront d'obtenir des performances en adéquation avec des applications comme le retrait d'information privé.

Chapitre 3

Private Information Retrieval

Table des matières

3.1 Introduction 35
3.2 Fondements 36
3.3 Principe général 37
3.4 Kushilevitz et Ostrovsky 38
3.4.1 Protocole 38
3.4.2 Multi-bits 40
3.5 Construction J . Stern 40
3.6 Construction de H. Lipmaa 42
3.7 Gentry et Ramzan 43
3.8 J. Trostle et A. Parrish 45
3.8.1 cPIR fondé sur l'anonymat 45
3.8.2 cPIR avec fonction à sens unique 46
3.9 Synthèse du chapitre 47

Résumé : *Le retrait d'information privé ou en anglais Private Information Retrieval est un protocole capable de garder secret l'index d'un élément d'une base de données accédé par un utilisateur sur un serveur distant. Dans ce chapitre, nous faisons un état de l'art de ce protocole.*

3.1 Introduction

L'avènement du web interactif, nommé *web 2.0* par certains spécialistes comme Tim O'Reilly, ainsi que l'augmentation de la vitesse des connexions internet privées a fait croitre les possibilités des internautes. Le web est devenu une immense société de services parmi lesquels les internautes naviguent. Ce sont ces choix, d'un service plutôt qu'un autre, d'un site web plutôt qu'un autre, qui produisent les données qui seront ensuite employées par des régies publicitaires numériques telles que Google AdSense.

Certaines données personnelles, comme les photos, sont plus aisément dissimulables, mais ces choix finalement matérialisés par les clics de la souris sont particulièrement difficiles à garder privé.

C'est la proposition du PIR, acronyme de *Private Information Retrieval*, traduisible en français par récupération (ou téléchargement) d'informations privé : cacher, dissimuler un choix parmi plusieurs.

Autrement dit, le *Private Information Retrieval* est un protocole qui donne le moyen à ses utilisateurs d'obtenir un élément d'une base de données publique sans concéder la moindre information sur l'élément obtenu.

L'intérêt dudit protocole est justement le fait de garder secrets, même aux yeux de l'administrateur de la base de données, les éléments téléchargés par les utilisateurs. Par

exemple dans le cas d'une bibliothèque électronique, comme peut proposer Amazon, cet outil permet de garder secrets les centres d'intérêts des lecteurs. Dans le cadre d'un service de diffusion en flux comme YouTube, et même Netflix, l'application de ce protocole garantit le secret des choix des vidéos visionnées par les abonnées.

La méthode naïve pour résoudre ce problème est de télécharger l'intégralité des informations de la base de données. Ainsi l'utilisateur ne divulgue pas les éléments qui l'intéressent. Néanmoins, la limite de ce procédé est vite atteinte dans une application réaliste. En effet, télécharger des milliers de Mb est impensable avec le débit moyen que l'on peut trouver chez les potentiels utilisateurs. Selon le site web de test de débit `ariase.com`, le débit moyen en France est de 6.6 Mb/s en 2014 [1]. En supposant que l'on désire télécharger 8000 Gb à ce débit, la durée du téléchargement serait de 2 semaines. Un tel délai est impensable dans un monde numérique ou l'instantanéité de l'accès à l'information est la règle. En effet le démarrage d'une vidéo en flux [2] requiert, en général, quelques secondes d'attente [3]. De plus, ce procédé est inapplicable sur des bases de données qui changent très régulièrement, par exemple des indices d'une bourse d'échanges, car l'information obtenue serait alors obsolète.

Le PIR s'inscrit dans un ensemble de protocoles respectueux de la vie privée composé de l'*Oblivious Transfert* (OT), de l'Oblivious RAM (ORAM) et du PIR.

— L'OT, proposé par Michael O. Rabin dans [44], est un protocole dont la particularité principale est que les éléments de la base de données restent à la discrétion de l'administrateur de la base. De ce fait, lorsque l'utilisateur (1) obtient un élément de la base de données (2) il n'acquiert aucune information sur les autres éléments (3) l'administrateur n'a pas connaissance de l'élément obtenu par l'utilisateur, comme présenté dans [11].

— L'ORAM quant à lui, est un protocole de stockage de données externalisé qui garantit à l'utilisateur la confidentialité et des données placées chez un tiers et des opérations de lecture et d'écriture effectuées. Brièvement, les données de l'utilisateur placées chez le tiers sont chiffrées et, lorsque l'utilisateur écrit ou lit l'une ces données distantes les deux opérations, lecture et écriture, sont effectuées à chaque fois. Ainsi, il est impossible pour l'administrateur du serveur d'inférer des informations sur les données stockées que ce soit par les données chiffrées elles-mêmes ou par les opérations réalisées.

La proximité de ces protocoles peut amener à les confondre. Pour illustrer ce qui vient d'être dit, la différence fondamentale entre un PIR et un OT est la condition (2) présentée dans la définition de l'OT. Afin d'ôter toute ambiguïté, le tableau suivant récapitule ces trois protocoles.

Prot.	Propriétaire	Stockage	Anonymat
PIR	Publiques	Serveur	De l'élément récupéré
OT	Serveur	Serveur	De l'élément récupéré et des données serveur
ORAM	Utilisateur	Serveur	Des données et des opérations

Table 3.1 – Récapitulatif des principales propriétés des protocoles ORAM, OT et PIR

3.2 Fondements

Le PIR fut présenté en premier lieu par Chor, Goldreich, Kushilevitz et Sudan en 1995 [9]. Dans cet article fondamental, les auteurs proposent un ensemble de protocoles PIR

1. `http://www.ariase.com/fr/news/etude-internet-akamai-t4-2013-article-3259.html`
2. On trouvera plus souvent le terme anglais *streaming*.
3. Ce délai est un constat empirique.

qui utilisent des bases de données répliquées dont la sécurité est fondée sur la théorie de l'information. Ce type de PIR sera nommé plus tard information-theoretic PIR ou itPIR. Cette dénomination signifie que la requête de l'utilisateur ne donne pas, au sens de la théorie de l'information, d'information à propos de l'élément qu'il choisit. Ceci étant dit, la discrétion de ce choix n'est préservée que si les serveurs de réplication ne complotent pas tous contre l'utilisateur. Ainsi, si k est le nombre de serveurs de réplication, n le nombre d'éléments de la base de données et ℓ leur taille en bits. L'utilisateur va produire une requête de k vecteurs de n bits et la réponse sera composée de k éléments de ℓ bits. On voit qu'un tel protocole demande d'importantes ressources en communication, ce qui nous rapporte au problème du débit exposé dans l'introduction de ce chapitre.

Dans ce mémoire de thèse, nous nous concentrons sur les protocoles PIR qui ne nécessitent pas de réplication, ils sont appelés «*single-database PIR*». La sécurité du choix de l'utilisateur dépend ici de la limite de puissance de calculs de l'attaquant.

Le premier PIR sans réplication fut présenté par Kushilevitz et Ostrovsky en 1997 [31]. Il est fondé sur un cryptosystème à clé publique sécurisé. Une nomenclature supplémenataire accompagne ces protocoles, ils sont nommés «*computational* PIR» cPIR que l'on traduit par PIR calculatoire.

3.3 Principe général

Cette partie va présenter le principe général de la récupération d'informations privé, avant d'aborder les constructions phares de la littérature.

Comme vu dans l'introduction, on s'intéresse aux protocoles sans réplication. Ainsi on peut définir un PIR à base de données unique comme un jeu à deux joueurs : un utilisateur $\mathcal{U}$ et une base de données $\mathcal{DB}$. $\mathcal{U}$ souhaite obtenir un élément i de $\mathcal{DB}$ (comme le i-ème bit) sans révéler l'élément souhaité (l'indice i de l'élément reste caché). On décrit le protocole PIR comme l'enchainement des étapes suivantes :

1. $\mathcal{U}$ génère une requête à partir du générateur de requête Q(.) et de l'entrée i.
2. La requête est envoyée à $\mathcal{DB}$.
3. $\mathcal{DB}$ utilise un générateur de réponse R(.) pour combiner la requête avec les éléments de la base de données.
4. La réponse est envoyée à $\mathcal{U}$.
5. Enfin, $\mathcal{U}$ extrait i-ème élément à partir de son extracteur de réponse X(.).

Afin de visualiser les interactions des différents algorithmes en fonctions du temps, on représente le protocole PIR générique sur le diagramme de séquence 3.1.

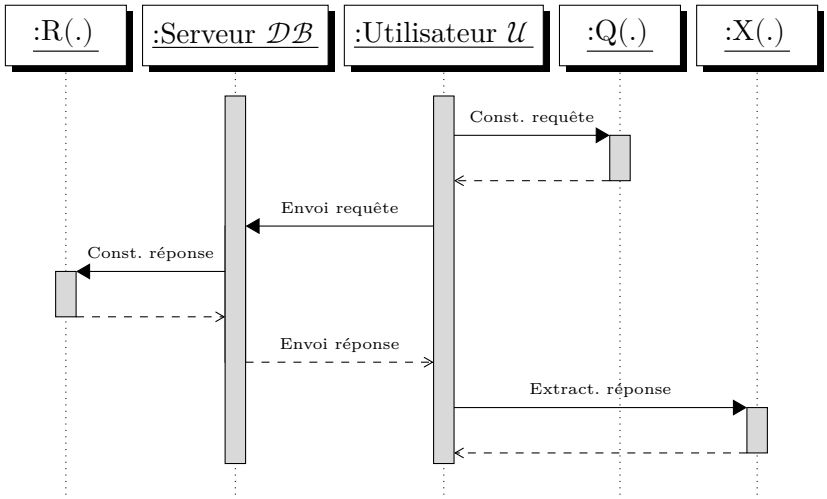


Figure 3.1 – Diagramme de séquence du protocole PIR générique. On voit que $\mathcal{U}$ s'occupe de générer la requête avec la fonction Q(.) et d'extraire la réponse avec la fonction X(.). Le serveur $\mathcal{DB}$ construit la réponse à partir de la requête en faisant appel à la fonction R(.).

3.4 Kushilevitz et Ostrovsky

3.4.1 Protocole

Comme décrit dans l'introduction Kushilevitz et Ostrovsky [31] proposèrent le premier PIR fondé sur la théorie des nombres (et non plus de l'information) afin d'utiliser une unique base de données.

Dans cette construction, l'utilisateur récupère 1 bit, à partir d'un cryptosystème qui s'appuie sur les propriétés des résidus quadratiques. En arithmétique modulaire, on dit qu'un entier naturel q est un résidu quadratique (respectivement QR pour résidu quadratique et QNR pour résidu non quadratique) modulo p s'il existe un entier x tel que :

$$x^2 \equiv q \,(\mathrm{mod} p)$$

Propriétés : relatives aux résidus quadratiques

1. Le produit de 2 QR donne 1 QR ;
2. Le produit de 1 QR et d'un QNR donne un QNR ;
3. Le produit de deux QNR donne un QNR.
4. Un QNR au carré donne 1 QR.

Cryptosystème : $\mathcal{U}$ va calculer un entier N de k bits. Il tire deux nombres premiers p_1 et p_2 de $k/2$ bits, puis il calcule $\mathrm{N} = p_1 \cdot p_2$. Lorsque l'on connait la factorisation de N, on peut aisément vérifier (en $\mathrm{O}(\log^3 \mathrm{N})$) si un nombre entier est un résidu quadratique.

Dans ce protocole, la base de données est composée de n bits. Elle représentée sous la forme d'une matrice de $s \times t = n$ bits, nommée M. L'utilisateur veut transférer un bit en gardant secret ses coordonnées dans la matrice. Le bit x correspond à l'entrée (a, b) de la matrice M. Le protocole de Kushilevitz et Ostrovsky se déroule comme suit :

Génération de la requête

1. $\mathcal{U}$ tire deux premiers p_1 et p_2 de $k/2$ bits puis construit N, enfin $\mathcal{U}$ envoie N à $\mathcal{DB}$.
2. $\mathcal{U}$ tire uniformément t entiers y tel que $y_1, \cdots, y_t \in \mathbb{Z}_{\mathrm{N}}$. Avec y_b, pour b l'indice de la colonne ou se situe x dans M, un QNR et y_j un QR pour $j \neq b$. Enfin $\mathcal{U}$ envoie ces t nombres à $\mathcal{DB}$ pour un total de $t \cdot k$ bits.

Génération de la réponse

1. $\mathcal{DB}$ calcule pour chaque élément j de la requête et chaque bit $\mathrm{M}_{r,j}$ de la matrice :
$$\mathrm{W}_{r,j} = \begin{cases} y_j^2 & si \quad \mathrm{M}_{r,j} = 0 \\ y_j & si \quad \mathrm{M}_{r,j} = 1 \end{cases}$$
puis il calcule pour chaque rangée r un nombre $z_r \in \mathbb{Z}_{\mathrm{N}}$ tel que :
$$z_r = \prod_{j=1}^{t} \mathrm{W}_{r,j}$$
2. $\mathcal{DB}$ envoie $z_1, \cdots, z_s$ (matrice de $s \times t$) à $\mathcal{U}$ pour un total de $s \cdot k$ bits.

Récupération de la réponse

1. $\mathcal{U}$ se concentre sur z_a car cette valeur correspond à la ligne de M qui contient le bit x_i. Comme $\mathcal{U}$ connait les facteurs de N alors il peut vérifier la forme de z_a. Si c'est un QR alors $\mathrm{M}_{a,b} = 0$, et si c'est un QNR alors $\mathrm{M}_{a,b} = 1$.

Justesse : Dans la phase de génération de la réponse du protocole lorsque $j \neq b$ alors $\mathrm{W}_{r,j}$ est toujours un QR d'après les propriétés 1 et 3 de résidus quadratiques. Lorsque $j = b$ alors $\mathrm{W}_{r,b}$ est un QR si et seulement si $\mathrm{M}_{r,b} = 0$ par la propriété 2 (QNR sinon). Suite au produit des $\mathrm{W}_{r,j}$, z_r est un QR si et seulement si $\mathrm{M}_{r,b} = 0$ (QNR sinon), car les autres $\mathrm{W}_{r,j}$ sont aussi des QR. Ainsi dans la phase 5, z_a est QR *ssi* $\mathrm{M}_{a,b} = 0$ (QNR sinon). En conclusion z_a est QR si $x_i = 0$ sinon c'est un QNR, $\mathcal{U}$ obtient bien le bit x_i.

Exemple : Soit x la base de données composée de $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$ avec $s = t = 2$ la matrice M prend la forme suivante :

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

On admet que $\mathcal{U}$ veut l'élément $x_4 = 1$ de coordonnées $(2, 2)$. Il tire deux premiers p_1 et p_2 afin de calculer N, puis il génère la requête $q = (y_1, y_2)$ avec y_1 QR et y_2 QNR, car x_4.est dans la seconde colonne. Afin de simplifier l'écriture, on écrira $q = (\mathrm{QR}, \mathrm{QNR})$. Puis $\mathcal{U}$ envoie la requête q à $\mathcal{DB}$ qui va suivre l'étape 3 du protocole décrit précédemment. $\mathcal{DB}$ calcule les $\mathrm{W}_{r,j}$ tel que, en restant sous l'écriture simplifiée, $(\mathrm{W}_{1,1}, \mathrm{W}_{1,2}) = (\mathrm{QR}, \mathrm{QNR})$ et $(\mathrm{W}_{2,1}, \mathrm{W}_{2,2}) = (\mathrm{QNR}, \mathrm{QNR})$. À noter que le serveur ne peut pas déterminer la forme des $\mathrm{W}_{r,j}$ car la factorisation de N reste secrète. Puis il calcule $z_1 = \mathrm{QR} \times \mathrm{QNR} = \mathrm{QNR}$ et $z_2 = \mathrm{QR} \times \mathrm{QNR} = \mathrm{QNR}$ et envoie ces deux valeurs à $\mathcal{U}$. Ce dernier va seulement regarder z_2 car c'est la la ligne dans laquelle se trouve x_4. Comme ce dernier est un QNR il apprend que le bit x_4 de coordonnée $(2, 2)$ dans M est 1.

Défauts : Cette construction possède deux défauts majeurs. Le premier, vient de la taille de la requête qui est seulement intéressante si $s > k$. De plus, le facteur d'expansion de

la réponse est très élevé, car pour 1 bit, la réponse est de $s \cdot k$ bits. Dans la suite de leur article Kushilevitz et Ostrovsky, proposent des optimisations, qui seront utilisées dans de nombreuses autres constructions postérieures, afin d'améliorer ces différentes bornes.

3.4.2 Multi-bits

Le protocole peut être adapté simplement afin de récupérer des éléments de ℓ bits. En effet, supposons que l'on ait une base de données de n éléments de ℓ bits. On peut transformer cette unique base de données en ℓ bases de données d'éléments de 1 bit. Dans cette configuration, pour transférer les ℓ bits de l'élément (a, b), la requête est la même que pour transférer 1 seul bit. Le serveur l'applique à chaque « sous » base de données de 1 bit et envoie $\ell \cdot n \cdot t$ bits au client.

3.5 Construction J . Stern

Julien Stern propose dans [55] le premier PIR calculatoire qui utilise un cryptosystème homomorphe avec un espace de clairs de grande taille. Dans cet article, le protocole est classé comme un *All-Or-Nothing Disclosure of Secret*. Il exprime le problème comme suit : un acheteur (l'utilisateur $\mathcal{U}$) veut acquérir l'un des secrets du vendeur (la base de données $\mathcal{DB}$). Apparait ici une notion de secret, en effet le vendeur détient une liste de secrets, dans cette conformation la base de données n'est plus publique. Ce protocole est plus proche d'un OT que d'un PIR. Cependant, les contributions proposées sont applicables et ont été appliquées au PIR c'est pourquoi il nous semble important de l'expliciter.

Nous supposons que le système de chiffrement utilisé dispose d'une opération sur l'espace des chiffrés (dans l'article original une multiplication modulaire) permettant de réaliser une somme sur les clairs, et d'une deuxième opération sur l'espace des chiffrés (dans l'article original une exponentiation modulaire) permettant de réaliser un produit scalaire entre un entier et un clair. Nous notons ces opérations multiplicativement, mais rien n'interdit d'utiliser un système de chiffrement ou les opérations sur l'espace des chiffrés sont autres (comme par exemple pour le système de chiffrement basé sur Ring-LWE du chapitre précédent).

Justesse : La requête produite par $\mathcal{U}$ est composée de $n-1$ chiffrés de 0 et d'un unique chiffré de 1, d'indice j. L'index du chiffré de 1 est l'index de l'élément que $\mathcal{U}$ veut téléchargé. Soit R la requête, on peut la représenter : comme suit $\mathrm{R} = (\mathrm{E}(0)_1, \mathrm{E}(0)_2, \cdots, \mathrm{E}(1)_j, \cdots, \mathrm{E}(0)_n)$. Comme la séquence Q est chiffrée $\mathcal{DB}$ ne peut pas tirer d'informations des différents éléments de la requête. Lors du calcul de S, $\mathcal{DB}$ utilise les propriétés 2 et 3 du système de chiffrement. Lors du calcul de $q_i^{s_i}$, il multiplie une constante, c'est-à-dire le secret s_i par le clair associé à q_i. Comme Q est composé de chiffrés de 0 et d'un chiffré de 1 on a les deux cas suivants :

1. $s_i \cdot \mathrm{E}(0) = \mathrm{E}(0)$ pour $n-1$ des s_i
2. $s_j \cdot \mathrm{E}(1) = \mathrm{E}(s_j)$ pour s_j, c'est-à-dire l'élément choisi par $\mathcal{U}$

Si l'on effectue uniquement l'ensemble des exponentiations $q_i^{s_i}$, on récupère une séquence de la forme suivante : $(\mathrm{E}(0)_1, \mathrm{E}(0)_2, \cdots, \mathrm{E}(s_j)_j, \cdots, \mathrm{E}(0)_n)$. Autrement dit, $n-1$ chiffrés de 0 et un chiffré du secret s_j. Lorsque l'on fait le produit de ces chiffrés, on fait appel à la première propriété du système de chiffrement. Ce qui donne :

$$(\mathrm{E}(0)_1 \cdot \mathrm{E}(0)_2 \cdot \cdots \cdot \mathrm{E}(s_j)_j \cdot \cdots \cdot \mathrm{E}(0)_n = \mathrm{E}(0 + 0 + \cdots + s_j + \cdots + 0) = \mathrm{E}(s_j) = \mathrm{R}$$

Puis $\mathcal{DB}$ envoie R à $\mathcal{U}$ qui calcule :

$$\mathrm{D}(\mathrm{R}) = \mathrm{D}(\mathrm{E}(s_j)) = s_j$$

$\mathcal{U}$ a bien téléchargé l'élément d'indice j souhaité tout en le gardant secret. Et, dans le cas de ce protocole, l'élément s_j récupéré.

Protocole : Soient n le nombre de secrets et $s_1, \cdots, s_n$ les secrets. Le protocole de Julien Stern se déroule comme suit :

Génération de la requête

1. $\mathcal{U}$ (Ou l'acheteur dans l'article original) initialise un cryptosystème, envoie ses paramètres publics à $\mathcal{DB}$ puis prouve leur validité.
2. $\mathcal{U}$ Chiffre une séquence de n nombres et l'envoi à $\mathcal{DB}$. C'est la fonction Q(.) de notre terminologie. Puis, il donne une preuve à divulgation nulle de connaissance que la séquence, que l'on peut nommer requête, est composée de chiffrés $t-1$ zéro et d'un unique 1. On appelle l'ensemble de la séquence Q avec $\mathrm{Q} = (q_1, \cdots, q_n)$

Génération de la réponse

1. $\mathcal{DB}$ reçoit la requête puis calcule (il utilise son générateur de réponse R(.)) :

$$\mathrm{S} = \prod_{i=1}^{n} q_i^{s_i}$$

2. $\mathcal{DB}$ transfert la réponse S à $\mathcal{U}$

Récupération de la réponse

1. $\mathcal{U}$ déchiffre S et retrouve le s_j en fonction du choix effectué lors de la création de la requête. Il a utilisé l'extracteur de réponse X(.) .

Découpage : Un problème survient lorsque la taille d'au moins l'un des éléments s ne peut pas être entièrement absorbé par un élément de la requête ou, autrement dit, lorsque la taille des éléments secrets est trop importante pour « tenir » dans un unique chiffré. Pour résoudre ce problème, J. Stern propose de découper les secrets en portions de taille adéquat puis d'appliquer la requête sans la modifier aux différentes portions obtenues.

On découpe le secret s_i en m portions puis on applique q_i à chacune des m portions de s_i. Ensuite, on somme les m bloques des s_i indépendamment et on obtient une réponse R composée, elle aussi, de m portions différentes.

Ainsi la taille de la requête est inchangée et la taille de la réponse est multipliée par la taille du secret - de l'élément de la base de données - le plus grand divisé par la taille maximum d'absorption d'un chiffré.

Récursivité : Dans ce protocole, $\mathcal{U}$ envoi n chiffrés au serveur afin de télécharger le secret qu'il désire. J. Stern montre qu'il est possible de réduire le nombre d'éléments de requête à $\log n$.

Comme précédemment, soient n le nombre de secrets et $s_1, \cdots, s_n$ les secrets eux-mêmes. Soit h un entier, on va séparer les n secrets en n/h groupes de h secrets. De plus h est connu des deux participants, ils savent tous deux comment sont répartis les secrets dans les groupes.

Puis, on applique le protocole original à quelques détails près. $\mathcal{U}$ va produire une requête de h éléments et cette requête est appliquée indépendamment sur les n/h groupes de secrets. Il en résulte n/h éléments chiffrés qui forme une nouvelle base de données. Cette dernière peut aussi être découpée en groupe plus petit et ainsi de suite. $\mathcal{U}$ produits des requêtes jusqu'à obtenir un unique élément, la réponse R, qui lui sera transmis par $\mathcal{DB}$.

Remarque.

À noter que pour chaque récursion la taille des secrets, qui sont alors chiffrés, grandit. Ce qui ne pose pas directement de problème, car on peut alors réaliser un découpage. Ainsi il faut choisir la taille de h et le nombre de récursions en tenant compte de la taille de la requête, mais aussi de la réponse. Car une réponse plus grande est plus longue à envoyer, mais aussi à déchiffrer.

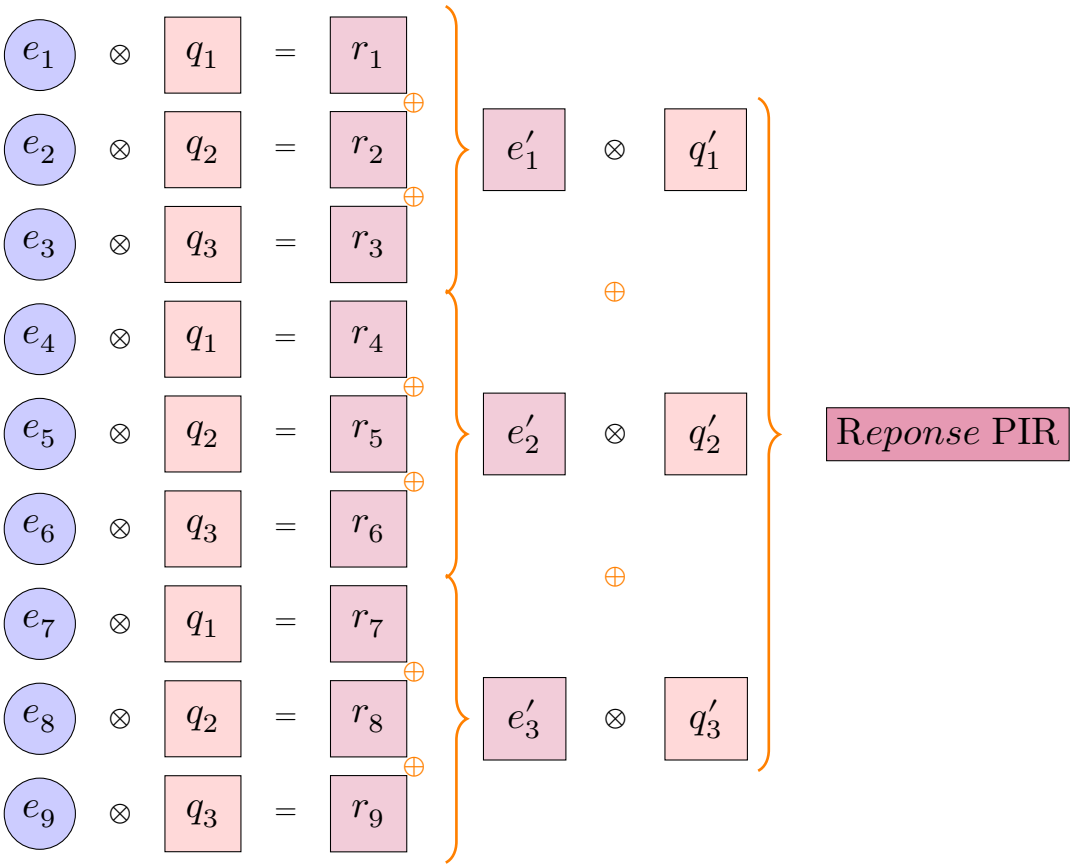


Figure 3.2 – Représentation de la génération de la réponse pour 1 niveau de récursivité. La base de données est composée de 9 éléments $(e_1, \cdots, e_9)$ séparés en groupes de 3 : (e_1, e_2, e_3), (e_4, e_5, e_6) et (e_7, e_8, e_9). La première partie de la requête appliquée à ces groupes est composée de trois éléments (q_1, q_2, q_3). Un fois les opérations multiplicatives et additives réalisées, on obtient une nouvelle base de donnée composée de 3 éléments (e'_1, e'_2, e'_3) à laquelle est appliquée une nouvelle requête (q'_1, q'_2, q'_3) afin d'obtenir la réponse finale.

3.6 Construction de H. Lipmaa

H. Lipmaa publie un article en 2004 [36] qui est une adaptation du protocole de Stern avec un système de chiffrement bien plus favorable à la taille de la réponse lorsque la récursion est utilisée. En effet, le système de chiffrement de Damgård et Jurik [10], qui est une généralisation du système proposé par Paillier [42], permet d'avoir une croissance plus lente (linéaire) de la taille de la réponse.

Le système de chiffrement utilisé fait grandir la taille les chiffrés linéairement lors des récursions et des éléments de la requête correspondants. Ainsi, lorsque d niveaux de récursion sont utilisés, la taille de la réponse est un élément chiffré d fois sans avoir été découpé à chaque récursion comme avancé par J. Stern. Pour $d = 2$, cette construction fait passer le facteur d'expansion du PIR de J. Stern avec le cryptosystème de Paillier de 4 à 3.

Cette amélioration de la taille des requêtes et de la réponse relance l'intérêt porté

au PIR comme le montreront les nombreuses contributions qui suivirent. Cependant, les opérations du système de chiffrements sont particulièrement lentes. En effet, l'opération de multiplication d'un chiffré par une constante revient à réaliser une exponentiation modulaire avec des modules de taille importante (2048 bits pour une sécurité de 80 bits). Lorsque plusieurs niveaux de récursion sont appliqués ces calculs deviennent extrêmement couteux sur un processeur d'ordinateur ordinaire.

3.7 Gentry et Ramzan

Craig Gentry et Zulfikan Ramzan dans [15] proposent une approche très différente à celle de J. Stern en conservant la propriété de PIR capable de récupérer un élément de plus d'un seul bit. Cette construction permet de réduire les coûts calculatoires et de communication mais demande en échange une optimisation hors ligne, c'est-à-dire en dehors de la phase d'échange entre le client et le serveur. Ce protocole utilise des entiers dits friables, dont la particularité est d'être le produit de *petits* nombres premiers.

Éléments spécifiques du protocole : Soient B une base de données de n-bits, et $f_1(x, y)$ et $f_2(x, y)$ deux fonctions. On détermine deux paramètres de sécurité k et k' tel que $k' = \Theta(\log n)$ et $k = f_2(k', \log n)$. Soient $\ell = \lfloor f_1(k, \log n) \rfloor$ et $t = \lceil \frac{n}{\ell} \rceil$. Ainsi qu' un ensemble $\mathrm{P} = p_1, \cdots, p_t$ de *petits* premiers et soit $\pi_i = p_i^{c_i}$ avec $c_i = \left\lceil \frac{\ell}{\log_2 p_i} \right\rceil$ et $\mathcal{S} = \{\pi_i\}$. Soit $\mathcal{G}_i$ l'ensemble des groupes cycliques dont l'ordre appartient à $[2^k, 2^{k+1}]$ et est divisible par π_i. Précisons qu'un groupe cyclique est un groupe fini dont tous les éléments peuvent être générés par un singleton que l'on appelle aussi générateur du groupe. Pour terminer, soit D_i une distribution aléatoire avec laquelle les éléments de $\mathcal{G}_j$ sont efficacement échantillonnés.

Remarque.

Ce protocole apporte de nombreuses améliorations vis-à-vis du protocole original (appelé CMS) sur lequel il est fondé [8] comme la possibilité de récupérer un bloc complet de bits et non un seul bit ainsi que, pour une sécurité comparable, une réponse de plus petite taille. Contrairement à ce qui est indiqué dans l'article l'opération la plus prohibitive pour le serveur n'est pas le calcul de $g^e \bmod m$ en $\mathcal{O}(n)$, mais celui du calcul de e qui oblige à utiliser l'algorithme de relèvement du théorème des restes chinois en $\mathcal{O}(n^2)$ (traité à la section 1.6 du chapitre 1).

Génération de la requête

À partir des éléments $(n, f_1, f_2, \mathcal{S}, \{\mathrm{D}_i\})$, l'utilisateur $\mathcal{U}$ identifie l'index i du bloc qu'il souhaite télécharger et génère une requête afin de récupérer le bloque C_i. $\mathcal{U}$ fait les opérations suivantes :

1. Générer $\mathrm{G} \xleftarrow{\mathrm{D}_i} \mathcal{G}_i$ et un « quasi générateur » g de G tiré d'une distribution uniforme. Autrement dit, g est un élément de G tel que le $\mathrm{PGCD}(|\mathrm{G} : \langle g \rangle|, \prod_{j=1}^{t} p_j) = 1$. Avec $\langle g \rangle$ un sous-groupe propre de G.
2. Envoyer la requête composée de (G, g), conserver $q = |\langle g \rangle|/\pi_i$ secret et précalculer $h = g^q$.

Réponse de la base de données

À partir des éléments $(\mathrm{B}, f_1, f_2, \mathcal{S})$, le serveur $\mathcal{DB}$ génère une réponse à la requête de $\mathcal{U}$ comme suit :

1. Exprimer chacun des ℓ-bits des blocs C_j comme un entier inclus dans $[0, 2^\ell - 1]$. (Le dernier bloc peut être complété avec des 0 ci nécessaire).
2. Calculer e le plus petit entier positif tel que (CRT) :

$$\begin{cases} e & \equiv & \mathrm{C}_1 \bmod \pi_1 \\ e & \equiv & \mathrm{C}_2 \bmod \pi_3 \\ & \vdots & \\ e & \equiv & \mathrm{C}_k \bmod \pi_k \end{cases}$$

3. Envoyer la réponse $g_e = g^e \in \mathrm{G}$.

Récupération de la réponse

À partir des données $(\pi_i, g_e, \mathrm{G}, q, h)$, $\mathcal{U}$ détermine le bloc C_i comme suit :

1. Calculer $h_e = g_e^q$.
2. Calculer C_i comme le logarithme discret $\log_h h_e$ dans le sous-groupe $\mathrm{H} \subset \mathrm{G}$ d'ordre $pi_i = p_i^{c_i}$ en utilisant l'algorithme de Pohlig-Hellman [4] dont la complexité dépend de la base du log à résoudre. Ainsi dans ce protocole, il faut privilégier des p_i de petite taille afin de conserver de bonnes performances d'extraction de la réponse.

3.8 J. Trostle et A. Parrish

Jonathan Trostle et Andy Parrish proposent en 2011 dans [57] deux protocoles PIR qui ont la particularité de ne pas être directement fondés sur un cryptosystème connu. Dans le premier, nommé *Anonymity Based cPIR Procol*, les utilisateurs vont séparer leurs requêtes et les mélanger ensemble afin de conserver leur anonymat. Le second, nommé *Trapdoor Group cPIR Protocol*, est dérivé du premier mais repose sur un système de chiffrement.

3.8.1 cPIR fondé sur l'anonymat

Dans cette version du protocole PIR proposé la base de données est ensemble de n bits indépendants et représentés sous la forme d'un carré de $\sqrt{n} \times \sqrt{n}$. L'utilisateur va sélectionner avec sa requête l'une des lignes de la base de données. Le protocole n'étant pas fondé sur un cryptosystème les utilisateurs vont mixer leurs requêtes formant ainsi une méta requête. Ainsi, la sécurité du protocole se situe dans l'hétérogénéité des éléments de la méta requête qui sera appliquée sur la base de données par le serveur.

Génération de la requête

1. $\mathcal{U}$ construit un vecteur B de $\sqrt{n}$ bits avec un 1 pour la ligne qu'il souhaite récupérer et un 0 pour les autres.
2. $\mathcal{U}$ construit un vecteur U vecteur de $\sqrt{n}$ bits dont les valeurs sont tirées suivant une distribution aléatoire uniforme puis calcule R tel que $\mathrm{R} = \mathrm{B} + \mathrm{U} \bmod 2$
3. $\mathcal{U}$ construit w vecteurs D dont les valeurs sont tirées suivant une distribution uniforme en respectant la contrainte suivante : $\sum_{i=1}^{w} \mathrm{D}_i \bmod 2 = \mathrm{U}$. $\mathcal{U}$ envoie ces w vecteurs ainsi que R.

Génération de la réponse

1. $\mathcal{DB}$ réaliser une multiplication matricielle chacun des vecteurs (sous forme de ligne) par la base de données que l'on peut voir comme une matrice carrée. $\mathcal{DB}$ envoie les $w + 1$ vecteurs résultants de ces multiplications au client.

Récupération de la réponse

1. $\mathcal{U}$ fait la somme des vecteurs (mod2) et efface le bruit. Ainsi, il récupère la ligne de la base de données sélectionnée au début du protocole.

3.8.2 cPIR avec fonction à sens unique

Dans ce protocole, la base de données est représentée comme une matrice de $\sqrt{n} \times \sqrt{n}$ éléments d_{ij} avec i la colonne et j la ligne. De plus, les éléments ne sont plus des bits uniques, mais peuvent être des blocs de N bits de taille arbitraire. La confidentialité de l'index est garantie par une fonction à sens unique.

Génération de la requête

1. $\mathcal{U}$ détermine un grand entier secret m qui dépend du nombre d'éléments n et du nombre de lignes r qu'il veut récupérer.
2. $\mathcal{U}$ choisi un secret b tel que $b \in \mathbb{Z}_m*$ tel que le PGCD$(b, m) = 1$ ainsi que $t = \sqrt{n}$ entiers secrets e_i tel que $e_i < m/t(\mathrm{N}-1)$. Le e_i pour les lignes désirées (celles que $\mathcal{U}$ récupèrera à la fin du protocole) sont de la forme $\mathrm{N}^l + a_l\mathrm{N}^r$ pour un $\ell < r$ et a_ℓ choisis et sont des multiples de N^r pour les autres.
3. $\mathcal{U}$ envoie $q_i = be_i$ pour i de 1 à t à $\mathcal{DB}$.

Génération de la réponse

1. Pour chaque ligne de la base de donnée ; $\mathcal{DB}$ calcule

$$s_j = \sum_{i=0}^{\sqrt{n}} q_i \cdot d_{ij}$$

   avec j de 0 à $\sqrt{n}$. $\mathcal{DB}$ obtient de la sorte les s_j formant la réponse $\mathcal{S}$.
2. $\mathcal{DB}$ envoie $\mathcal{S}$ à $\mathcal{U}$.

Récupération de la réponse

1. $\mathcal{U}$ multiplie tous les éléments de $\mathcal{S}$ par b^{-1} mod m et récupère l'élément désiré en base N de ces quotients. Autrement exprimé, $\mathcal{U}$ trouve des entiers en base 10 modulo m et pour déterminer les lignes qu'il désirait récupérer il doit calculer le reste de ces entiers modulo N. Ainsi, le reste du premier élément de réponse et le premier élément de la colonne désirée, le second le second élément et ansi de suite.

3.8.2.1 Exemple

Voici une base de données de $n = 9$ élément de 1 bit (N = 2), représentée sous la forme d'un carré de 3×3 :

$$\mathrm{DB} : \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$\mathcal{U}$ souhaite récupérer la seconde colonne, $r = 2$.

Requête : $\mathcal{U}$ choisit le module $m = 27$ et $b = 7$ (ces entiers restent secret). Puis il va générer les e_i tel que $e_i < 27/2$. De plus, comme $\mathcal{U}$ cherche à obtenir la seconde colonne ($r = 2$), e_1 et e_3 doivent être pairs et e_2 impair. On a donc : $e_1 = 4, e_2 = 5$ et $e_3 = 2$. $\mathcal{U}$ va ensuite calculer les $b \cdot e_i$ mod m ainsi il obtient comme élément de requête, les entiers $\{1, 8, 14\}$.

Réponse : $\mathcal{DB}$ reçoit le vecteur de requête de calcul le produit matriciel entre le vecteur et la base de données sous forme de matrice carrée. Il obtient le vecteur $\{15, 9, 22\}$ à envoyer à $\mathcal{U}$.

Extraction : $\mathcal{U}$ calcule d'abord b^{-1} mod m c'est à dire $b^{-1} \equiv 4 \bmod 27$, multiplie le vecteur de réponse par 4 mod 27 et obtient les trois entiers suivants : $\{6, 9, 7\}$. $\mathcal{U}$ calcule ensuite le reste modulo 2 de chacun de ces entiers et obtient $(0, 1, 1)$.

3.9 Synthèse du chapitre

En 2007, Carlos Aguilar-Melchor et Philippe Gaborit dans [38] proposent un protocole de PIR rapide fondé sur un cryptosystème non conventionnel utilisant les réseaux euclidiens. Cependant, le cryptosystème fut cassé en 2012 par Jingguo Bi, Mingjie Liu et Xiaoyun Wang [30].

Ces travaux, malgré le défaut de sécurité, ont montré une nette amélioration dans la durée de génération de la réponse. En effet, dans [38], récupérer un fichier de la taille d'un morceau de musique encodé en mp3 (quelques mégaoctets), dans une bibliothèque de plusieurs milliers de fichiers similaires est de 10 minutes. Avec la même configuration, le protocole de Lipmaa nécessite 33h, celui de Gentry et Ramzan 17 min et celui de Trosle et Parrish 13 min.

Nous avons montré dans ce chapitre que le protocole PIR a fortement évolué ces dernières années au point de se révéler être un protocole crédible dans des cas réalistes.

Les différentes contributions présentées, mise à part celle de J. Stern, ont la particularité d'être fortement dépendantes de leur cryptosystème. Ce qui permet de connaitre à l'avance la complexité des différents algorithmes utilisés ainsi que le facteur d'expansion. C'est-à-dire le nombre de bits reçus par $\mathcal{U}$ par rapport aux nombre de bits de l'élément choisi. Mais un invariant fondamental du PIR apparait, lors de la génération de la réponse par la fonction R(.) tous les bits de la base de données sont traités par l'algorithme.

Notre objectif est d'essayer de réduire le coût calculatoire de génération de la réponse par bit dans la base de données, car celui-ci est dans la plupart des cas le facteur limitant la vitesse que l'on peut atteindre dans un protocole de téléchargement privé.

Chapitre 4

Bibliothèque NFL

Table des matières

4.1 Introduction 49
4.2 NFLlib : Présentation 50
4.2.1 Représentation CRT à taille fixe 51
4.2.2 Optimisation de la multiplication modulaire 52
4.2.3 Un algorithme de NTT paresseux 53
4.2.4 Génération de bruit uniforme et gaussien 53
4.3 Évaluation des performances et comparaison avec NTL, FLINT et HElib . 54
4.4 Cas d'utilisation de NFLlib 56
4.4.1 Échange performant de clés 56
4.4.2 Chiffrement homomorphe avec NFLlib 58
4.5 Conclusion 59

Résumé : *Nous avons vu dans les chapitres précédents que les systèmes de chiffrement homomorphes sont fondamentaux dans les protocoles de retrait d'information privé calculatoires. Par leurs performances exceptionnelles, les cryptosystèmes fondés sur les réseaux euclidiens semblent être un choix adéquat pour ce type d'application. Dans ce chapitre, nous présentons une bibliothèque de manipulation de polynômes utilisés dans les réseaux euclidiens d'idéaux.*

4.1 Introduction

Dans ce chapitre, nous présentons NFLlib pour *NTT-based Fast Lattice Library*, une bibliothèque écrite dans le langage de programmation orientée objet `C++` spécialisée dans la manipulation de polynômes appartenant à des anneaux de la forme $\mathbb{Z}_p[x]/(x^n+1)$(voir le chapitre 1). L'objectif principal de cette bibliothèque est de faciliter la mise en œuvre de cryptosystèmes fondés sur les réseaux euclidiens d'idéaux qui nécessitent d'effectuer de nombreuses opérations arithmétiques sur les polynômes.

Nous nous sommes particulièrement intéressés à ces opérations afin d'en améliorer les performances, ce qui nous a amenés à concevoir deux types d'optimisation. Le premier type concerne les optimisations liées au matériel disponible sur les ordinateurs grands publics et plus particulièrement les processeurs. Le second type concerne les optimisations algorithmiques, comme par exemple, les multiplications modulaires.

NFLlib est sous la licence libre GNU General Public License v3.0 afin de favoriser sa diffusion dans le milieu scientifique, mais aussi public. Ainsi, nous souhaiterions que NFLlib incite les chercheurs à expérimenter des systèmes cryptographiques fondés sur les réseaux euclidiens d'idéaux dans toutes sortes de situations.

Notre bibliothèque est restreinte à un anneau de polynômes particuliers avec des modules particuliers ce qui diffère des bibliothèques plus génériques telles que NTL écrite principalement par Victor Shoup et FLINT [23]. Ces dernières couvrent largement des

outils mathématiques issus de la théorie des nombres et offrent ainsi de plus nombreuses possibilités. A contrario, c'est cette restriction qui nous a permis de fortement optimiser notre bibliothèque.

Il existe une autre bibliothèque spécialisée dans la cryptographie fondée sur les réseaux euclidiens nommée HElib pour *Homomorphic Encryption library*. Cette dernière, fondée sur NTL [40], est devenue une référence pour les bancs d'essai, car elle met en oeuvre un cryptosystème complet ainsi que de nombreuses optimisations. Notamment, NFLlib ne peut pas être comparé à HElib en termes de fonctionnalités, mais pourrait, par exemple, remplacer NTL dans HElib pour des constructions fondées sur un anneau de polynômes.

Dans la suite de ce chapitre nous présentons la bibliothèque, ses composants principaux ainsi que les optimisations. Puis nous comparons ses performances sur différents algorithmes (NTT, multiplication, addition, etc.) avec les bibliothèques citées précédemment. Nous terminons par la description de la mise en œuvre d'algorithmes fondés sur les réseaux euclidiens d'idéaux et nous explicitons les résultats de performance obtenus.

4.2 NFLlib : Présentation

Remarque.

1. Cette section fait usage de notions et notations présentées dans les chapitres précédents.
2. Le pronom « on » désigne dans la suite du document, un utilisateur quelconque de la bibliothèque.

Dans cette section, nous présentons NFLlib, une bibliothèque `C++` dédiée à la manipulation de polynômes employés dans cryptographie fondée sur les réseaux euclidiens d'idéaux. Plus précisément, cette bibliothèque accorde la possibilité de manipuler des polynômes sur $\mathrm{R}_p = \mathbb{Z}_p[x]/(x^n + 1)$, pour des nombres p et n permettant d'optimiser fortement les opérations sur les polynôme

Le point d'entrée de notre bibliothèque est la classe, nommée `poly` pour polynôme, suivante :

```
poly< class T, size_t degree, size_t sizeModulus >.
```

Sachant qu'un gabarit (`template` dans le langage `C++`) seul est inutilisable, trois paramètres sont à définir afin d'obtenir une classe fonctionnelle :

- `T` Le type de mot machine utilisé par les coefficients, en `C++` ces types sont `uint16_t, uint32_t ou uint64_t` pour des entiers non signés de respectivement 16, 32 et 64 bits.
- `degree` Le degré n du polynôme $x^n + 1$ qui définit R_p (qui doit être une puissance de 2).
- `sizeModulus` La taille en bits du module p, construit comme le produit de ℓ premiers de taille fixe : $p = p_1 \times \cdots \times p_\ell$.

La classe `poly` est consituée des fonctionnalités suivantes :

- Des opérateurs, comme ceux utilisées pour les types natifs, tels que `+, - , *, =` . Ces derniers sont surchargés afin d'augmenter l'accessible des opérations arithmétiques modulaires, la manipulation des données et la lisibilité du code.
- De gabarits `C++` afin de minimiser la dégradation des performances issues de la surcharge d'opérateur.
- Des fonctions de génération de bruit suivant différentes distributions (distribution uniforme, bornée, gaussienne discrétisée) pour échantillonner des polynômes dans R_p
- Des fonctions liées à la transformation des polynômes (NTT, CRT, importation, exportation)
- Des optimisations utilisant les instructions SSE et AVX2 pour des architectures processeurs compatibles.

Le paramètre clé de la classe `poly` est le type de mots machine `T`. Il définit les premiers p_i disponibles, leur nombre, le degré maximum n des polynômes et les portions de codes utilisées. En effet, l'algorithme est spécialisé et utilise des optimisations SIMD [1] spécifiques en fonction de `T` (particulièrement pour les mots de 32 et 16 bits).

Aucune des fonctions de la classe `poly` ne se fonde sur un code préexistant. Ce qui permet de faire appel à des instructions natives (scalaire ou vectorielles) disponibles sur les processeurs modernes. L'exception est le générateur de nombres pseudo-aléatoire Salsa20, pour lequel une implémentation assembleur existante est utilisée, et la fonction de relèvement du CRT qui utilise la bibliothèque GMP [56] (pour les entiers de taille arbitraire) si la taille du module utilisée dépasse la taille des registres processeurs.

Représenter les polynômes par leurs valeurs plutôt que leurs coefficients (forme évaluée) et utiliser le théorème des restes chinois afin d'améliorer les performances est devenue commun. Les performances de notre bibliothèque reposent sur le fait que la majorité des fonctions présentes ont été conçues avec des instructions natives ainsi que 4 choix majeurs dont l'efficacité est montrée dans la suite du chapitre. Ces choix sont :

1. la représentation CRT à taille fixe ; – section 4.2.1
2. une multiplication modulaire des scalaires optimisée ; – section 4.2.2
3. l'algorithme de NTT ; – section 4.2.3
4. le générateur de bruit uniforme et gaussien ; – section 4.2.4
5. Les optimisations utilisant les instructions SSE et AVX ;

4.2.1 Représentation CRT à taille fixe

Pour des raisons d'efficacités, les modules p utilisés dans notre bibliothèque spécialisée dans les réseaux idéaux sont le produit de ℓ premiers p_i de taille fixe. Cette taille, inférieure à la taille des mots machine présents dans les processeurs modernes, conduit à des performances élevées. Nous utilisons le fait que manipuler la représentation CRT $(a_1(\mathrm{X}), \dots, a_\ell(\mathrm{X})) \in \mathrm{R}_{p_1} \times \cdots \times \mathrm{R}_{p_\ell}$ d'un polynôme $a(\mathrm{X}) \in \mathrm{R}_p$, ou $a_i(x) = a(x) \bmod p_i$ est possible et tous les a_i peuvent être traités indépendamment.

Les nombres premiers qui composent l'ensemble des modules sont choisis en suivant les contraintes suivantes :

1. Leurs tailles doivent au plus être d'un mot machine moins 2 bits (p. ex. pour des mots machine de 32 bits les premiers doivent faire 30 bits). On peut alors effectuer une réduction modulaire paresseuse dans l'algorithme de la NTT ce qui améliore les performances d'environ 30% ;
2. Ils doivent satisfaire l'équation 4.1 pour un paramètre $s_0 = 2$ afin d'être utilisable dans l'algorithme de multiplication modulaire présenté dans la section 4.2.2 ;
3. Ils doivent être congruents à 1 $(\mathrm{mod} 2n)$ pour n une puissance de deux le plus grande possible afin de pouvoir trouver une racine n-ème de -1 afin d'être utilisable dans l'algorithme de NTT présenté dans la section 4.2.3.

La contrainte 2 assure que la contrainte 1 est satisfaite lorsque $s_0 \leq 2$. Par défaut, NFLlib est configurée avec $s_0 = 2$. De sorte à satisfaire la contrainte 3, nous avons fixé arbitrairement un degré maximum n_{max}. À noter a) que la contrainte est satisfaite quelque soit le degré $n \leq n_{max}$ et b) plus n_{max} est grand, moins les premiers satisfont la contrainte 3. Lorsque la taille des mots est de 16 bits, ces contraintes sont plus fortes que pour des mots machine de plus grande taille. Illustrons avec $n_{max} = 2048$, seul un unique premier de 14 bits satisfait la contrainte 3 (en supposant que $s_0 = 2$). D'un autre coté, pour des entiers de 64-bits, des milliers de premiers vérifient les contraintes, et ce, même pour de très grands polynômes avec un $n_{max} = 2^{20}$. L'algorithme 15, présenté plus loin dans ce chapitre, trouve et renvoie un premier qui satisfait les contraintes 1-3.

Trouver au préalable ces premiers et les conserver est avantageux d'un point de vue des performances. À cet effet, ils ont été inclus dans le fichier de paramètres `params.hpp` avec $n_{max} = 2^{10}$ pour $s = 16$ (2 premiers), $n_{max} = 2^{15}$ lorsque $s = 32$ (291 premiers)

1. *Single Instruction Multiple Data* voir la section 1.3.1 du chapitre 1.

et $n_{max} = 2^{20}$ lorsque $s = 64$ (premiers limités volontairement à 1000). Évidemment, l'utilisateur peut redéfinir les valeurs de s_0 et n_{max} dans NFLlib .

4.2.2 Optimisation de la multiplication modulaire

Comme expliqué dans la section 4.2.1, la bibliothèque NFLlib est fournie avec des nombres premiers de 14, 30 et 62 bits. Comme souligné dans [39], calculer une réduction modulaire avec un entier fixé en utilisant des coefficients de Newton suivit d'une multiplication et d'un ajustement est plus efficace que la division effectuée matériellement sur un processeur et ce même par rapport à un processeur moderne 64-bits.

Lors de la construction de la bibliothèque, nous avons observé que le compilateur `gcc` optimise de lui-même les multiplications modulaires sur des mots de 16 et 32 bits (*par ex.* pour des premiers de 14 et 30 bits). Néanmoins, ce n'est pas le cas pour des mots de 64 bits. Dans cette section, nous nous focalisons sur le problème suivant :

Comment diviser un entier de deux mots par un entier d'un seul mot ?

Ce problème, étudié en profondeur dans le chapitre 1 ainsi que dans [39]. Dans cet article, les auteurs proposent un algorithme (Alg. 4 de l'article cité) 30% plus performant que l'algorithme fondé sur les coefficients de Newton [39]. D'ailleurs le premier algorithme présenté est inclus dans la version 4.3 de la bibliothèque GMP.

Dans NFLlib, les premiers sont formés de telle sorte que leurs deux bits de poids forts valent 0, et les algorithmes dans [39] sont, quant à eux, optimisés pour des nombres dont le bit significatif est à 1. Dans le reste de la section, nous décrivons un nouvel algorithme qui perfectionne significativement [39] pour des nombres p plus petits que la base d'un mot machine $\beta = 2^s$, comme illustré dans la table 4.1.

Table 4.1 – Durée par multiplication de polynômes sous forme évaluée de degré 1024 avec des valeurs modulo un entier de 62 bits (résultat moyen obtenu à partir de 100 000 multiplications de polynômes sur un processeur Intel Xeon CPU E5-2666 v3 à 2.90GHz). Nous avons implémenté les algorithmes 1 et 4 de [39] (avec $4p$ au lieu de p et deux soustractions conditionnelles à la fin). Néanmoins, ces derniers s'avèrent moins performants que notre algorithme d'un ordre de grandeur.

Algorithme	Naïf (*i.e.* utiliser %)	[39] Alg. 1	[39] Alg. 4	NFLlib **Alg. 15**
Mult. mod. de polynôme (µs)	29.8µs	15.5µs	12.9µs	**2.90µs**

Supposons que l'on veuille calculer une réduction modulaire avec un module p tel que :

$$(1 + 1/2^{3s_0}) \cdot \beta/(2^{s_0} + 1) \ < \ p \ < \ \beta/2^{s_0} \,, \tag{4.1}$$

Pour un entier $1 \leq s_0 \leq s - 1$ (à noter que tous les premiers de 62 bits inclus dans NFLlib vérifient l'équation 4.1). Notons $\langle u_1, u_0 \rangle$ la décomposition de p en mots plus petits que β tel que $u = u_1 \cdot \beta + u_0$. l'Alg. 15 décris notre proposition d'algorithme de réduction modulaire.

Algorithm 15 Réduction modulaire dont le module vérifie l'équation 4.1

Require : $\mathrm{A} = a_1, a_0$ et n vérifie (4.1), $1 \leq s_0 \leq \ell - 1$ bits de marge et $m_0 = \lfloor \beta^2/n \rfloor \bmod \beta$

```
procedure Reduction(A)
    q ← m0 · a1 + 2^s0 · a mod β^2                ▷ ≈ (v · a)/β
    r ← A − ⌊q/β⌋ · n mod β                       ▷ mullow
    if r ≥ n then
        r ← r − n
    end if
    return r
end procedure
```

Nous avons le théorème suivant :

Théorème 1. *Supposons que* $1 \leq s_0 \leq s - 1$ *et* p *vérifies Eq.* (4.1) *et que* $u = \langle u_1, u_0 \rangle \in [0, p^2)$. *Soit* $v = \langle v_1, v_0 \rangle = \lfloor \beta^2/p \rfloor$. *Alors Alg. 15 avec les entrées* (u, p, v_0, s_0) *restitue* $(u \bmod p)$.

4.2.3 Un algorithme de NTT paresseux

Nous utilisons l'algorithme de NTT proposé par D. Harvey dans [24]. Cet algorithme fait appel à deux techniques afin de réduire son coût calculatoire : des quotients précalculés pour accélérer la multiplication modulaire et une réduction paresseuse (c.-à-d. la taille des entrées et des sorties peuvent être le double de celle du module). L'utilisation du précalcule de quotients dans la NTT a déjà été implémenté dans la bibliothèque NTL [40]. Toutefois, Harvey a montré que certaines méthodes de NTL pouvaient être modifiées de telle façon que la sortie se trouve dans $[0, 2p)$ lorsque l'entrée est dans $[0, p)$ en utilisant uniquement une soustraction conditionnelle (à la place des 3 dans l'algorithme initial). Cette modification donne un substantiel gain de performances (d'environ 30%), comme montré dans [24]. Cela justifie le fait de sélectionner des premiers comme présenté dans la section 4.2.2.

Notre bibliothèque ne possède pas de contributions particulières concernant la NTT, nous montrons seulement dans ce chapitre sont efficacité par rapport à la Bluestein FFT utilisée dans HElib (voir la section 4.3).

4.2.4 Génération de bruit uniforme et gaussien

La génération de bits aléatoires de NFLlib est fondée sur le code qui accompagne l'article [19], de T. Güneysu, T. Oder, T. Pöppelmann, P. Schwabe. Une version écrite en assembleur du chiffrement par flot Salsa20 (en mode CRT) par D. J. Bernstein [3] est initialisée avec le générateur aléatoire classique des systèmes de type Unix, `/dev/random`. Le générateur proposé par les auteurs de [18] n'est pas *thread-safe*, c'est-à-dire que lors d'appels parallèles au générateur aléatoire, ce dernier peut être appelé plusieurs fois avec le même vecteur d'initialisation (données lues dans `/dev/random`). Afin de forcer les appels à être séquentiels (`thread-safe`) nous avons ajouté la directive de compilation `#pragma omp critical`.

En cryptographie fondée sur les réseaux euclidiens, l'utilisation d'aléa issu d'une distribution gaussienne à la place d'une distribution uniforme est fréquente. Par conséquent, NFLlib est fournie avec un générateur d'aléa qui suit une distribution gaussienne D_σ discrète (sur $\mathbb{Z}$) et centrée. La distribution gaussienne discrète sure $\mathbb{Z}$ avec une déviation standard $\sigma > 0$ et de centre $c \in \mathbb{R}$ est définie par :

$$\forall x \in \mathbb{Z}, \mathrm{D}_{\sigma,c}(x) = \frac{\rho_{\sigma,c}(x)}{\rho_{\sigma,c}(\mathbb{Z})},$$

$$\text{avec } \rho_{\sigma,c}(x) = \exp\left(-\frac{(x-c)^2}{(2\sigma^2)}\right) \text{ et } \rho_{\sigma,c}(\mathbb{Z}) = \sum_{y \in \mathbb{Z}} \rho_{\sigma,c}(y).$$

Génération de bruit gaussien dans NFLlib . La déviation standard σ est spécifiée en tant qu'argument au *template*, ainsi durant le processus de compilation, un tableau qui stocke les probabilités $p_y = \Pr[x \leq y : x \leftarrow \mathrm{D}_{\sigma,0)}]$ pour $y \in \mathbb{Z} \cap [-\tau \cdot \sigma, \tau \cdot \sigma]$ est généré. La valeur de τ est choisie telle que les valeurs échantillonnées à partir de $\mathrm{D}_{\sigma,0}$ sont contenues dans l'intervalle avec une très forte probabilité. Lors de l'exécution de l'algorithme, on tire d'abord (paresseusement) un bruit uniforme $z \in [0, 1)$, puis l'on effectue une recherche binaire dans le tableau généré à la compilation afin de déterminer un $y \in \mathbb{Z}$ tel que $z \in [p_{y-1}, p_y)$. Le tableau 4.2 présente le nombre de cycles par bits échantillonnés pour la distribution uniforme ainsi que pour une distribution gaussienne quelconque.

Table 4.2 – Nombre de cycles par bits générés dans NFLlib pour la génération de polynômes de 1024 coordonnées (moyenne calculée sur 50 millions d'appels au générateur effectués sur un processeur Intel Xeon e5-2666 v3 à 2.90GHz). Nous avons fait les calculs de probabilités (précision et limite de la *queue* de la courbe) de manière à ce que la distance statistique entre la sortie de notre générateur et une gaussienne idéale soit inférieure à 2^{128}

Distribution	Uniforme	$\mathrm{D}_{3.19}$	D_{300}
cycles / bit généré	0.4	1.39	3.43

4.3 Évaluation des performances et comparaison avec NTL, FLINT et HElib

Dans cette section, nous analysons les performances de notre bibliothèque et comparons les différents tests de performance effectués avec les bibliothèques NTL [40], flint [23] , et HElib [21].

Rappelons succinctement que NTL et flint sont des bibliothèques génériques qui offrent la possibilité d'utiliser des polynômes dans n'importe quel anneau. De plus HElib est une bibliothèque logicielle (fondée sur NTL) qui met en œuvre une version optimisée de cryptosystème homomorphe de Brakerski-Gentry-Vaikuntanathan [4] (BGV). Nous avons choisi de nous comparer à ces bibliothèques, car leurs utilisations sont très répandues dans la littérature qui traite de la mise en œuvre de cryptosystèmes basés sur les réseaux euclidiens. En conséquence, nous avons restreint ces bibliothèques à la configuration particulière de NFLlib, c'est-à-dire, sur $\mathrm{R}_p = \mathbb{Z}_p[\mathrm{X}]/(\mathrm{X}^n + 1)$ avec le module p comme dans la section. 4.2.1.

Configuration Nous avons évalué les performances de NFLlib face à NTL, flint et HElib sur la génération de polynômes aléatoires, la NTT et la NTT inverse, l'addition et la multiplication modulaire sous la forme évaluée (forme NTT). Tous les tests de performances accomplis se fondent sur l'ensemble de paramètres suivants :

(1) $n = 256$ avec un module p de 14 bits,
(2) $n = 512$ avec un module p de 30 bits,
(3) $n = 1024$ avec un module p de 62 bits,
(4) $n = 1024$ avec un module p d'environ 6200 bits (produits de 100 modules de 62 bits).

Comme attendu, les mots machine utilisés dans la NFLlib sont de 16 bits pour le paramètre (1) et de 32 bits pour le (2). Concernant NTL, nous avons utilisé la classe `zz_pX` (1 mot machine de 64 bits par coefficient) pour les paramètres (1), (2) et `ZZ_pX` pour les suivants. Pour flint, c'est le type `fmpz_mod_poly_t` qui a été utilisée. Finalement, HElib [2]

2. Dans HElib, l'instantiation d'un `FHEContext` – objet stockant la décomposition de modules – est nécessaire aux objets `DoubleCRT`. Ce constructeur tente de produire des premiers dont la taille, fixée directement dans le code par la variable `FHE_size`, tend vers 44 bits. Cette taille est inclue une classe `DoubleCRT` dont la représentation équivaut à celle de NFLlib .

4.3 Évaluation des performances et comparaison avec NTL, FLINT et HElib

Tous les tests de performances ont été effectués sur une instance `c4.2xlarge` d'Amazon Web Services afin de simplifier la reproductibilité des résultats obtenus. La partie matérielle de cette instance se composait d'un processeur Intel Xeon CPU E5-2666 v3 (Haswell) à 2900 MHz et 15 Go de mémoire vive.[3] Concernant les bibliothèques, la machine était équipée de `gcc` 4.9, GMP 6.0, NTL 8.1 et FLINT 2.5.

Remarque.

Afin de démontrer les performances de notre bibliothèque sur différentes architectures, nous avons testé les performances de la NTT sur un MacBook Air de Apple (nommé `macbookair`) équipé d'un processeur Intel Core i7-4650U à 1700 MHz et 8 Go de RAM. Nous avons employé le compilateur natif du système d'exploitation `clang++`, concernant les bibliothèques, GMP 6.0, NTL 8.1 et FLINT 2.5.

Table 4.3 – Temps de génération d'un polynôme aléatoire dans $\mathbb{Z}_p[X]/(X^n+1)$ en utilisant les fonctions incluses dans les différentes bibliothèques sur le `c4.2xlarge`.

Bibliothèques	NTL	FLINT	HElib	NFLlib
	`random`	`fmpz_mod_poly_randtest`		`nfl::uniform`
$(1) = (256, 14)$	9.2µs	4.8µs	69µs	**0.6µs**
$(2) = (512, 30)$	23.2µs	9.1µs	135.5µs	**2.6µs**
$(3) = (1024, 62)$	173.0µs	18.3µs	540.0µs	**9.7µs**
$(4) = (1024, 6200)$	8675µs	1082µs	37929µs	**1029.6µs**

Génération de polynômes aléatoires afin d'évaluer les performances de la génération aléatoire, nous avons employé la fonction `ntl::random` de NTL, la fonction `fmpz_mod_poly_randtest` de FLINT et la fonction par défaut de génération de NFLlib. Les résultats obtenus sont présentés dans la Table 4.3. À noter que la bibliothèque FLINT met en œuvre l'algorithme Mersenne Twister qui ne devrait pas être utilisée dans un contexte cryptographique.

NTT et iNTT Manipuler les polynômes dans leur représentation évaluée par la NTT est un point clé de notre optimisation, puisque les additions et les multiplications deviennent linéaires en nombre de coefficients. Les performances de la NTT sont reportées dans la Table 4.4 (*incluant* les convolutions circulaires négatives). À noter que NTL fournit une fonction de type NTT, `ToffRep` pour les `zz_pX` et `toFFTRep` pour les `ZZ_pX`. Inversement, de telles fonctions ne semblent pas être disponibles dans la bibliothèque FLINT[4]. Dans HElib la classe `DoubleCRT` possède deux fonctions pour convertir à partir de (par convolution circulaire négative et NTT) et vers (via inverse NTT et l'inverse de la convolution) un polynôme de type `ZZX`.

Optimisations SSE et AVX2 Par nature, les opérations sur les polynômes sont hautement parallélisables (la même opération est appliquée sur de nombreuses données). Ainsi, utiliser les instructions *Streaming SIMD Extensions* (SSE) et *Advanced Vector Extensions* (AVX) peut fortement améliorer les performances de ce type d'opérations. Par ailleurs les auteurs de [18] l'ont montré pour les signatures et le chiffrement sur les réseaux euclidiens.

3. Les technologies TurboBoost et Hyperthreading furent désactivées durant la campagne de tests.

4. Nous avons négligé le cout linéaire des convolutions négatives afin de mitiger l'impact d'une mise en œuvre peu optimisée

Table 4.4 – Temps pour calculer une NTT d'un polynôme de $\mathbb{Z}_p[X]/(X^n+1)$ en utilisant (si disponible) les fonctions fournies par les bibliothèques.

(a) NTT sur un `c4.2xlarge` avec `gcc`

Bibliothèque	NTL `toFFT/ToFFT`	flint	HElib `conv(DoubleCRT,ZZX)`	NFLlib
$(1) = (256, 14)$	7.2µs	–	33.7µs	**2.5µs**
$(2) = (512, 30)$	14.7µs	–	70.7µs	**4.5µs**
$(3) = (1024, 62)$	45.7µs	–	317.7µs	**13.9µs**
$(4) = (1024, 6200)$	33921µs	–	23240µs	**1341.0µs**

(b) NTT sur `macbookair` avec `clang`

Bibliothèque	NTL	flint	HElib	NFLlib
$(1) = (256, 14)$	7.7µs	–	37.6µs	**1.7µs**
$(2) = (512, 30)$	16.0µs	–	74.9µs	**5.7µs**
$(3) = (1024, 62)$	47.5µs	–	333.8µs	**15.3µs**
$(4) = (1024, 6200)$	34799µs	–	24713µs	**1163.4µs**

NFLlib utilise des instructions SSE et AVX2 dans l'algorithme de NTT ainsi que pour les opérations modulaires sur des mots machine de 16 et 32 bits. Nous comparons la NTT de NFLlib avec la NTT optimisée avec AVX de Güneysu *et al.* AVX-optimized [18] (ou de nouveau, la NTT inclut la convolution circulaire négative) sur le `c4.2xlarge`.

La mise en œuvre de GOPS est fondée sur le type natif `double` pour un module p de 23 bits (réduction paresseuse) et nécessite 5030 cycles processeur. En comparaison, NFLlib peut être initialisée avec des premiers de 14 ou 10 bits et nécessite respectivement 3324 et 7334 cycles avec les instructions SSE4, 2767 et 5956 cycles lors de l'utilisation des instructions AVX2. À noter que la version 62-bits de la NTT (sans instructions SIMD) prend 10020 cycles.

4.4 Cas d'utilisation de NFLlib

4.4.1 Échange performant de clés

Dans cette section, nous utilisons un équivalent du protocole de transport de clés RSASVE du NIST SP 800 56B en nous fondant sur [34] afin d'illustrer les performances de notre bibliothèque dans un cas d'utilisation concret.

Le client choisit un message aléatoire et, en utilisant la clé publique du serveur, le chiffre. Ce message est envoyé au serveur qui déchiffre cette valeur aléatoire de laquelle est dérivé, par une fonction de hachage, un secret commun.

Côté Serveur De manière générale, un serveur gère de nombreux clients, le problème principal provient du cout calculatoire des opérations qu'il doit effectuer pour mener à bien le protocole. Donc, nous nous focalisons sur les coûts du côté serveur.

Authentification Serveur et confidentialité persistante[5] La clé publique du serveur peut prendre la forme d'un certificat signé par, virtuellement, n'importe quel algorithme (*par ex.* DSA) tel que le client puisse vérifier et être convaincu de l'identité du serveur. En d'autres termes, le client doit pouvoir s'assurer de l'identité du serveur afin d'éviter toutes fraudes. Une telle opération n'a pas de coût calculatoire pour le serveur et nous ne la prenons pas en considération. Comme suggéré dans [29], le serveur peut envoyer 2 clés : une première signée pour prouver son identité, et une clé temporaire afin de renforcer la persistance de la confidentialité. Puis le client envoie deux secrets, le secret commun est alors dérivé des deux secrets envoyés en utilisant une fonction de dérivation de clés (une fonction de hachage). Comme l'une des clés publiques est signée, le client sait que seul le serveur identifié peut déterminer le secret commun. De plus, si la clé temporaire est détruite à la fin de l'échange de clés, la persistance de la confidentialité est conservée. En conséquence, conserver la confidentialité implique qu'en ce qui concerne le serveur le coût en communication et en calculs est multiplié par deux.

Nous avons utilisé le système cryptographique fondé sur Ring-LWE de [34]. Le code `C++` pour les fonctions de chiffrement et le déchiffrement est présenté dans les algorithmes 16 et 17.

Algorithm 16 Chiffremment à clé publique fondée sur Ring-LWE.

```
using value_t = typename P::value_type;

P tmpu = nfl::gaussian<value_t>(g_prng);     // pas de mul. bruit
P tmpe1 = nfl::gaussian<value_t>(g_prng, 2); // mul. bruit: 2
P tmpe2 = nfl::gaussian<value_t>(g_prng, 2); // mul. bruit: 2
tmpe2 += m;

tmpu.ntt_pow_phi();
tmpe1.ntt_pow_phi();
tmpe2.ntt_pow_phi();

resa = tmpu * pka + tmpe1; //multiplication et addition de poly.
resb = tmpu * pkb + tmpe2;
```

Algorithm 17 Fonction de déchiffrement à clé publique fondée sur Ring-LWE.

```
m = resb - resa * s;
m.invntt_pow_invphi();

for(auto & v : m) // auto détermine le type à la compilation
  v = (v<modulus/2) ? v%2 : 1-v%2;
```

Ces deux algorithmes (16 et 17), écrits en `C++` avec NFLlib, donnent un aperçu de la facilité d'utilisation de la bibliothèque. Les polynômes, en plus d'avoir des fonctions comme tout autre objet, se manipulent comme des types natifs.

5. La confidentialité persistante est une propriété d'un protocole cryptographique dans laquelle la découverte de la clé privée d'un correspondant par un opposant ne compromet pas le contenu des communications précédentes.

Table 4.5 – Nombre d'échanges de clés par secondes sur un serveur équipé d'un processeur i7-4770 en utilisant *un seul cœur*. Lorsque le processeur parallélise le programme sur 4 de ses coeurs, les performances sont multipliées par un facteur 4. Aucun modèle standard d'implémentation de RSA15360 n'existe et notre bibliothèque ne fonctionne pas avec 80 bits de sécurité pour cette application (entrées N/A dans le tableau ci-dessous). Nous avons obtenu les résultats de RSA et ECDH (pour des courbes de type p) avec le test de vitesse de openssl 1.0.1f. Les résultats de la ligne NFLlib correspondent au nombre de déchiffrements par seconde issus de notre implémentation de [34].

Protocole	80 bits	128 bits	256 bits
RSA	7.95 Kops/s	0.31 Kop/s	N/A
ECDH	7.01 Kops/s	5.93 Kops/s	1.61 Kop/s
NFLlib	N/A	1020 Kops/s	508 Kops/s

La Table 4.5 présente les performances du protocole pour 80, 128 et 256 bits de sécurité. Dans RSA et NFLlib, le serveur doit nécessairement déchiffrer alors que dans ECDH il doit calculer une exponentiation modulaire. NFLlib offre la possibilité de gérer un plus grand nombre de clients ou d'utiliser moins de temps processeur pour le même nombre de clients. Puisque l'écart est proche d'un facteur 200, on peut envisager de traiter 10 fois plus de clients en utilisant 10 fois moins de temps processeur et d'améliorer la sécurité d'un facteur 2 par rapport à ECDH avec 128 bits de sécurité (ou maintenir le niveau de sécurité et ajouter la propriété de confidentialité persistante).

4.4.2 Chiffrement homomorphe avec NFLlib

Une application bien connue de la cryptographie fondée sur les réseaux d'idéaux est le chiffrement homomorphe. Les premières mises en œuvre de cryptosystèmes FHE furent peu performantes. Cependant, la communauté scientifique a réalisé de grands progrès en 6 années et aujourd'hui certains calculent peuvent être réalisés en temps raisonnable. Néanmoins, la procédure d'amorçage (*bootstrapping* dans la littérature) – étape nécessaire à l'obtention d'un chiffrement entièrement homomorphe – reste un goulot d'étranglement.

Afin de surmonter ce dernier, la communauté cryptographique a porté son attention sur la cryptographie quelque peu homomorphe (SHE), c'est-à-dire des constructions dont le nombre d'opérations homomorphes est borné et plus particulièrement les opérations de multiplication. Cependant, même avec une configuration simplifiée, la taille des paramètres est très importante pour évaluer de façon homomorphe des fonctions non triviales. (*p. ex.* [33, 13]). Pour manipuler environ 40 niveaux, on utilise des paramètres tels que $2^{10} \leq n, \log p \leq 2^{20}$. Ainsi les paramètres fixés de NFLlib ont été choisis pour manipuler des polynômes dont le degré peut atteindre 2^{20} et les modules 62000 bits. Maintenant, en se reposant sur les résultats de la section 4.3, une implémentation sur R_p qui exploite NFLlib donnerait un gain de performances d'un facteur 15 à 50 par rapport à une implémentation fondée sur flint ou NTL. À titre d'exemple, nous avons modifié la version open source de l'implémentation du système cryptographique quelque peu homomorphe FV [33] en remplaçant directement flint par NFLlib . Nous avons obtenu les améliorations affichées dans la Table 4.6.

Table 4.6 – Le gain de performance obtenu pour les fonctions de chiffrement est de déchiffrement sont d'un ordre de grandeur. Concernant l'opération d'addition homomorphe se gain s'élève à deux ordres de grandeur. Néanmoins. On remarque que le gain obtenu pour la multiplication homomorphe est moins important (5.5×) car cet algorithme demande de faire une opération couteuse de relèvement (voir CRT au chapitre 1) pour laquelle NFLlib ne donne pas d'avantage particulier.

	`Encrypt`	`Decrypt`	`Hom. Add.`	`Hom. Mult.`
[33] avec flint	26.7ms	13.3ms	1.1ms	91.2ms
[33] avec NFLlib	**0.9ms**	**0.9ms**	**0.01ms**	**17.2ms**
Gain	×30	×15	×110	×5.5

4.5 Conclusion

Nous avons présenté dans ce chapitre nfllib, une bibliothèque `C++` libre optimisée pour la manipulation de polynômes sur R(.) p. Nous avons montré que, grâce aux contraintes que nous avons défini sur le degré des polynômes et la forme des modules, nous avons pu mettre en oeuvre de nombreuses optimisations poussant ainsi les performances de NFLlib au-delà de celles obtenues avec NTL et flint. Nous espérons que ses performances et sa facilité d'utilisation contribueront à la construction de prototypes fondés sur les réseaux euclidiens.

Chapitre 5

Private Information Retrieval for Everyone

Table des matières

5.1 Introduction 62
5.2 Choix des protocoles 63
5.2.1 Chiffrement homomorphe 63
5.2.2 Private Information Retrieval 64
5.3 Le protocole de XPIR 65
5.3.1 Cryptosystème fondé sur Ring-LWE 67
5.3.2 La représentation NTT-CRT et transformations 68
5.3.3 Coefficients de Newton précalculés 70
5.3.4 Performances de chiffrement et de déchiffrement 71
5.3.5 Achitecture et interface de programmation 72
5.4 Performance et cas d'utilisations 74
5.4.1 Paramètres expérimentaux 74
5.4.2 Débit de traitement sur les données statiques 76
5.4.3 Problèmes d'accès 78
5.4.4 Haut débit de traitement sur les données dynamiques 79
5.4.5 Latence sur les données statiques/dynamiques 81
5.4.6 Le cas d'utilisation de la bourse de New York 83
5.4.7 Autres cryptosystèmes 84
5.5 Conclusion 85

Résumé : *Pièce maitresse de ce mémoire, nous présentons dans ce chapitre notre application de retrait d'information privé. Fondée sur un système de chiffrement écrit avec* NFLlib *, nous évaluons notre proposition de PIR sur des cas réaliste afin de démontrer sa praticabilité. Ainsi, concevoir un service de vidéo à la demande ou d'accès à des flux d'informations continus respectueux des centres d'intérêt de leurs utilisateurs semblent des applications envisageable.*

5.1 Introduction

Nous avons vu dans le chapitre 3 l'évolution du retrait d'information privé. Nous ferons donc seulement un bref rappel sur le *Private Information Retrieval* (PIR). L'objectif d'un protocole de retrait d'information privé est de fournir à son utilisateur le même niveau de confidentialité que s'il téléchargeait l'intégralité d'une base de données publique, mais avec un coût en communication sous-linéaire. En 1995, Chor, Goldreich, Kushilevitz, et Sudan dans [9] donnèrent le coup d'envoi de la recherche sur le PIR. Dans cet article fondateur, ils présentent un ensemble de protocoles PIR dont la confidentialité de l'utilisateur est obtenue par l'envoi de requêtes sur des bases de données répliquées.

Dans ce chapitre, nous nous concentrerons sur notre contribution concernant les protocoles PIR qui ne nécessitent pas de bases de données répliquées et dont la sécurité est fondée sur des algorithmes cryptographiques. Ces protocoles sont nommés PIR computationnels (cPIR).

L'un des problèmes majeurs des protocoles cPIR est qu'ils sont coûteux en matière de calculs. Afin de répondre à la requête d'un utilisateur, le serveur effectue des opérations sur la totalité des éléments de la base de données. Si dans le protocole l'algorithme du serveur ne s'appliquait pas à toute la base de données, la confidentialité des utilisateurs pourrait être compromise. En effet, le serveur pourrait, à partir de ses informations partielles, inférer les éléments de la base qui intéressent ses utilisateurs. Le coût calculatoire de génération de la réponse par le serveur est donc linéaire en la taille de la base de données. Ce coût est d'autant plus important que les cryptosystèmes employés dans l'algorithme de génération de la réponse font parfois usage d'opérations complexes comme des multiplications et exponentiations modulaires à partir de modules de grande taille (plus de 1000 bits).

Dans un article [53] publié à NDSS'07 Sion et Cabunar traitent de la praticabilité du cPIR. Les auteurs démontrent que, en l'état actuel, les protocoles cPIR dont les systèmes cryptographiques se fondent sur la théorie des nombres, sont dans les faits inutilisables. Il serait toujours plus rapide de télécharger l'intégralité de la base de données que de calculer la réponse par l'un des protocoles cPIR étudié. En effet, ces protocoles traitent difficilement la base de données à plus de 1 Mb/s, car ils emploient des cryptosystèmes dont la sécurité est fondée sur la difficulté de factoriser un module RSA d'au moins 1024 bits. Ainsi, pour reprendre l'exemple donné dans le chapitre 3, si l'on prend le débit moyen en France en 2014 télécharger l'intégralité d'une base de données est plus rapide d'au moins un demi-ordre de grandeur.

Nous ferons en premier lieu dans ce chapitre quelques rappels rapides sur le chiffrement homomorphe et le retrait d'informations privé. Puis nous verrons plus en détail le protocole de XPIR avec ses optimisations et son interface de programmation. Puis nous nous concentrerons sur les performances obtenues avec notre proposition ainsi que sur des cas d'utilisation réalistes. Puis nous terminerons par la conclusion.

5.2 Choix des protocoles

5.2.1 Chiffrement homomorphe

Dans XPIR, nous avons choisi d'utiliser un système de chiffrement fondé sur Ring-LWE pour sa relative simplicité de programmation avec NFLlib et pour ses performances (nous abordons ce sujet dans le chapitre 2).

Version symétrique d'un cryptosystème fondée sur Ring-LWE

SKE.ParamGen($1^k, h_a$) :
Entrée : Le paramètre de sécurité k ; Un nombre d'additions h_a
Sortie : Un module q ; Le degré des polynômes n ; Une distribution aléatoire χ

SKE.KeyGen(q, n) :
Entrée : Un module q ; Le degré des polynômes n
Sortie : Un polynôme dans $\mathrm{R}_q = \mathbb{Z}_q[\mathrm{X}]/ < \mathrm{X}^n + 1 >$

1. Sortie : $s \leftarrow \chi$

SKE.Encrypt(s, db) :
Entrée : Une clé secrète s dans l'anneau de polynômes R_q ; Un message db dans l'anneau de polynômes R_q dont les coefficients se trouvent dans $[0..t[$
Sortie : Un chiffré $(a, b) \in \mathrm{R}_q^2$

1. $a \leftarrow \mathrm{R}_q$, $e \leftarrow \chi$
2. $e' = e * t_v + db$ ou t_v est le polynôme constant dans R_q valant t
3. $b = (a * s) + e'$
4. Sortie : (a, b)

SKE.Decrypt($s, (a, b)$) :
Entrée : Une clé secrète $s \in \mathrm{R}_q$; Un chiffré $(a, b) \in \mathrm{R}_q^2$
Sortie : À message $db \in \mathbb{Z}_t^n$

1. $e = b - (a * s)$
2. Sortie : $db = e \mod t$

SKE.Add($(a_1, b_1), (a_2, b_2)$) :
Entrée : Deux chiffrées respectivement des messages db_1 et db_2
Sortie : Un chiffré qui se déchiffre en $db_1 + db_2 \mod t$

1. Sortie : $(a_1 + a_2, b_1 + b_2)$

SKE.Absorb($db, (a, b)$) :
Entrée : Un polynôme $m \in \mathrm{R}_q$ dont les coefficients se trouvent dans $\{0..t-1\}$; Un chiffré $(a, b) \in \mathrm{R}_q^2$, valeur chiffrée d'un polynomial *constant* m
Sortie : Un chiffré qui se déchiffre en $db * m$

1. Sortie : $(db * a, db * b)$

Nous avons aussi écrit une fonction nommée **SKE.ParamGen**. Ses paramètres d'entrée sont le paramètre de sécurité k et un nombre maximum d'additions h_a. À partir de ces arguments, cette fonction rend un ensemble de paramètres permettant de définir les autres algorithmes.

Pour des raisons de performances nous avons limité la sortie de cette fonction. Il faut en particulier que n soit dans l'ensemble $1024, 2048, 4096$ et que q soit un multiple de premiers de 60-bits tels que chaque premier soit congru à 1 modulo $2n$ pour utiliser la NTT (voir chapitre 1).

En se reposant sur l'analyse de l'article [5], ce cryptosystème assure que les chiffrés sont indistinguables les uns des autres si le problème standard Ring-LWE l'est (pour les paramètres choisis). La difficulté de ce problème est l'un des postulats les plus importants utilisés pour construire des cryptosystèmes fondés sur les réseaux euclidiens.

Il est important de remarquer que chaque clair peut donner lieu à plusieurs chiffrés, et donc les chiffres doivent être plus grands que les messages clairs. Nous notons $\mathrm{F} > 1$ le

facteur d'expansion [1] associé. À titre de référence, la figure 5.1 ci-dessous présente quelques tailles de chiffrés et de clairs pour différents paramètres de notre cryptosystème.

Paramètres	Max Sec	Clairs	Chiffrés	F
(1024,60)	97	$\leq$ 20kbit	128kbit	$\geq$ 6.4
(2048,120)	91	$\leq$ 100kbit	512kbit	$\geq$ 5.12
(4096,120)	335	$\leq$ 192kbit	1Mbit	$\geq$ 5.3

Figure 5.1 – Paramètres utilisés dans notre cryptosystème Ring-LWE. Les chiffrés sont composés de deux polynômes. Le premier paramètre définit le nombre de coefficients par polynôme et le second le nombre de bits par coefficients (stockés dans des registres de 64 bits). La taille des chiffrés peut être aisément déductible de ces valeurs. La sécurité théorique maximum est atteinte si et seulement si suffisamment de bruit est inclus dans les chiffrés et que la distribution suivie correspond au niveau de sécurité. La taille des messages est (logarithmiquement) inversement proportionnelle au nombre de sommes à réaliser. Le facteur d'expansion reste donc très proche de l'optimum.

5.2.2 Private Information Retrieval

Nous utilisons un protocole cPIR fondé sur [55] décrit ci-après dont l'avantage est qu'il s'exécute avec n'importe quel cryptosystème homomorphe additif.

Protocole de cPIR basique

Configuration (utilisateur) :

1. Créer une instance du cryptosystème avec k bits de sécurité

Génération de la requête pour l'élément i_0 :

1. Pour i de 1 à n générer le i-ème élément de requête q_i tel que q_i soit
   - Un chiffré de 0 si $i \neq i_0$
   - Un chiffré de 1 si $i = i_0$
2. Envoyer l'ensemble ordonné $\{q_1, \cdots, q_n\}$ au serveur de la base de données

Génération de la réponse :

1. Soit ℓ_0 le nombre de bits que peut absorber un chiffré
2. Pour i de 1 à n
   - Découper m_i en blocs de ℓ_0 bits notés $m_{i,j}$ pour $j \in [1..\lceil \ell/\ell_0 \rceil]$
3. Pour j de 1 à $\lceil \ell/\ell_0 \rceil$
   - Calculer $\mathrm{R}_j := \mathrm{S}um_{i=1}^{n} \mathrm{A}bsorb(m_{i,j}, q_i)$
4. Retourner $\mathrm{R} = (\mathrm{R}_1, \ldots, \mathrm{R}_{\lceil \ell/\ell_0 \rceil})$

Extraction de la réponse :

1. Déchiffrer l'ensemble ordonné R des polynômes formant la réponse et reconstituer m_{i_0} en concaténant les blocs déchiffrés.

La sécurité de ce protocole repose sur l'indistingabilité des requêtes pour deux éléments différents de la base de données. Ainsi le fait d'utiliser un cryptosystème qui a une propriété d'indistinguabilité face aux attaques à message clair choisi assure que deux requêtes pour deux éléments différents sont indistinguables.

À partir de ce protocole simple, si on note n le nombre d'éléments dans la base de données et ℓ la taille de chacun des éléments, la taille de la requête est de n fois la taille d'un chiffré et la taille de la réponse est d'environ $\ell \times \mathrm{F}$ avec F le facteur d'expansion du cryptosystème employé. La taille de requête peut se réduire en regroupant les éléments de la base de données par groupes de α éléments qui donnent une base de données de $\lceil n/\alpha \rceil$

1. La taille d'un chiffré en bits divisé par la taille du message clair associé.

éléments de taille $\ell \times \alpha$. La version récursive ce protocole réduit aussi la taille de la requête au détriment de la taille de la réponse.

Protocole cPIR récursif

Génération de la requête (Client) :
Entrée : paramètres PIR (n, ℓ, α, d), paramètres crypto *Enc.Params*, indexe choisi i
Sortie : Requête

1. Redéfinir $n = \lceil n/\alpha \rceil$ et noter $n_1 = \cdots = n_d = \lceil n^{1/d} \rceil$
2. Définir $(i_1, \ldots, i_d)$ la décomposition en base $\lceil n^{1/d} \rceil$ de i
3. Pour $j \in [1..d]$ générer une requête Q_j à partir du protocole cPIR basique pour récupérer un élément d'index d'index i_j dans une base de données de n_j éléments à partir du cryptosystème configuré avec *Enc.Params*
4. Retourner $\mathrm{Q} = (\mathrm{Q}_1, \ldots, \mathrm{Q}_d)$

Génération de la réponse (Serveur) :
Entrée : paramètres PIR (n, ℓ, α, d), paramètres crypto *Enc.Params*, requête $(\mathrm{Q}_1, \ldots, \mathrm{Q}_d)$, éléments de la base de données $(m_1, \ldots, m_n)$
Sortie : Réponse PIR

1. Redéfinir $n = \lceil n/\alpha \rceil$ et noter $n_1 = \cdots = n_d = \lceil n^{1/d} \rceil$
2. Pour $j \in [1..n]$ redéfinir m_j comme l'agrégation des $m_{j\alpha+k}$ pour $k \in [1..\alpha]$
3. Pour $j \in [1..n]$
   - Noter $(j_1, \ldots, j_d)$ la décomposition de j en base $\lceil n^{1/d} \rceil$
   - Définir $m_{(j_1,\ldots,j_d)} = m_j$
4. Pour $j \in [1..d]$
   Pour chaque tuple $(i_{j+1}, \ldots, i_d) \in [1..n_{j+1}] \times \cdots \times [1..n_d]$
   - Calculer, en utilisant l'algorithme de cPIR Basique, une réponse PIR avec $(m_{(1,i_{j+1},\ldots,i_d)}, \ldots, m_{(n_j,i_{j+1},\ldots,i_d)})$ en tant que base de données et Q_j
   - Définir avec cette réponse PIR $m_{(i_{j+1},\ldots,i_d)}$ si $j < d$ ou RES si $j = d$
5. Retourner RES

Extraction de la réponse (Client) :
Entrée : paramètres PIR (n, ℓ, α, d), paramètres crypto *Enc.Params*, c indexe choisi i, réponse PIR
Sortie : Élément à l'indexe i de la base de données

1. Déchiffrer les d couches de chiffrement de la réponse afin d'obtenir α éléments
2. Retourner l'élément d'index i et effacer les autres.

En pratique, l'algorithme récursif nécessite un paramètre supplémentaire d, nommé dans sa version longue *dimension*. Cette approche réduit les n éléments de requête envoyés par le client à $d \times n^{1/d}$ éléments, la taille de la réponse, quant à elle, est d'environ $\mathrm{F}^d \times \ell$. Par exemple, si le facteur d'expansion du cryptosystème utilisé est $\mathrm{F} = 2$, et que la base de données est composée d'un million d'éléments, on peut envoyer une requête de 10^6 éléments et obtenir un élément de la base de données avec un facteur d'expansion de 2 (à noter que $d = 1$, signifie pas de récursion) ; envoyer 2×1000 chiffrés et obtenir la réponse avec un facteur d'expansion 4 ($d = 2$) ; envoyer une requête de 3×100 chiffrés et obtenir la réponse avec un facteur d'expansion de 8 ($d = 3$) ; ainsi de suite.

La justesse du protocole récursif a été prouvée par Julien Stern dans l'article [55]. De fait, ce protocole permet de jouer sur différents paramètres comme le découpage de la base de données, le nombre de récursions et les paramètres des cryptosystèmes utilisés pour différentes récursions. Ces nombreuses possibilités ont été étudiées et généralisées par Helger Lipmaa dans [36].

5.3 Le protocole de XPIR

Une vue d'ensemble du protocole utilisé dans XPIR est présentée ci-dessous. L'ensemble des paramètres PIR est composé de l'agrégation et du nombre de récursions d. Lorsque les paramètres du PIR et cryptographiques sont envoyés par le serveur, le mode de fonctionnement est nommé *driven-mode*. Dans ce mode, seulement les étapes 1 (saut conditionnel dans le mode *server-driven*), 4 (choix de l'élément par le client) et 5 (récupération de l'élément)

du protocole sont exécutées. Inversement, dans le mode *client-driven*, le client effectue une batterie de tests afin de déterminer les meilleurs paramètres en fonction de ses contraintes (étapes 2-3), puis les utilise (étapes 4-5). Par défaut, nous supposons que les paramètres optimaux ont déjà été trouvés ; le mode utilisé est le *server-driven*. De plus, l'optimisation se fonde sur des valeurs conservées dans des fichiers, appelés caches, qui doivent être générés lors de la première utilisation. Cette opération, qui prend plusieurs minutes, pourrait être ennuyeuse pour un utilisateur de XPIR désirant tester rapidement l'application.

Protocole PIR client-serveur (Vue d'ensemble)

Entrée : Intervalle de récursion (d_1, d_2), Agrégation(α_1, α_2), liste des paramètres de chiffrement *EncParams*, bande passante montante et descendante (U, D), fonction d'optimisation f_{target}, index de l'élément à télécharger i (pot. indéfini), booléen *serverDriven*
Sortie : L'élément de la base de données d'index i

1. Si serverDriven est vrai
   - Serveur : Envoie les paramètres nécessaires au client
   - Client : Vérifie si les paramètres donnés sont suffisamment sécurisés
   - Sauter à l'étape 4
2. Si les résultats du test de performance ne sont pas disponibles pour tous les paramètres de *EncParams*
   - Client et Serveur : Éxécutent l'algorithme de mise en cache des Performances
   → Résultats de performances
3. Si plusieurs paramètres de PIR et de chiffrement sont disponibles pour les contraintes données
   - Client et Serveur : Éxécutent l'algorithme d'Optimisation
   Sinon ils utilisent le seul choix disponible
   → Paramètres optimaux pour ces paramètres et contraintes données
4. Si i est indéfini
   - Client et Serveur : Éxécutent l'algorithme de sélection
   → Sélection de l'index i
5. Si l'algorithme de chiffrement sélectionné est *no cryptography*
   - Client : Télécharge la base de données, conserve l'élément d'index i
   Sinon
   - Client : Éxécute l'algo. de gen. de requête et envoie le résultat
   - Serveur : Éxécute l'algo. de gén. de la réponse et envoie le résultat
   - Client : Éxécute l'algo. de ext. de la réponse et retourne le résultat
   → Élément sélectionné de la base de données.

Cette description de l'algorithme client-serveur étant générale, il nous faut, pour être complet, décrire les algorithmes appelés.

XPIR contient un ensemble de paramètres cryptographiques prédéterminés :

— *no cryptography* qui représente le PIR trivial dans lequel l'intégralité de la base de données est télévisionchargée ;
— Le cryptosystème de Paillier ;
— Le cryptosystème Ring-LWE.

Chaque cryptosystème est fourni avec une liste définie de paramètres dont la sécurité varie de 80 à 256 bits. Pour le chiffrement de Paillier, nous avons suivi les recommandations du NIST et pour les cryptosystèmes fondés sur les réseaux nous avons suivi les résultats de l'article [35].

Par défaut, la fonction objectif de l'optimisation est la durée nécessaire à l'utilisateur pour récupérer un élément de la base de données, ce qui est donné (en utilisant des noms de variables explicites) :

$$\text{MAX(tempsGénérationRequête, tempsEnvoiRequête)} + \text{MAX(tempsGénérationRéponse, tempsEnvoiRéponse, tempsDéchiffrementRéponse)}$$

Cette formule s'explique par le fait que la génération de la requête et son envoi sont pipelinés. C'est-à-dire que les parties de la requête (les chiffrés de 0 et le chiffré de 1) sont envoyées directement après avoir été calculées. Le programme n'attend pas d'avoir généré l'intégralité

de la requête avant de l'envoyer. Le comportement est similaire pour la génération, l'envoi, et l'extraction de la réponse. Dès que le premier morceau de la réponse est généré, il est envoyé au client qui, une fois réceptionné, commence à le déchiffrer. Nous proposons d'autres fonctions objectifs :

— Les ressources minimums (qui utilise la somme des 5 variables citées précédemment) ;
— Le coût *cloud* (qui est un équivalent pondéré donnant le montant en dollars pour chaque milliseconde de CPU utilisé ainsi que pour chaque bit transmis et donne le coût total de l'opération).

Précisons que la fonction objectif se choisit directement en ligne de commandes.

5.3.1 Cryptosystème fondé sur Ring-LWE

Notre contribution dans le domaine des performances est concentrée en deux points : une méthode efficace de représentation NTT-CRT et des transformations associées ; l'usage de quotient de Newton pour les éléments de la requête. Pour plus d'information sur le cryptosystème, nous invitons le lecteur à se référer à la section 2.4.2 du chapitre 2. L'idée vient de constat que les coûts calculatoires les plus importants de la génération de la requête, de la réponse et de l'extraction de la réponse proviennent de la multiplication polynomiale. Concrètement, pour la génération de la réponse, les coûts calculatoires sont majorés par l'opération d'absorption du protocole cPIR décrit plus haut. À partir de notre cryptosystème fondé sur Ring-LWE, cette phase peut être modélisée comme suit :

Pour j de 1 à $\lceil \ell/\ell_0 \rceil$
  Calculer $\mathrm{R}_j = (\sum_{i=1}^{n} m_{i,j} * q_{i,1}, \sum_{i=1}^{n} m_{i,j} * q_{i,2})$

Avec $q_{i,1}, q_{i,2}$ les deux polynômes qui forment un élément de la requête et toutes les sommes et produits dans l'anneau R_q (les polynômes sont réduits modulo $\mathrm{X}^{\mathrm{N}} + 1$ et les coefficients modulo q).

La représentation sous forme de NTT-CRT fait passer le coût calculatoire de la multiplication de deux polynômes de $\mathrm{O}(\mathrm{N}^2 \times \log q)$ à $\mathrm{O}(\mathrm{N} \times \log q)$. Comme nous l'avons écrit dans le chapitre 2 cette représentation n'est pas en soi une nouveauté.

Néanmoins, le précalcul de coefficients de Newton appliqué à la cryptographie sur les réseaux euclidiens nous semble novateur. Lorsque deux polynômes sont multipliés, le produit scalaire associé doit être réduit modulo q ce qui augmente considérablement le coût calculatoire de cette opération. Dans l'algorithme de génération de la réponse, les produits sont toujours de la forme « des données transformées de polynômes » fois « un élément de requête». Ainsi les éléments de requête sont utilisés dans de nombreuses multiplications. Les quotients de Newton précalculés pour les multiplicandes réutilisés (comme les scalaires de la requête) permettent de substituer la formule « multiplier et diviser par le module » à une multiplication modulaire spécifique composée de deux multiplications d'entiers et une soustraction conditionnelle (éliminant de fait, la division coûteuse).

5.3.2 La représentation NTT-CRT et transformations

Paramètres	$(1024, 60)$	$(2048, 120)$	$(4096, 120)$
Taille : entrée (par poly.)	20kbit	100kbit	192kbit
Prétraitement (par poly)	4.2us	19us	38us
Prétraitement (PIR tput)	4.8Gbit/s	5.2Gbit/s	5Gbit/s
Traitement (par poly)	0.57us	2.3us	4.8us
Traitement (PIR débit)	18Gbit/s	22Gbit/s	20Gbit/s

Table 5.1 – Prétraitement PIR, le traitement, et le débit sur un ordinateur portable MSI avec un Core i7-3630QM 2.64 GHz, pour différents paramètres cryptographiques. Les tailles en entrée sont les maximums donnés dans la figure 5.1. Le prétraitement d'un polynôme correspond aux transformations NTT et CRT (opérations les plus importantes lors de l'importation d'une base de données). La transformation inverse donne des résultats similaires. Le traitement correspond à une multiplication et une addition fusionnées (FMA) (l'opération la plus importante lors de la génération de la réponse). Nous utilisons les même transferts de mémoire que dans notre protocole PIR : pour un thread donné, seule une opérande varie la majorité du temps. Le débit est obtenu par rapport aux entrées : dans le prétraitement pour chaque polynôme (Taille entrée) bits sont traités ; dans traitement deux polynômes doivent être traités pour correspondre aux (Taille entrée) bits.

Dans XPIR, nous utilisons une représentation mixte NTT-CRT afin de réduire les coûts calculatoires : Number Theoretic Transform pour les polynômes [17] et le Théorème des Restes chinois (CRT) pour les entiers. Nous appelons la partie de XPIR qui gère les transformations et effectue les calculs dans cette représentation NFLlib. À titre de comparaison, la bibliothèque de chiffrement homomorphe de Halevi et Shoup [22] met en œuvre le cryptosystème de Brakerski, Gentry et Vaikuntanathan. Ils fournissent un objet nommé DoubleCRT qui propose une représentation NTT et CRT des polynômes proche de NFLlib.

Une description plus détaillée de la NTT et du CRT est présentée dans les chapitres 1 et 2. Dans la suite de cette section, nous allons nous concentrer sur l'impact de leur utilisation. L'utilisation de la représentation sous forme de NTT offre la possibilité de réaliser les multiplications de polynômes avec un coût linéaire (amorti) en N au lieu du coût quadratique de l'algorithme trivial. Transformer un polynôme dans sa forme NTT ainsi que l'opération inverse se fait en un temps quasi linéaire (en $O(N \log N)$). La représentation CRT assure que le coût des multiplications et aussi linéaire en $\log p$, à la place de l'algorithme trivial en temps quadratique. La figure 5.2 expose les performances de précalcules qui correspondent à l'importation des données sous forme de polynômes NTT-CRT et au traitement, qui lui correspond à une multiplication et une addition fusionnées [2]. Les opérations de séparations des données et de CRT (si nécessaire) sont rapides. Ainsi le principal goulet d'étranglement dans la partie de prétraitement est le calcul de la NTT dans notre anneau de polynômes.

2. Le terme anglais est plus connu : *fused mutiply and add (FMA)*.

Paramètres	$(1024, 60\|\|44)$	$(2048, 120\|\|132)$
Prétraitement (DoubleCRT)	178us	1100us
Prétraitement (NTTTools)	16us	78us
Traitement (DoubleCRT)	5us	27us
Traitement (NTTTools)	2.3us	9.6us

Table 5.2 – Durées du prétraitement (NTT et CRT) et du traitement (multiplication et additions) avec DoubleCRT et NFLlib. La taille des modules doit être un multiple de 44 dans DoubleCRT (pour faire des réductions modulaires sur des doubles flottants). Nous avons choisi des tailles de modules les plus proches possible. Les tests ont été réalisés en monoprocesseurs, car les objets DoubleCRT donnent des erreurs de segmentation sur l'ordinateur portable MSI avec un Core i7-3630QM 2.64 GHz. Le prétraitement est plus rapide sur NFLlib ($\times 10$), sur le traitement l'écart est moins important (entre $\times 2$ et $\times 3$).

Ces valeurs ont un impact évident dans la génération de la réponse cPIR. Le prétraitement correspond à l'importation des éléments de la base de données et le traitement à la génération de la réponse. Lorsque le serveur est lancé, les élément de la base de données peuvent être importés dans la RAM sous forme de polynômes NTT-CRT à environ 5Gbit/s. Suite à l'importation, la base de données est traitée durant la de génération de la réponse à environ 20Gbit/s. Si les données deviennent rapidement obsolètes (telles qu'un flux de télévision sur IP, ou des valeurs boursières) le goulet d'étranglement principal est la transformation des données sous leur forme NTT-CRT. La phase de traitement se trouve limitée à 5Gbit/s. À des fins comparatives, utiliser un cPIR fondé sur le cryptosystème de Paillier sur la même configuration matériel donne une vitesse de traitement de la base de données de 1Mb/s (pour un module de 2048 bits et 112 bits de sécurité). De nouveau, à des fins comparatives, dans un PIR trivial (télécharger l'intégralité de la base de données) traiter un bit revient à l'envoyer au client et donc, la base de données est traitée à la vitesse du réseau disponible (p. ex. 100Mb/s pour une ligne FTTH [3]). Ainsi le PIR dans sa version trivial sera de manière générale plus rapide que le cPIR fondé sur le cryptosystème de Pailier, mais plus lent que notre implémentation fondée sur Ring-LWE ce qui correspond aux résultats de Sion et Cabunar [53].

L'objet DoubleCRT présenté dans [22] est plus élaboré que ce que nous proposons dans NFLlib. Et plus précisément, il contient des fonctions nécessaires aux chiffrements entièrement homomorphes que nous n'avons pas écrites. De plus, DoubleCRT permet de fixer arbitrairement le degré des polynômes alors que les degrés des polynômes utilisés dans NFLlib doivent être une puissance de deux.

D'un autre côté, la relative simplicité de nos paramètres nous a conduits à faire des choix intéressants. Premièrement, nous utilisons l'algorithme de NTT proposé par Harvey [25] qui est très efficace, mais uniquement fonctionnel sur les polynômes dont le degré est une puissance de deux. Nous avons défini les premiers qui forment potentiellement les modules ce qui produit des optimisations lors de la compilation.

Les fonctionnements de HElib implique que l'utilisateur définisse ses besoins en termes d'opérations homomorphes, puis, à partir de ces entrées, une routine construit un contexte FHE. L'utilisateur ne peut pas choisir directement d'utiliser un ou deux premiers, nous avons modifié très légèrement le code afin d'effectuer des tests comparatifs.

Une amélioration des performances des multiplications de polynômes avec NFLlib est a noter (entre $\times 2$ et $\times 3$), comme montré sur le Tableau 5.2, de plus, l'écart augmente avec le prétraitement ($\times 10$). Ces résultats s'expliquent par le fait d'avoir réduit le degré des polynômes à des puissances de deux par l'utilisation d'algorithmes comme celui d'Harvey pour la NTT.

3. Connexion fibre optique

Pour terminer, l'utilisation de la mémoire est plus faible avec NFLlib car les paramètres et la bibliothèque sont plus simples. Pour des polynômes de degré 1024 composés de coefficients de 60 bits, l'empreinte mémoire de NFLlib et de 8 Koctets par défaut et du double avec les coefficients de Newton précalculés. L'empreinte mémoire de DoubleCRT est plus difficile à évaluer, car certains objets (tels que le FHE context) sont partagés, mais on remarque qu'a chaque nouvel objet DoubleCRT, 40 Koctets supplémentaires sont utilisés.

NFLlib ainsi que le cryptosystème que nous avons développé pourraient être un remplaçant efficace à l'objet DoubleCRT pour les utilisateurs qui souhaite manipuler des polynômes sur les réseaux d'idéaux ou pour des opérations homomorphes basiques. Pour des opérations plus complexes, DoubleCRT reste la référence.

5.3.3 Coefficients de Newton précalculés

Notre approche du calcul de la réduction modulaire a été traitée dans le Chapitre 4. Nous ferons un court rappel puis nous nous concentrerons sur la comparaison entre NFLlib et Helib.

Le principe de notre algorithme de réduction modulaire est de précalculer les quotients de Newton des opérandes utilisés plusieurs fois. Ainsi pour un module p donné, nous calculons pour un y donné, une approximation de y/p (< 1) dans un entier de 128 bits. Nous multiplons d'abord y par 2^{64} et on effectue une division entière par p. Ce qui donne y', les premiers 64 bits de la division de $y/p * 2^{64}$. De cette manière, l'opération coûteuse est effectuée une seule fois. Lors que nous calculons $xy \mod p$, nous faisons appel à un algorithme qui prend en entrée x, y, y'.

NFLlib fournis les fonctions pour précalculer les coefficients nécessaires aux opérations arithmétiques sur les polynômes. Puisque cet algorithme nécessite seulement deux multiplications d'entiers, un décalage et une soustraction conditionnelle, la multiplication de deux polynômes se fait en $2n$ multiplications plutôt que n multiplications modulaires de l'algorithme naïf.

L'intérêt de cet algorithme réside dans le nombre de fois ou les mêmes opérandes sont réutilisés afin d'amortir le coût de calcul des coefficients de Newton. Le système de chiffrement que nous avons construit à partir de NFLlib, exécute seulement deux multiplications polynomiales pour chiffrer et une seule multiplication polynomiale pour déchiffrer en utilisant toujours deux multiplicandes : les clés secrètes et publiques. Dans le protocole cPIR, la réponse est obtenue en multipliant constamment les morceaux des éléments de la base de données avec des éléments de requête. Ceux deux cas utilisent ce mécanisme de précalcul. En pratique, toutes les multiplications présentes dans le code de XPIR utilisent ce mécanisme, car ce dernier est très rapidement amorti.

Paramètres	$(1024, 60\|\|44)$	$(2048, 120\|\|132)$
`DoubleCRT`	3.4us	20.2us
`NTTTools-noQuotients`	29us	115us
`NTTTools`	1.8us	7.3us

Table 5.3 – Durée d'une multiplication en microsecondes pour différents paramètres (voir Table 5.2). Les tests ont été réalisés en monoprocesseurs sur l'ordinateur portable MSI avec un Core i7-3630QM 2.64 GHz. À noter que les résultats de DoubleCRT sont bien meilleurs sans le précalcule des quotients, car DoubleCRT utilise des modules de 44 bits afin de faire des opérations flottantes et des réductions modulaires fondées sur des arrondis. Le précalcule des quotients donne de $\times 2$ à $\times 3$ meilleures performances. Sans le précalcule les performances de NFLlib serait très mauvaise par rapport à HElib car la bibliothèque ne fait pas d'opérations sur les flottants, néanmoins notre protocole cPIR est conçu pour fonctionner avec les coefficients de Newton.

5.3.4 Performances de chiffrement et de déchiffrement

Rappelons que le cryptosystème que nous proposons dans cette section est équivalent en matière de sécurité au cryptosystème décrit dans la section 5.2.1.

Le principe général est le suivant : les polynômes qui servent à décrire les entrées des différents algorithmes (clé secrète, aléa, messages) subissent un prétraitement qui consiste à les passer en représentation NTT-CRT. Ainsi toutes les opérations entre le début du chiffrement et la fin du déchiffrement se font coordonnée à coordonnée ce qui contribue aux bonnes performances du cryptosystème.

Nous avons adapté les algorithmes pour la représentation NTT-CRT des polynômes sur deux points. Premièrement, lors de l'utilisation de polynômes dont les coefficients ont été tirés suivant une distribution aléatoire uniforme, il n'est pas nécessaire de changer leur représentation. En effet, la NTT, le CRT et leurs inverses respectifs sont des fonctions bijectives qui associent un ensemble fini à lui même. Donc, ces fonctions permutent des éléments au sein de leur domaine de définition. Par conséquent, prendre un élément uniforme et changer sa représentation sous forme NTT-CRT est équivalent à prendre directement un élément uniforme. Deuxièmement, pour tous les produits présents dans le protocole cPIR, l'un des deux termes est réutilisé à plusieurs reprises. Par conséquent, il est toujours possible de faire appel à la multiplication modulaire rapide qui utilise les quotients de Newton précalculés.

À partir de ces deux propriétés, la fonction de chiffrement nécessite seulement trois transformations NTT-CRT et quelques opérations de base. À première vue, l'opération la plus coûteuse dans le déchiffrement est l'opération de NTT inverse. Ce qui est le cas avec des modules de 60 bits ou moins. Cependant, si plusieurs modules sont employés (sous la forme CRT), il est nécessaire de reconstituer chacune des coordonnées sous sa représentation non-CRT afin de séparer le bruit du message (sous forme CRT, il n'y a pas de division euclidienne simple). Cette opération, appelée relèvement, revient à multiplier les éléments du CRT par ses coefficients de relèvement (voir section 1.6 du chapitre 1 pour plus de détails). Le relèvement se fait sans réduction modulaire et requiert de l'ordre de $(\log_2 q)/60$ multiplications d'éléments de 60 bits avec des éléments de $\log_2 q$ bits éléments. Puisque NFLlib utilise uniquement des types natifs nous faisons appel à une bibliothèque capable de gérer des entiers de taille arbitraire. En pratique et lorsqu'une telle bibliothèque est utilisée, le coût du déchiffrement est multiplié par un facteur au moins 10 pour gérer ces multiplications. La figure 5.2 montre clairement cette évolution.

La bibliothèque GMP est uniquement appelée pour le relèvement sur les objets NTTLWE (fonction `poly2mpz` de NFLlib). En pratique, ces résultats résultent en une perte de performance majeure. Néanmoins, pour un module de 60 bits, les performances sont très élevées. Le cryptosystème est capable de produire des requêtes à 2.2Gbit/s et de déchiffrer des réponses à 5Gbit/s. De plus, ce débit est indépendant de la taille des polynômes, car le coût du chiffrement et de déchiffrement augmente linéairement, mais la taille des chiffrés et des clairs aussi. Un ordinateur portable récent peut générer et envoyer la requête ainsi que déchiffrer la réponse à la vitesse maximum de la bande passante de sa connexion réseaux en utilisant un seul processeur. Avec un module de 120 bits, le chiffrement s'adapte bien et il est possible de générer des requêtes à 2.5Gbit/s, cependant le déchiffrement souffre du surcoût engendré par le relèvement du CRT et les réponses peuvent «seulement» être traitées à 710Mb/s.

Le déchiffrement est seulement un facteur limitant pour de très gros modules (p. ex. 480 bits) ou si le client et le serveur sont connectés par une ligne gigabit et que la réponse est générée très rapidement (par exemple pour une base ayant très peu d'éléments). En pratique, si l'optimiseur de paramètres choisit en général de petits modules (120 bits au plus) c'est à cause du coût de chiffrement. En effet, pour des modules dont la taille est au-delà de 120 bits, l'augmentation dans le coût de chiffrement, même faible, ajoute plus de temps à l'aller-retour (ou au total des ressources dépensées) que ce qui est gagné en facteur d'expansion ou en vitesse de traitement de la réponse.

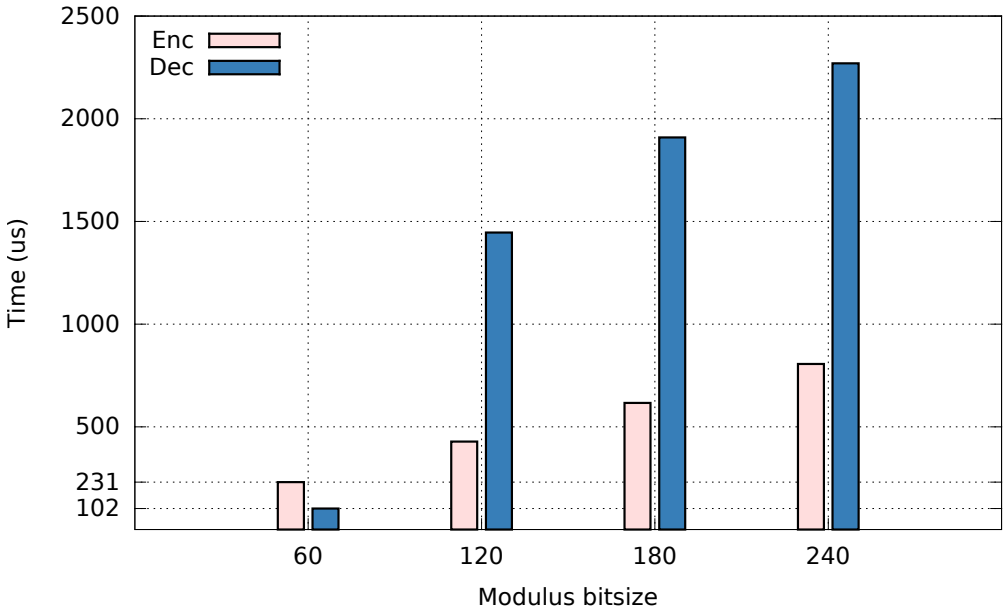


Figure 5.2 – Durées de chiffrement et de déchiffrement pour des polynômes de degré 4096 et différentes tailles de module sur un ordinateur portable MSI GT60 avec un Core i7-3630QM 2.67GHz. Le coût calculatoire du chiffrement augmente linéairement en fonction de la taille du module, mais aussi de la taille du texte clair et chiffré associé. La différence marquée dans le déchiffrement provient de l'utilisation de GMP pour les modules strictement supérieurs à 60 bits.

5.3.5 Achitecture et interface de programmation

L'architecture de XPIR, présentée dans la figure 5.3, peut être vue comme une superposition de couches qui interagissent entre elles. La couche de plus bas niveau est celle composée par les cryposystèmes et leurs bibliothèques (GMP et NFLlib) interfacées avec le reste de XPIR par les `Homomorphic Crypto Interfaces`. Au dessus des cryptosystèmes se situent deux blocs différents. La partie cliente est composée du générateur de requête `PIRQueryGenerator` et de l'extracteur de réponse `PIRReplyExtractor`. Le tout est orchestré par `PIRClient` qui gère principalement les communications réseau et les différents modes de fonctionnement. Le serveur peut utiliser deux générateurs de réponses différents. Le premier, nommé `PIRReplyGeneratorGMP`, peut générer la réponse PIR en utilisant tout cryptosystème (homomorphe) fondé sur GMP (comme Paillier) . Le second, `PIRReplyGeneratorNFL` peut générer la réponse PIR en utilisant tout cryptosystème (homomorphe) fondé sur NFLlib. Le serveur est géré par le couple `PIRServer-PIRSession`. L'objet `PIRServer` gère les nouveaux clients pour lesquelles il construit une `PIRSession`.

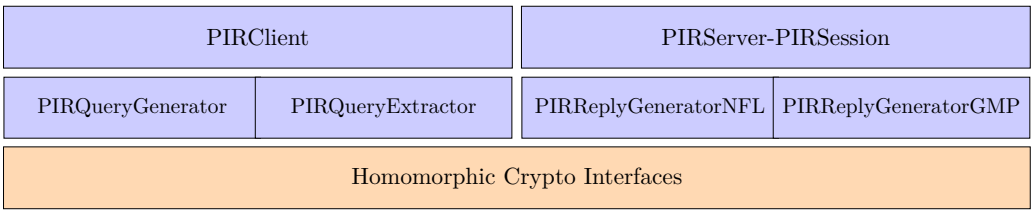


Figure 5.3 – Représentation de l'architecture de XPIR avec, dans la partie gauche le client et dans la partie droite le serveur. À noter que les classes concernant les cryptosystèmes sont partagées entre `PIRClient` et `PIRServer`.

L'optimiseur de paramètres n'est pas séparé en deux parties, il s'exécute en fonction du contexte comme représenté par la figure 5.4. Si lancé sur le client il s'occupera d'optimiser les paramètres dépendant de la génération de la requête, de l'extraction de la réponse,

de générer les caches correspondants et de communiquer avec le serveur. Si lancé sur le serveur, ce dernier s'occupera de générer les caches relatifs à la génération de la réponse et de communiquer avec le client. Afin de différencier le client et le serveur, ces derniers communiquent avec l'optimiseur par des interfaces nommées Optimization Interfaces.

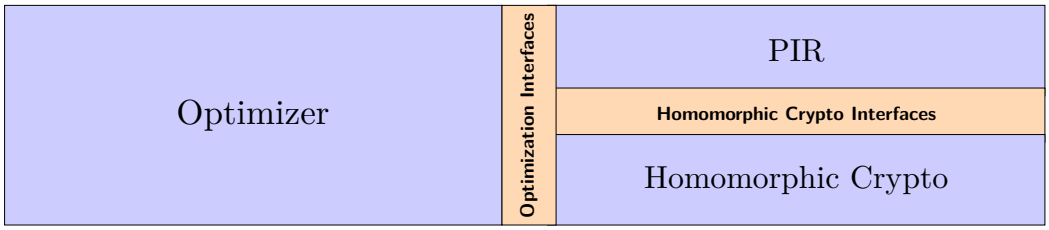


Figure 5.4 – Représentation de l'architecture de l'optimiseur de paramètres relativement à client et au serveur.

Nous avons ajouté à XPIR, en plus du couple client/serveur et de l'optimiseur de paramètres la possibilité de manipuler les fonctions de XPIR par le biais d'une interface de programmation. L'objectif de cette interface de programmation (API) est de donner accès aux programmeurs aux algorithmes de cPIR en faisant abstraction de leur fonctionnement. Le cPIR pour ainsi être aisément utilisé comme brique logicielle dans d'autres programmes.

L'API[4] de XPIR est composée de 5 objets `c++`, dont trois pour les algorithmes de cPIR. L'appel à l'optimiseur est, quant à lui, effectué automatiquement par les objets.

- `imported_database` représente la base de données utilisée lors de la génération de la réponse ;
- `PIRQueryGenerator` initialise le protocole cPIR en générant la requête en fonction d'un index donné en argument ;
- `PIRReplyGenerator` représente la partie serveur, il va traiter la requête et produire la réponse correspondant ;
- `PIRReplyExtraction` va extraire l'élément choisi dans la réponse générée par le serveur.
- `HomomorphicCryptoFactory` crée des objets cryptographiques génériques utilisés par les trois objets précédents (actuellement, seul Ring-LWE est disponible par l'API).

Les trois fonctions de cPIR prennent en paramètre un objet `pirParameters` représentant les paramètres du PIR ainsi qu'un objet cryptographique générique.

Algorithm 18 Exemple simplifié d'utilisation de l'API de XPIR

```
1  PIRQueryGenerator q_generator(params, crypto);
2  q_generator.generateQuery(chosen_element);
3
4  /*Transfert des données entre le gen. de requêtes
5   * et la générateur de réponses */
6
7  PIRReplyGenerator r_generator(params, crypto, db);
8  r_generator.generateReply();
9
10 /*Transfert des données entre le gen. de réponse
11  * et l'extracteur de réponses */
12
13 PIRReplyExtraction r_extractor(params, crypto);
14 r_extractor.extractReply(clientside_maxFileBytesize);
```

4. Pour retrouver directement les fonctions disponibles de l'API se référer au fichier source `libpir.hpp`.

L'algorithme 18 est un exemple d'utilisation des objets proposés dans l'API de XPIR. Les lignes 4-5 et 10-11 représentent le transfert de la requête pour les premières et le transfert de la réponse pour les secondes. Ces transferts peuvent être simulés à titre expérimental, en déplaçant des données d'une mémoire tampon à une autre. Bien évidemment, le transfert des données par le réseau est toujours possible.

5.4 Performance et cas d'utilisations

Dans cette section, nous analysons les performances de XPIR suivant deux métriques : la latence (*latency*) et le débit perçu par l'utilisateur (*user-perceived throughput*). La mesure de latence est la durée d'aller-retour qui débute au moment où l'utilisateur commence à générer sa requête cPIR et qui termine au moment où il finit de déchiffrer la réponse reçue. Le débit perçu par l'utilisateur est le débit mesuré en bits par secondes auquel l'utilisateur est capable de récupérer l'élément choisi après déchiffrement.

Nous examinerons de façon détaillée et critique deux types de configurations de base de données :

— bases de données statiques pour lesquelles leurs éléments subissent un prétraitement ;
— bases de données dynamiques dont les éléments sont éphémères (flux TV sur IP, cours de bourse, données de capteurs, ...), et ne peuvent pas être prétraités.

Le prétraitement est appliqué indépendamment à chaque élément d'une base à une vitesse variant de 5Gbit/s (pour un processeur d'ordinateur portable haut de gamme) à 10Gbit/s (pour un processeur de serveur) comme montré dans la section 5.3.2. Une base de données est considérée comme statique si la durée de vie de ses éléments est bien supérieure à la durée de leur prétraitement (e.g 1-2 secondes pour un film de 10 Gbit - 1.25 Go) et que les éléments sont connus suffisamment tôt pour être prétraités avant leur première utilisation dans le protocole cPIR.

Dans le but d'illustrer la versatilité de notre bibliothèque, nous présentons les résultats de performances obtenues par quatre cas d'utilisation combinant des configurations statiques ou dynamiques et en mesurant débit ou latence. Nous illustrons les applications pour lesquelles un débit élevé est la principale contrainte, avec un serveur ayant une configuration de type Netflix (les données peuvent être considérées comme statiques) et avec un sniffeur réseau qui cache son filtre (données dynamiques). Pour les applications dont la contrainte est une faible latence, nous utilisons une configuration semblable à Match.com (base de données relativement statique de rencontre en ligne) et un service d'informations de données de marché boursier (données dynamiques). À noter que trouver la meilleure application d'un protocole cPIR rapide est en dehors du champ de cette thèse. Les cas d'utilisations présentés ont été choisis, car leurs paramètres éprouvent le protocole cPIR, et non parce qu'ils sont ou semblent être le meilleur moyen de résoudre les problèmes de vie privée dans ces configurations. Notre but et de montrer que le protocole cPIR est plus performant qu'un PIR trivial et qu'il est possible de l'employer sur des bases de données de très grande taille avec de fortes contraintes sur le débit effectif obtenu chez le client.

5.4.1 Paramètres expérimentaux

Pour montrer que XPIR est vraiment utilisable par «tout le monde» dans de nombreuses applications, nous employons du matériel facile à trouver dans le commerce dans presque toutes les configurations. Le serveur cPIR s'exécute sur un ordinateur portable MSI GT60 avec un processeur Core i7-3630QM et 8Go de RAM DDR3. Le périphérique de stockage a une importance particulière, car XPIR traite le contenu de la base de données, stocké sur ledit support, très rapidement. Ainsi le périphérique de stockage influence directement les performances. Dans nos évaluations, les données sont stockées sur deux médias différents : la RAM dont les accès atteignent les 100Gbit/s, ou un SSD [5] OCZ Vertex 460 dont les accès atteignent 4Gbit/s. La vitesse de lecture de données contigües du SSD est suffisante pour approvisionner le serveur dans toutes les configurations de bases de données dynamiques.

5. *Solid State Drive* : est un matériel informatique de stockage de données sur mémoire flash.

Si les données sont statiques, le traitement des données est bien plus rapide que la vitesse de lecture contigüe d'un SSD. À première vue, placer les données dans la mémoire RAM suffit à résoudre ce problème. Néanmoins, dans des cas comme celui de Netflix, la quantité de données est énorme et ne peut pas loger entièrement dans la RAM. Nous discutons de ce problème par la suite. Les lignes FTTH et ADSL ont été simulées au moyen de temporisateurs appropriés sur le client et sur le serveur qui sont physiquement reliés par une ligne gigabit.

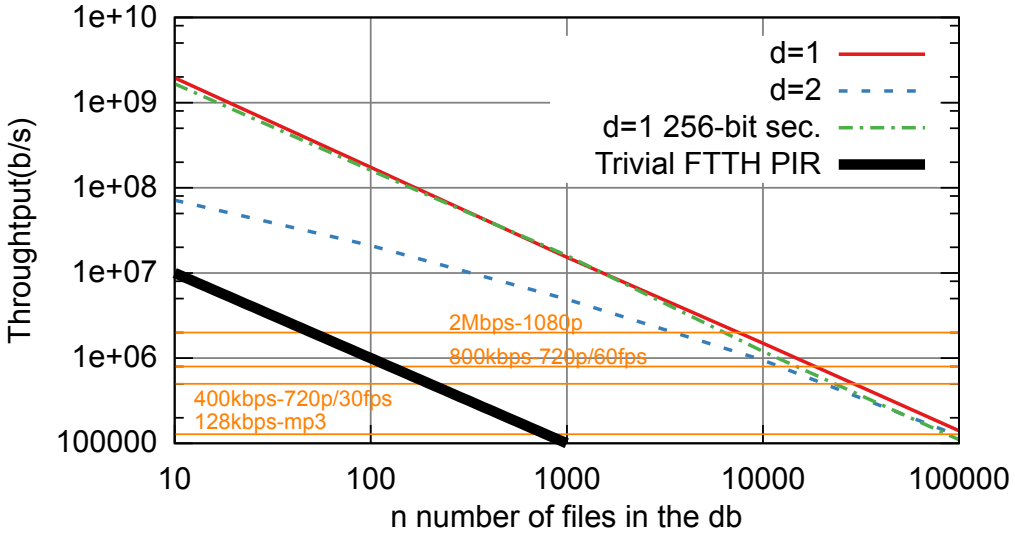
Dans la majorité de nos mesures de performances, l'optimiseur a trouvé que les meilleurs paramètres pour le cryptosystème Ring-LWE étaient $(2048, 120)$ où $(1024, 60)$. Selon la méthode de génération de paramètres présentés dans la Section 5.2.1, le premier jeu de paramètres entraine 91 bits de sécurité et le second 81. Nous rappelons que les bits aléatoires sont générés à partir de Salsa20/20 [3] dont le niveau de sécurité peut atteindre les 256 bits. Le niveau de sécurité de notre cryptosystème fondé sur le problème Ring-LWE est ainsi limité à 256 bits.

Augmenter le niveau de sécurité d'un jeu de paramètres dans cryptosystèmes fondés sur les réseaux euclidiens a généralement un faible coût. En effet, pour des modules de taille constante, la sécurité (le nombre d'opérations à effectuer par un attaquant) augmente exponentiellement avec le degré des polynômes et le coût calculatoire augmente (pour l'utilisateur) presque linéairement. Par exemple, si nous utilisions les paramètres $(4096, 120)$, la sécurité théorique attendrait les 335 bits. Néanmoins, le générateur d'aléa limite notre cryptosystème à 256 bits de sécurité. Avec un tel niveau de sécurité, la génération de la requête, le prétraitement de la réponse, la génération de la réponse et son déchiffrement ont un coût seulement multiplié par un facteur 2 (2.18 pour le prétraitement et 2 pour le reste). Avec de tels paramètres, les chiffrés peuvent contenir bien plus de données (presque le double) et de fait le niveau de sécurité augmente pour un très faible coût. Nous présenterons les coûts avec le niveau de sécurité maximum $(4096, 120)$, puis nous laisserons l'optimiseur choisir les meilleurs paramètres avec un niveau de sécurité minimum placé à 91 bits afin de tester le couple $(2048, 120)$ qui est, comme nous le verrons, un bon compromis entre la taille des chiffrés, le débit de génération de la réponse et le niveau de sécurité.

Les résultats donnés dans cette section représentent l'exécution d'un protocole cPIR avec le cryptosustème Ring-LWE et un protocole PIR trivial afin de montrer l'intérêt du cPIR par rapport au téléchargement d'une base de données complète. Le chiffrement de Paillier donne toujours les plus mauvais résultats et n'apparait donc pas dans les différentes figures.

5.4.2 Débit de traitement sur les données statiques

Figure 5.5 – Débit utile perçu par un utilisateur de XPIR qui reçoit une réponse PIR générée à partir d'un ordinateur portable MSI GT60. Un PIR trivial sur une ligne FTTH (ligne noire épaisse) est entre 10 et 200 fois plus lent que XPIR avec le chiffrement basé sur Ring-LWE. Les lignes rouges (91 bits de sécurité) et pointillés verts (256 bits de sécurité) donnent le débit utile lorsqu'il n'y a pas de récursion (dimension $d = 1$). La ligne en pointillés bleus donne le débit pour $d = 2$. Les lignes horizontales correspondent au débit nécessaire pour regarder un film en 1080p (2Mb/s), 720p 60ips (800Kbp), 720p 30ips (400Kbps) ou écouter un fichier audio mp3 (128Kbps). Les performances obtenues sur un serveur équipé d'un meilleur processeur (par ex. un Intel Xeon 10 coeurs E7-4870) sont doublées et atteignent un nouveau seuil produit par la saturation de la bande passante de la RAM.



Les applications qui nécessitent un débit élevé n'ont de sens que si la taille des éléments de la base de données est élevée (p. ex. plusieurs minutes de téléchargement), si les éléments sont rapidement envoyés nous considérons que le facteur le plus important est la latence (sujet traité dans la section 5.4.5). Ainsi, nous nous concentrons uniquement sur les bases de données dont la taille des fichiers est de 10Mbit et plus. Nos résultats expérimentaux montrent que le débit perçu par l'utilisateur est indépendant de la taille des fichiers lorsque ces derniers se trouvent dans l'intervalle donné.

La figure 5.5 représente le débit perçu par l'utilisateur (le contexte expérimental est donné dans la légende). La ligne rouge représente les performances sans récursion ($d = 1$), c'est-à-dire que la taille de la requête est proportionnelle aux n éléments de la base de données. Cette ligne a été obtenue à partir des paramètres (donnés par l'optimiseur) les mieux adaptés au débit utile :

— pas de récursion ;
— pas d'agrégations des éléments ;
— cryptosystème Ring-LWE avec les paramètres $(2048, 120)$.

Ce paramétrage donne des chiffrés, et donc des éléments de requête, de 500kbit, dont le facteur d'expansion est $\simeq 5$. Ainsi, afin d'obtenir un débit utile par l'utilisateur de 2Mb/s, il s'avère que 10Mb/s de bande passante devront être utilisés. Comme écrit plus tôt, ce paramétrage est le plus favorable au débit utile. Cependant, la taille de la requête peut devenir un problème lorsque le nombre d'éléments n de la base de données augmente, comme nous pouvons l'observer sur la figure 5.6. Mentionnons le fait que cette ligne est particulièrement proche de la ligne droite définie par $15/n$ Gbit/s et plus précisément la ligne glisse lentement de $19/n$ Gbit/s à $14/n$ Gbit/s par secondes pour de grands n.

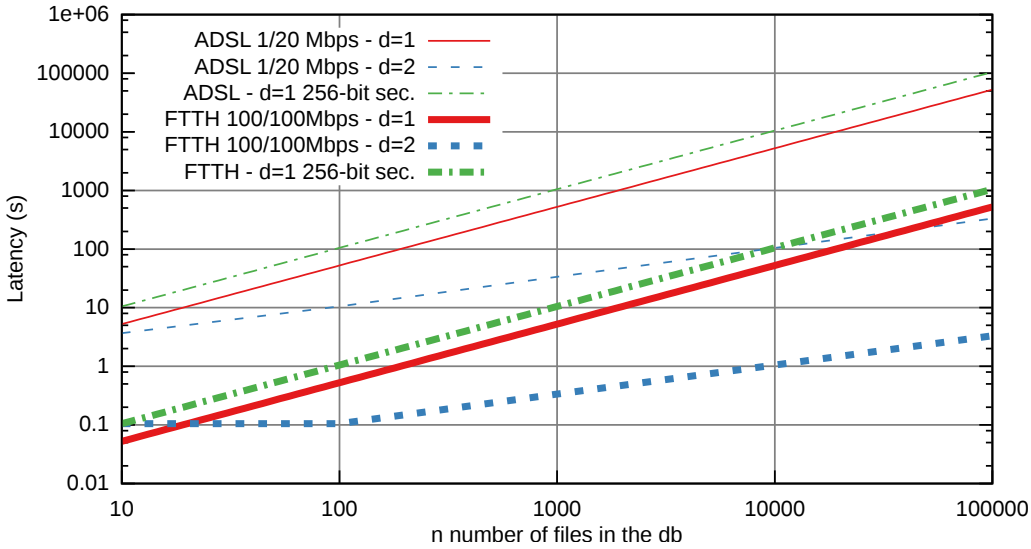
La ligne verte représente un résultat équivalent à la ligne rouge, mais avec une sécurité

de 256 bits. Comme nous l'avions écrit plus tôt, cette configuration n'a pas d'impact sur la vitesse de traitement cependant elle double la taille de chaque chiffré (l'impact se voit dans la figure 5.6 qui mesure la latence au démarrage). L'échelle employée sur la figure est logarithmique donc, même si les différences entre ces deux lignes semblent ténues, cette configuration est 10% moins performante.

La ligne bleue représente les performances avec un niveau de récursion. C'est-à-dire lorsque la base de données est représentée par un tableau à deux dimensions de $\sqrt{n} \times \sqrt{n}$ éléments et que la taille de la requête est proportionnelle à $2\sqrt{n}$. La récursion provoque un surcoût important lors du traitement de petites bases de données, car les données sont traitées une première fois, puis les éléments obtenus servent de base de données intermédiaire dont la taille est proportionnelle à $\mathrm{F}\sqrt{n}$. Cette base intermédiaire doit être traitée afin d'obtenir la réponse. Dans notre implémentationi, l'application d'une récursion sur ce type de base de données est environ dix fois plus coûteuses que sans récursion. Si $\sqrt{n} >> 10\mathrm{F}$, alors le surcoût induit par la base de données intermédiaire devient négligeable, car petit proportionnellement à la base de données originale.

5.4.2.1 Latence initiale

Figure 5.6 – Période de latence avant qu'un utilisateur commence à réceptionner le flux de données (essentiellement produit par la génération de la requête et sa transmission). Pas de période de latence pour le PIR trivial. Les traits fins sont pour l'ADSL et les épais pour la FTTH. Les couleurs et les styles des lignes sont associés aux même paramètres que la figure 5.5. Ces résultats soulignent que la période de latence grandit linéairement en n lorsque la dimension $d = 1$, en $\sqrt{n}$ lorsque $d = 2$ et que le facteur limitant est la bande passante d'envoi de l'utilisateur.



Même si obtenir le meilleur débit perçu par l'utilisateur est souvent l'objectif d'une application qui utilise le réseau internet, un autre paramètre non négligeable est la durée qu'un utilisateur doit patienter avant que de flux démarre. La Figure 5.6 souligne l'avantage d'utiliser un niveau de récursion pour les bases de données composées de nombreux éléments. Ce qui est particulièrement vrai lorsque $n \geq 1000$ car, comme expliqué précédemment, cette configuration n'implique qu'un faible surcoût calculatoire. Sur une ligne FTTH, le temps de latence se situe en dessous des dix secondes avec $d = 2$ et $n \geq 1000$. Une ligne ASDL est bien plus limitée en termes de débit montant, ainsi le temps de latence varie de 5 à 500 secondes. Par conséquent, dans de tels cas, un niveau de récursion doit impérativement être utilisé, même si cela entraine un fort surcoût pour la génération de la réponse. Le comportement inattendu de la latence pour un faible nombre d'éléments sur la ligne FTTH

provient du fait que nous utilisons des sockets TCP pour transmettre les requêtes durant une brève durée qui est maximisée par un tampon et le système de fenêtrage. Il est possible de configurer ces deux paramètres ou d'utiliser directement des sockets UDP afin d'obtenir un comportement plus linéaire.

5.4.2.2 Le cas d'utilisation Netflix

Netflix est un service de vidéo à la demande diffusé en flux continu. La base de données de l'entreprise était, en 2013 composées de 100000 titres stockés en tant que fichiers statiques et pouvant être prétraités. Le standard actuel de compression vidéo et le H.264, mais tend à être remplacé par son successeur, le H.265 [50, 41]. Le niveau de compression obtenu par ce dernier donne des flux vidéo en 720p à un débit situé autour des 400kbit/s pour 30ips (images par secondes) ou 800kbit/s à 60 ips et 2Mb/s pour un film en 1080p. Un flux audio encodé en MP3, a typiquement un débit de 128kbit/s. Ces niveaux sont représentés par des lignes horizontales sur la Figure 5.5.

Dans [20] Gupta et coll. présentent un prototype de consommation de médias nommé Popcorn. Dans cet article, les auteurs considèrent les mêmes types de débit pour une base de données de 8 000 films (au lieu de 100 000). L'approche, qui combine un protocole PIR fondé sur des bases de données répliquées et un cPIR, est probablement meilleur qu'utiliser uniquement un cPIR. Nous sommes aussi conscients qu'utiliser un protocole cPIR dans un scénario de type Netflix, bien qu'efficace en termes de protection de la vie privée, pourrait poser des problèmes de droits d'auteurs. Comme nous l'avons indiqué auparavant, nous avons pris ce cas d'utilisation principalement pour illustrer les excellentes performances atteintes par XPIR à large échelle et non pour débattre du fait qu'une société offrant un service de vidéo à la demande devrait utiliser un protocole cPIR (ou un système plus élaboré) pour envoyer les données aux clients. Enfin, il nous semble important de préciser que le cPIR employé dans Poporn se fonde sur le cryptosystème cassé de Trostle et Parrish. Le remplacer par XPIR augmenterait la sécurité de leur construction et augmenterait leurs performances de $\times 100$ sur leur partie cPIR.

D'après les résultats de la Figure 5.5, un serveur de type Netflix fondé sur XPIR a la possibilité d'envoyer des données à ses clients en jouant sur un équilibre entre vie privée et qualité de réception. Si un utilisateur accepte de recevoir un flux vidéo en 720p-30ips, il peut dissimuler son choix parmi 35K titres sur le serveur. S' il préfère privilégier la qualité de la vidéo, il pourra télécharger un flux en 1080p-60ips au détriment de sa vie privée, car il ne pourra cacher son choix que parmi 8K titres. Il est aussi envisageable de réaliser un compromis sur le coût calculatoire, par exemple pour 8K films en 720p-30ips huit pour cent de la capacité d'un CPU du serveur est utilisée, par conséquent un seul CPU peut gérer, dans ce cas, 12 clients.

5.4.3 Problèmes d'accès

Obtenir des résultats expérimentaux avec des bases de données fut aisé tant que la taille de celles-ci ne dépassait pas les 10 Gbit, car suffisamment de RAM était disponible sur nos machines de tests. Afin de recueillir les résultats issus de bases de données plus grandes, nous les avons découpées en morceaux adaptés à la RAM et supprimés les temps de transferts entre le support de stockage et la RAM pour chaque morceau. En utilisant le SSD et en prenant en compte les temps de transfert, les temps d'accès au support de stockage deviennent le goulet d'étranglement. Alors, les résultats de performances sont représentés par une ligne droite à $2/n$ Gbit/s (la vitesse de lecture du SSD est de 4 Gbit/s et le données prétraités sont deux fois plus grandes que leur version originale). Dans le cas d'utilisation décrite jusqu'à maintenant, l'impact produit est la réduction par 7 du maximum de films parmi lesquels le choix des utilisateurs est dissimulé pour une résolution donnée. Nous estimons que, dans le cas d'applications nécessitant de grandes bases de données (et débit utile) comme Netflix, le fournisseur de services emploie des supports de stockage hautes performances. Afin d'avoir des supports de stockage dont les performances sont compatibles avec XPIR, on pourrait par exemple utiliser deux SSD Samsung 950 Pro PCIe NVMe qui donneraient un débit de lecture séquentielle d'environ 40 Gbit/s.

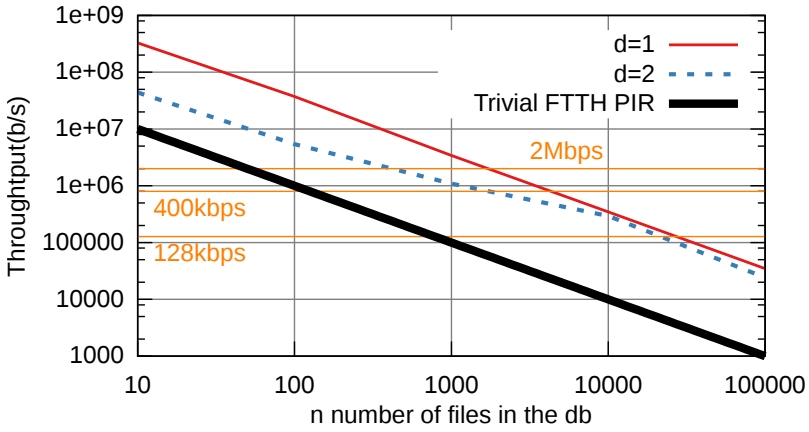
Dans le cas de la gestion de plusieurs utilisateurs simultanés, si l'accès aux données et fait de façon synchrone pour des utilisateurs concurrents, les temps d'accès disque n'augmentent pas ce qui ne pose pas de problèmes lors d'une mise à l'échelle. Pour illustrer notre propos, prenons par exemple 12 utilisateurs qui dissimulent leur choix parmi 8K films avec une qualité de 720p-30ips. Dans cette configuration le serveur accède à la base de données à huit pour cent de sa vitesse de traitement, c'est-à-dire $15 \times 8/100 = 1.2$Gbit/s, donc un support de stockage de base suffit.

5.4.4 Haut débit de traitement sur les données dynamiques

La gestion des bases de données dynamiques peut sembler, au premier abord, similaire aux bases statiques sur lesquelles le serveur n'appliquerait pas le prétraitement. Nous pouvons citer comme exemple la télévision sur IP (IPTV). Néanmoins, ces bases de données ont des configurations variées et ne sont pas toujours une simple extension des bases de données statiques avec des fichiers de taille «infinie». Une analyse exhaustive des cas d'utilisation des bases de données dynamiques dépasse le cadre de ce chapitre cependant nous nous concentrons sur deux cas : IPTV et sniffeur.

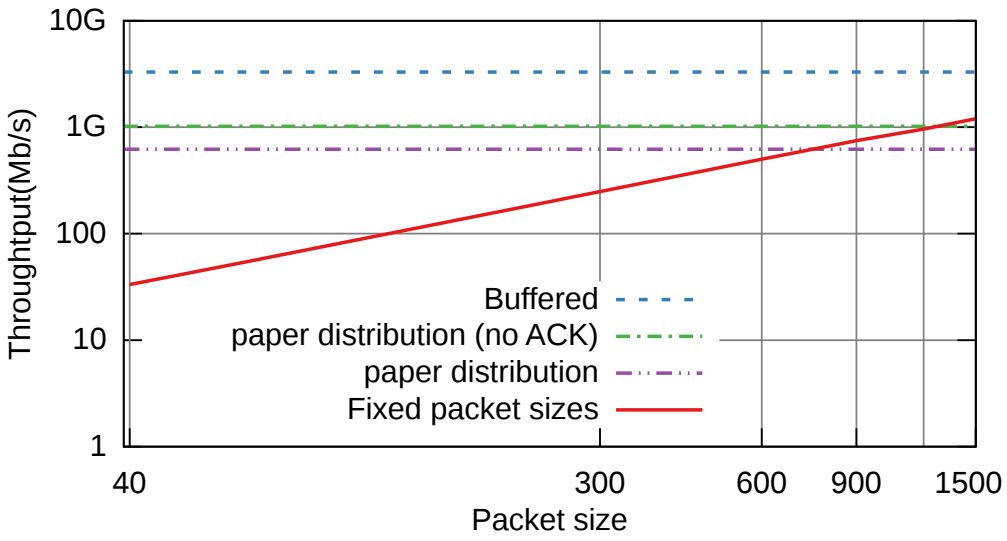
Le premier cas d'utilisation est relativement simple : des données ordinaires envoyées sur forme de flux continus qui ne peuvent pas être prétraités tels que la télévision sur IP. La figure 5.7 présente des résultats de débit utile perçu par l'utilisateur de la même manière que dans la figure 5.5. Comme on peut le constater, le débit utilisable est divisé environ par 6. Pour une application de type IPTV, un seul processeur est capable de gérer une centaine de flux vidéo en 720p-30ips pour 50 clients simultanément (télévision classique).

Figure 5.7 – Débit de traitement perçu par utilisateur recevant une réponse cPIR générée par un ordinateur portable MSI GT60 à partir de données dynamiques. Le PIR trivial sur une ligne FTTH de 100Mbits (ligne noire épaisse) est de 5 à 50 fois plus lent que notre cPIR. La ligne rouge représente le débit de traitement sans récursion (la base de données est traitée sous la forme d'une seule dimension) et la ligne bleue, avec un niveau de récursion (2 dimensions). La ligne horizontale correspond au débit nécessaire afin de regarder un file en 1080p (2Mb/s), 720p-60ips (800Kbps) et 720p-30ips (400Kbps), ou pour écouter un fichier audio à 128Kbps. Les performances sur un serveur avec un processeur plus performant (par ex. un Intel Xeon E7-4870 à dix coeurs) sont de deux à trois fois plus élevées.



5.4.4.1 Le cas d'utilisation Sniffeur

Figure 5.8 – Débit de traitement des paquets dans le cas d'utilisation du Sniffer sur un ordinateur portable MSI GT60. Les performances du PIR trivial n'ont pas de sens dans ce cas particulier. La ligne rouge, qui représente les performances obtenues avec différentes tailles de paquets, en octets sur la ligne des abscisse, indépendamment (c.-à-d. mesure des performances pour des paquets de 40 octets uniquement, puis 80 octets…). La ligne verte représente le débit de traitement lorsque le trafic internet suit une distribution bimodale classique. La ligne verte représente les performances pour un trafic dans lequel nous ignorons les paquets dont la taille est inférieure à 60 octets (des ACKs par exemple). La ligne bleue représente les performances dans le cas ou nous attendons que les buffers soient pleins et nous générons la requête cPIR seulement lorsque suffisamment d'informations sont présentes pour remplir un chiffré



Dans ce cas d'utilisation, nous supposons l'existence d'un utilisateur qui utiliserait un sniffeur réseau qui stocke tous les paquets qui ont une adresse IP source donnée. Cet utilisateur voudrait, dans le cas où un adversaire aurait accès au sniffeur, qu'il soit impossible de connaître l'IP du filtre sans secret. La solution naïve et de stocker l'intégralité des paquets sur le sniffeur (ce qui revient à faire un cPIR trivial). Cependant en utilisant un cPIR il est possible de réduire fortement le stockage nécessaire. Dans cette approche, une requête cPIR est générée et une adresse IP différente est associée à chaque élément de requête. Ce qui entraine une première question : quelle est la taille maximale de la plage d'adresse IP gérable par le protocole ? Supposons que nous utilisons le cryptosystème Ring-LWE avec comme paramètres $(1024, 60)$ où $(2048, 120)$. Dans le premier cas, les éléments de requêtes font 128kbit et dans le second 512kbit. Si notre but et de couvrir une adresse IPv4 de classe B (65535 adresses) la taille de la requête sera de 1Goctet dans le premier cas et 4Goctets dans le second. Dans cette forme particulière de cPIR, la requête n'a pas besoin d'être envoyée régulièrement. Dans un sniffeur la requête peut même être fixe (on peut considérer la requête comme une partie chiffrée du programme embarqué dans le sniffeur) et utilisée pour stocker les données résultantes sur un support de stockage local avant d'être récupérées (les résultats du filtrage peuvent aussi être transférés par le réseau). Cette taille n'affecte pas les performances, comme montré par nos résultats sur le débit de traitement qui ne dépendent pas du nombre d'éléments dans la base de données, tant qu'il y a suffisamment d'espace dans la RAM, ce que nous supposons vrai ici.

Pour chaque paquet intercepté, le sniffeur construit une base de données telle que chaque élément de requête est associé avec un élément nul excepté l'élément qui correspond à l'adresse IP source du paquet à intercepter. Le paquet et l'élément de requête sont associés dans ce cas. Puis, le sniffeur produit une réponse cPIR qu'il place sur son support de stockage.

Cette base de données est très particulière, car presque tous ses éléments sont nuls et que l'élément à traiter sera souvent bien plus petit (entre 320bits et 12kbit) que ce qu'un chiffré peut contenir (20kbit pour les paramètres les plus petits et 90kbit pour les plus grands).

Dans la figure 5.8 est représenté le débit de traitement des paquets interceptés par le sniffeur. Comme expliqué plus tôt nous avons adopté les paramètres cryptographiques sécurisés les plus petits $(1024, 60)$ car les paquets sont plus petits que de messages clairs classiques. Nous estimons qu'après une absorption, un chiffré doit pouvoir subir jusqu'à 100 sommes (utile à des opérations comme l'insertion dans un filtre de Bloom, etc.). Selon la structure de notre cryptosystème, cette contrainte implique que la taille des messages clairs doit être de 15kbit. Si nous générons une réponse cPIR pour chaque paquet de 40 octets interceptés par le sniffeur, la majorité de l'espace disponible dans le chiffré obtenu sera perdu, mais le coût de la génération de la réponse ne sera pas diminué (pour les éléments nuls cette opération est gratuite, mais pour les petits éléments cette opération a le même coût que pour un message clair entier à cause de la transformée NTT). Ainsi, si le sniffeur doit traiter des paquets de 400 octets (au lieu de 40), le coût de la génération de la réponse est le même pour $10\times$ plus d'informations traitées.

Si nous supposons que les paquets suivent une distribution bimodale classique (40% de paquets très petits, 40% proches de la MTU et 20% entre les deux), le sniffeur est susceptible de traiter une connexion réseau à 600Mb/s (représenté par la ligne violette dans la figure 5.8). Si nous supposons que le filtre du sniffeur ignore les très petits paquets (par ex. paquets de synchronisation et d'acquittement TCP), le sniffeur est alors capable de traiter 1 Gbit/s (ligne verte). Si, pour générer la réponse cPIR, nous attendons d'avoir suffisamment de paquets en provenance d'une adresse IP donnée afin de remplir entièrement un chiffré, les performances sont alors bien meilleures. Dans ce cas nous pouvons choisir des paramètres plus adaptés afin d'augmenter la vitesse de traitement tels que $(2048, 120)$. Dans cette configuration le sniffeur peut traiter 3Gbit/s (ligne bleue) si nous stockons 90kbit de données provenant d'une adresse IP donnée avant de générer la réponse cPIR. Utiliser des paramètres fournissant une sécurité plus élevée donne des performances de traitement équivalentes pour une requête deux fois plus grande.

Certes, concevoir un prototype complet de recherche privée impliquerait d'autres questions, comme s'assurer que d'autres aspects du système (l'interception des paquets, la fonction de compression de la sortie telle que le filtre de Bloom) ne ralentissent pas le traitement, cependant ces questions sortent du champ de ce chapitre.

5.4.5 Latence sur les données statiques/dynamiques

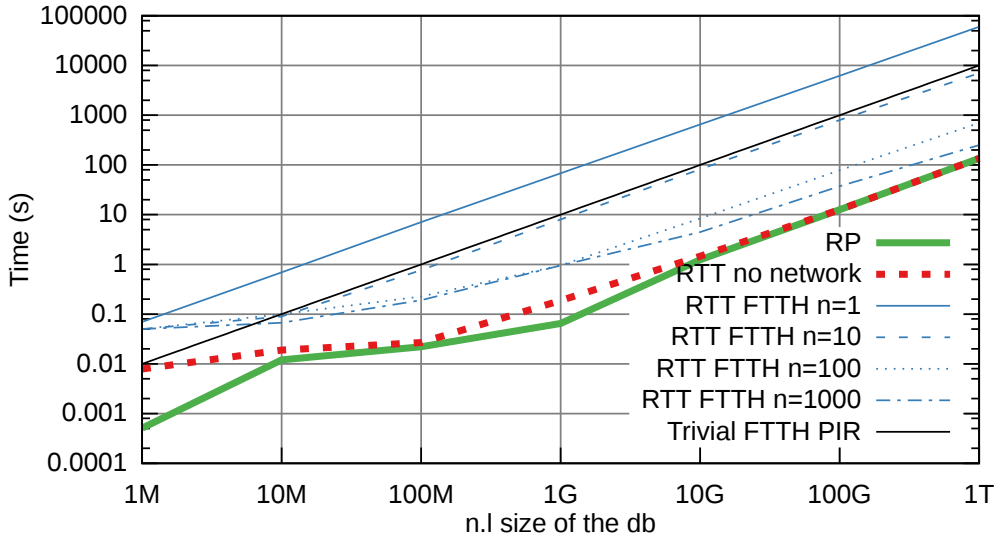
Dans cette section, nous voulons évaluer la latence totale de XPIR, c'est-à-dire la durée que doit attendre un utilisateur avant de recevoir intégralement l'élément pour des bases de données statiques et dynamiques. Les termes durée d'aller-retour (*Round Trip Time* RTT) et latence sont employés de manière similaire.

La figure 5.9 donne les mesures de RTT obtenues à partir de données statiques tandis que la figure 5.10 donne les mesures de RTT à partir de données dynamiques. Sur l'axe d'abscisse est représentée la taille de base de données de 1Mbit à 1Tbit. La ligne rouge donne la RTT obtenue sans communications réseau, le client et le serveur se trouvent donc sur la même machine. La ligne verte donne la durée de génération de la réponse (*Processing Time*, RP) et les différentes lignes bleues donnent la RTT sur une ligne FTTH pour différentes valeurs de n. Lorsque nous examinions précédemment la vitesse de traitement, le traitement de la requête et l'importation des données étaient les paramètres les plus coûteux, dans le cas de la RTT, les performances sont le résultat d'un équilibre entre là durée traitement de la réponse et les durées de téléchargements montants/descendants.

Il est très important de noter que les techniques de cPIR d'agrégation et de récursion sont indispensables pour garder la RTT le plus bas possible. Pour produire la figure 5.9 nous avons employé les paramètres cryptographiques $(1024, 60)$ pour Ring-LWE. Ainsi la taille des éléments de requête est de 128kbit et le facteur d'extension $\mathrm{F} \simeq 6$. Pour $n = 100000$ et $\ell =$1Mbit, si nous n'utilisons pas l'agrégation et la récursion, envoyer entièrement une requête de (100000 $\times$128kbit) sur la ligne FTTH dure 12,8 secondes et envoyer la réponse $(6 \times 1\mathrm{Mb})$ dure 0,06 seconde alors que générer la requête (à 2,2 Gbit/s) dure 0,05 seconde, et

la traiter (à 10 Gbit/s) dure 0,1 seconde et déchiffrer la réponse (à 5,6Gbit/s) dure environ 1ms. Employer la récursion divise la durée d'envoi de la requête d'un facteur 50 et à un très faible impacte sur le reste du processus, dans ce cas la récursion est avantageuse.

Figure 5.9 – Durées d'aller-retour (RTT) et de traitement de la requête (RP) sur un ordinateur portable MSI avec un Core i7-3630QM 2.64 GHz pour des données statiques sur un réseau FTTH (la taille des bases de données est en bits). Le PIR trivial (seconde ligne pleine en partant du haut) est plus rapide que le cPIR pour les bases de données de moins de 10 éléments, car notre cPIR a un facteur d'expansion d'environ 5. Pour les bases de données ayant plus d'éléments, un cPIR peut être jusqu'à 50× plus rapide que le PIR trivial. Lorsque le client est local, la latence (RTT, ligne rouge épaisse en pointillés) tend vers RP (ligne verte épaisse continue), le temps de traitement de la réponse Ceci particulièrement vrai pour les bases de données les plus grandes. Chaque fine ligne bleue donne une valeur de RTT pour un n fixe et taille de base de données variante. Pour les grandes bases de données, la taille de la réponse est le facteur limitant, ce qui explique pourquoi les performances obtenues sont proches d'un RTT idéal, lorsque n grandit. Pour les petites bases de données, la taille de la requête est le facteur limitant, car l'optimiseur utilise l'agrégation d'éléments.



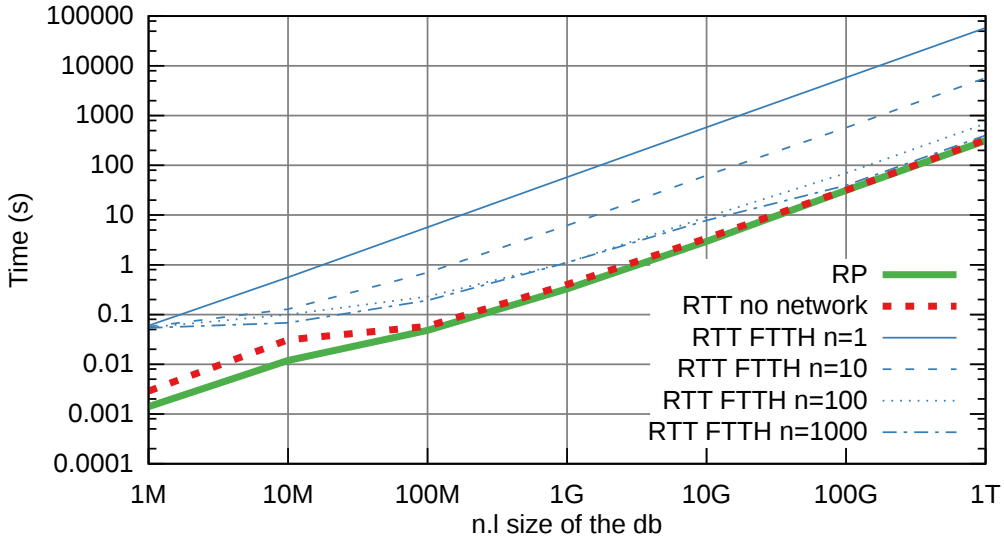
L'utilisation de l'agrégation et de la récursion permet à l'optimiseur de déterminer des paramètres cPIR afin de transformer une base de données dont le nombre d'éléments n est très grand en un base de données dont le n est plus petit. Voilà pourquoi, sur les deux figures, la RTT semble inversement proportionnel à n. En effet, la configuration de la base de données est choisie afin de réduire la RTT si un n plus petit est avantageux. Comme on peut l'observer, la partie des lignes pour laquelle n est élevé tend à se rapprocher à la durée minimum de RTT possible qui est la durée de RP (génération de la réponse). La seule différence entre les bases de données dynamiques et statiques repose sur la vitesse de traitement de la requête. Cette dernière est impactée par la durée de prétraitement dans le cas de la base de données dynamique. Dans le cas dynamique, la partie des résultats obtenus avec un n élevé tend plus tard vers RP, c'est-à-dire avec des bases de données de taille plus importante.

5.4.5.1 Le cas d'utilisation Match.com

Dans ce cas d'utilisation, nous considérons qu'une base de données d'un serveur fournissant un service de rencontre en ligne veut proposer un mécanisme payant de recherche privée par mots clés. Pour utiliser un tel système, les clients définissent un ensemble de critères publics, tels que la ville dans laquelle ils souhaitent rencontrer d'autres personnes (qui est probablement déjà révélée par leur adresse IP). Cet ensemble de paramètres publics réduira la taille de la base de données sur laquelle une seconde recherche sera effectuée, sur

des critères privés cette fois. Supposons l'existance d' une base de données d'un million de profils de chacun 1 Mbit alors la taille de la base de données est de 1 Tbit.

Figure 5.10 – Durées d'aller-retour (RTT) et de traitement de la requête (RP) sur un ordinateur portable MSI avec un Core i7-3630QM 2.64 GHz pour des données dynamiques sur un réseau FTTH (la taille des bases de données est en bits). Par souci de lisibilité le PIR trivial a été enlevé, car les remarques de la Figure 5.9 s'appliquent ici aussi. Comme les données ne sont pas prétraitées ici, la durée de traitement de la requête est plus élevée alors que la durée de téléchargement montant et descendant reste inchangée. Ceci explique pourquoi les lignes bleues sont presque identiques sauf pour l'écart par rapport au RTT idéale (plus petit). En pratique et sauf pour les grandes bases de données, la durée d'aller-retour n'est pas affectée par le prétraitement.



Nous devons aussi prendre en compte que chacun des profils peut correspondre à un ensemble de mots clés et que le coût de la génération de la réponse et multiplié par la moyenne des correspondances de mots clés lors des recherches privées. Si nous supposons qu'en moyenne un profil est constitué de cinq mots clés, et si nous prenons le RTT donné par la figure 5.9, un utilisateur devra attendre une dizaine de minutes avant de recevoir une réponse. Une telle durée n'est clairement pas acceptable pour une expérience du web confortable. Employer le pré filtrage par mots clés publics décrits précédemment pourrait diviser la taille de la base de données d'un facteur 10 à un facteur 100 (si les utilisateurs sont répartis dans des villes différentes et que les mots clés sont suffisamment distinctifs) et réduire le temps d'attente de 6-60 secondes, ce qui semble plus acceptable. Néanmoins, si nous considérons une base de données équivalente à celle de Match.com, c'est-à-dire 5 millions d'utilisateurs [7] et des profils de plusieurs mégaoctets, le filtrage public se devrait d'être bien plus efficace.

Remarque.

Être en mesure d'appliquer un protocole cPIR sur un réseau social d'une taille aussi importante était impensable jusqu'à maintenant.

5.4.6 Le cas d'utilisation de la bourse de New York

Dans ce dernier cas d'utilisation, nous nous sommes concentrés sur l'utilisation de XPIR avec des flux dynamiques en minimisant la latence. L'infrastructure sécurisée de transaction financière de la bourse de New York (*New York Stock Exchange*, NYSE) gère entre 5 et 10 Gbit/s de données provenant des marchés financiers de la planète. Le scandale de «Bloomberg News» [58] est une bonne illustration de l'importance du respect du secret dans le cas des

centres intérêts de clients pour des valeurs boursières. Deux usages du cPIR dans ce type de cas d'utilisation sont envisageables : le premier est orienté vitesse de traitement et le second est orienté latence. Dans le premier cas, un client souhaiterait s'inscrire à un ensemble de flux de données afin de recevoir des informations concernant les actions boursières d'entreprises, des analyses, etc. constamment à jour. Dans un tel cas, l'application est très semblable à un service de télévision sur IP dont le flux de données concerne des informations financières. Pour une analyse de performances, le lecteur pourra se référencer à la section 5.4.4.

Dans le second cas, un client souhaiterait recevoir le plus vite possible le dernier ensemble d'informations concernant une société en particulier. Dans ce cas, les services des marchés financiers peuvent être représentés comme une collecte de données générées par des capteurs distants dont les accès seraient donnés au client suite à des requêtes. La question la plus difficile est : quelle est la durée nécessaire à un client pour recevoir les données demandées ? Ou autrement dit, quelle est la fraîcheur des données reçues ? Par exemple, supposons qu'un utilisateur veuille récupérer des informations des dernières 100ms (un utilisateur ne peut pas espérer recevoir des données plus récentes étant donné le temps de latence engendré par le réseau). Dans le cas d'un flux de données financières à 5Gbit/s, la quantité de données obtenues en 100ms est 500Mbit. Comme ce type de données est composé de nombreux éléments différents, nous pouvons espérer que la durée de latence obtenue est proche de la ligne optimale de la figure 5.10. Ainsi, un utilisateur obtiendrait ses informations en environ 100ms, ce qui est un temps d'attente raisonnable pour des informations déjà âgées de 100ms.

5.4.7 Autres cryptosystèmes

Comme précisé précédemment, l'optimiseur choisit dans la grande majorité des situations le cryptosystème Ring-LWE. Toutefois dans quelques cas extrêmes, l'optimiseur choisit un cPIR fondé sur le cryptosystème de Paillier ou un PIR trivial (téléchargement complet de la base de données).

Le cryptosystème de Paillier sera choisi dans les cas ou le débit réseau est particulièrement faible ce qui implique que le temps de génération de la réponse sera négligé, car il ne représente qu'une fraction du temps passé à envoyer la réponse. Le paramètre à minimiser dans ce cas est le facteur d'expansion et pour cela Paillier est le meilleur choix. À l'inverse, le PIR trivial est le choix évident lorsque le débit disponible est supérieur à la vitesse de traitement de la base de données. La limite se situe proche de 20Gbit/s pour les bases de données statiques prétraitées et 5Gbit/s pour les bases de données dynamiques. D'autres configurations extrêmes justifient aussi l'utilisation du PIR trivial. Par exemple pour les bases de données composées de 2 à 4 éléments. Dans ce cas la réponse cPIR générée avec notre système de chiffrement fondé sur Ring-LWE sera plus grande que la base de données elle-même à cause de facteurs d'expansion. Un autre exemple valable est le cas de petites bases de données pour lesquelles la requête se trouve être plus grande que la base de données. Prenons la configuration suivante : une connexion ADSL (1Mb/s montant / 20Mb/s descendant) et une base de données de 10 Mbit avec 10 éléments. Une seconde sera nécessaire pour renvoyer une requête construite avec Ring-LWE, alors que le téléchargement de la base de données en entier prendra une demi-seconde (l'agrégation ne résout pas le problème).

5.5 Conclusion

La cryptographie fondée sur les réseaux euclidiens a apporté des avancées fondamentales sur le chiffrement entièrement homomorphe (FHE). Néanmoins et malgré ses excellents résultats asymptotiques, elle est jugée trop coûteuse. Mais la possibilité d'employer les réseaux d'idéaux ainsi que la mise au point des nombreuses optimisations ont permis d'atteindre d'excellentes performances hors des cas asymptotiques.

Le cPIR a été longtemps considéré comme un protocole quasi-inutilisable [53] mais la cryptographie fondée sur les réseaux euclidiens a changé la donne. Nous avons montré que notre protocole traite un grand nombre de bases de données en quelques secondes même pour des bases de données de 100Gbits.

Si nous avions réalisé ces expérimentations avec le cryptosystème de Paillier, les durées mesurées auraient atteint des milliers de secondes, car la base de données aurait été traitée à 1Mb/s. Envoyer par un PIR trivial la base de données sur un lien 100Mb/s est également des centaines de fois plus lent (à part dans des cas extrêmes) que ce que nous avons présenté. De plus, les résultats ont été obtenus sur un ordinateur portable et utiliser un serveur équipé d'un processeur de dernière génération ne ferait qu'augmenter cet écart. La cryptographie fondée sur les réseaux euclidiens a changé le « presque infaisable» en faisable par tous. Afin d'universaliser le plus possible XPIR, nous l'avons accompagné d'un outil d'auto-optimisation afin qu'un utilisateur non-spécialiste en cryptographie et cPIR puissent l'utiliser.

En conclusion, il nous semble important de préciser que, même si nous montrons que le cPIR peut être employé avec de très grandes bases de données, en pratique les bases de données grandissent plus vite que la puissance de calculs, et le cPIR peut être combiné à d'autres techniques afin d'améliorer la mise à l'échelle. Un exemple intéressant d'utilisation du cPIR comme brique technique est Popcorn [20] qui combine intelligemment le cPIR et itPIR afin de fournir un service de type Netflix plus efficace que l'exemple simple que nous avons proposé. Un autre exemple est l'efficacité des protocoles ORAM qui se fondent sur un cPIR.

Pour finir, nous souhaitons que XPIR augmente la diffusion des applications utilisant le cPIR comme sous-routine.

Conclusion générale

Dans ce mémoire de thèse, nous avons proposé une application respectueuse de la vie privée reposant sur un système cryptographique fondé sur les réseaux euclidiens. Notre démarche s'articule en trois grandes parties.

En premier lieu, nous avons fait un état de l'art des différentes applications respectueuses de la vie privée. Nous entendons par « applications respectueuses de la vie privée», des applications interactives entre un client et un serveur, dont les informations comme les centres d'intérêt, du client ou du serveur restent privées. Nous avons tout d'abord considéré trois grandes classes d'applications :

— L'Oblivious RAM (ORAM) dans laquelle un utilisateur stocke ses données chez un tiers et dans laquelle ils dissimulent la nature de ses données (ces dernières sont chiffrées) ainsi que la nature des opérations réalisées (lecture ou écriture)

— L'Oblivious Transfert (OT), dans lequel le serveur donne accès au client à un élément de sa base de données privée. Le serveur n'est pas informé de l'élément accédé par le client.

— Le Private Information Retrieval (PIR), dans lequel le client dissimule l'index de l'élément accédé de la base de données publique du serveur.

De ces trois grandes classes d'applications, nous nous sommes concentrés sur le PIR et plus particulièrement sur le computationnal PIR ou cPIR pour lequel il n'existait pas encore de solutions satisfaisantes contrairement à l'ORAM et l'OT.

Dans un protocole cPIR, l'indexe de l'élément qu'un client souhaite télécharger est dissimulé par un protocole cryptographique. Ainsi le client va générer une requête chiffrée, qu'il enverra au serveur. Ce dernier, à partir de la requête du client et de sa base de données, va générer la réponse. Comme l'index est dissimulé dans la requête du client, le serveur va traiter l'intégralité de sa base de données pour générer une réponse, ce qui rappelle la méthode triviale du protocole PIR (le serveur envoie l'intégralité de sa base de données). Une fois la réponse reçue, le client l'extrait (la déchiffre) et récupère ainsi l'élément à l'index désiré.

En second lieu, nous nous sommes focalisés sur la partie cryptographique du cPIR, car utilisé dans les trois phases du protocole (génération de la requête, génération de la réponse, extraction de la réponse). Les protocoles de cPIR existant sont soit très spécialisés, soit lents, car construits autour du protocole de Paillier. En partant de ce constat, la cryptographie fondée sur les réseaux euclidiens nous a semblé particulièrement appropriée comme remplacement potentiel. En effet, cette dernière permet de réaliser, comme pour le système de chiffrement de Paillier, les opérations sur les chiffrés nécessaires au serveur à la génération de la réponse, mais sans faire d'exponentiation modulaire (le point faible majeur niveau performances calculatoires du cryptosystème de Paillier). La particularité de ces cryptosystèmes réside dans l'utilisation de polynômes et non pas de grands entiers pour représenter les données chiffrées.

Nous avons construit une bibliothèque de manipulation de polynômes sur un anneau de polynômes (les coefficients sont représentés modulo en entier et les polynômes modulo un polynôme) dénommé NFLlib pour *NTT Fast Lattice Library*. Cette bibliothèque écrite dans le langage informatique `C++`, est conçue afin d'optimiser au maximum les performances calculatoires des opérations arithmétiques effectuées entre polynômes. Afin d'accélérer les multiplications entre polynômes, nous les avons représentés sous forme évaluée et pour les évaluer efficacement nous avons employé la transformée de Fourrier discrète et son inverse pour retrouver les polynômes sous forme de coefficients. Pour profiter au maximum des instructions SIMD des processeurs modernes nous nous sommes restreints à l'utilisation de types gérés nativement. Ceci nous a poussés à utiliser le théorème des restes chinois pour représenter des polynômes dont le module des coefficients est plus grand que la taille des types natifs. Dans le but d'accélérer les multiplications modulaires directement

entre coefficients, et plus particulièrement dans le cadre du cPIR, nous avons utilisé une méthode consistant à précalculer les coefficients de Newton de certains polynômes et d'un algorithme particulier afin de réduire au maximum le nombre de divisions (opération lente) réalisées par le processeur. NFLlib donne de très bonnes performances par rapport aux autres bibliothèques capables de manipuler des polynômes, car sont nombre de fonctions est relativement réduit, bien que les polynômes soient aisément manipulables, et restreint aux anneaux de polynômes dont le polynôme quotient est de la forme X^n+1 avec n une puissance de 2 ce qui permet de nombreuses optimisations. À partir de cette bibliothèque, nous avons construit un cryptosystème fondé sur le problème *Learning With Errors* sur un anneau (Ring-LWE) ayant pour objectif d'être beaucoup plus performant, pour des opérations équivalentes, que le cryptosystème de Paillier. Nous espérons que NFLlib popularisera les systèmes cryptographiques fondés sur les réseaux euclidiens.

En dernier lieu, nous avons construit, à partir du cryptosystème fondé sur le problème Ring-LWE écrit avec NFLlib, un cPIR que nous avons nommé *XPIR : Private Information Retrieval for Everyone*. XPIR est une application de cPIR composée d'un client, d'un serveur et d'un optimiseur de paramètres. Cet optimiseur simplifie la configuration des paramètres aux non-spécialistes. De plus, XPIR est aussi disponible sous la forme d'une bibliothèque dont l'accès est facilité par une interface de programmation (API) ce qui permet d'intégrer un protocole cPIR à d'autres applications. En vue des démontrer la versatilité de XPIR, nous avons étudié deux familles de bases de données dans différents cas d'utilisation. Les bases de données dites statiques prétraitables et les bases de données dynamiques non prétraitables sur deux critères : le débit de données disponible pour l'utilisateur et la durée attendue par l'utilisateur avant de recevoir le premier bit de la réponse (latence). Nous avons illustré ces deux familles par des cas d'utilisations. Concernant la partie statique, nous avons choisi les cas Netflix, un service de diffusion de média en flux, et *Match.com* un service de rencontre en ligne. Pour la partie dynamique nous avons choisi, le cas de la télévision sur IP, et des informations issues de places financières. Nous avons conclu des performances de XPIR qu'il est possible de construire des cPIR utilisables dans des cas réalistes. Nous espérons que XPIR popularisera le retrait d'information privé par ses performances et sa simplicité d'utilisation.

Comme nous l'avons écrit dans ce mémoire, l'utilisation unique de cPIR pour construire des applications respectueuses de la vie n'est peut-être pas la meilleure solution. Néanmoins, employer un protocole cPIR dans une application hybride ou comme brique technique semble prometteur. En conclusion, l'emploi massif de telles applications sur Internet, qui semble maintenant réaliste, permettrait de réduire les atteintes constantes à la vie privée des internautes et ainsi réduire les abus de certains gouvernements et entreprises.

Bibliographie

[1] Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian et Tancrede Lepoint. "NFLlib : NTT-based fast lattice library". In : *Cryptographers' Track at the RSA Conference*. Springer. 2016, p. 341–356.

[2] Paul Barrett. "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor". In : *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1986, p. 311–323.

[3] Daniel J Bernstein. "The Salsa20 family of stream ciphers". In : *New stream cipher designs*. Springer, 2008, p. 84–97.

[4] Zvika Brakerski, Craig Gentry et Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption Without Bootstrapping". In : *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. Cambridge, Massachusetts : ACM, 2012, p. 309–325. isbn : 978-1-4503-1115-1. doi : `10.1145/2090236.2090262`. url : `http://doi.acm.org/10.1145/2090236.2090262`.

[5] Zvika Brakerski et Vinod Vaikuntanathan. "Efficient fully homomorphic encryption from (standard) LWE". In : *SIAM Journal on Computing* 43.2 (2014), p. 89.

[6] Zvika Brakerski et Vinod Vaikuntanathan. "Fully homomorphic encryption from ring-LWE and security for key dependent messages". In : *Advances in Cryptology–CRYPTO 2011*. Springer, 2011, p. 505–524.

[7] Zvika Brakerski et Vinod Vaikuntanathan. "Reed Hasting : Cet homme veut pulvériser la télévision". In : *Le Point* 2142 (2013), p. 83.

[8] Christian Cachin, Silvio Micali et Markus Stadler. "Computationally private information retrieval with polylogarithmic communication". In : *Eurocrypt* (1999), p. 402–414.

[9] B. Chor, O. Goldreich, E. Kushilevitz et M. Sudan. "Private Information Retrieval". In : *FOCS* (oct. 1995), p. 41–50.

[10] Ivan Damgard et Mads Jurik. "A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System". In : *PKC 01, LNCS SERIES* (2001), p. 119–136.

[11] Giovanni Di Crescenzo, Tal Malkin et Rafail Ostrovsky. "Single Database Private Information Retrieval Implies Oblivious Transfer". In : *Advances in Cryptology – EUROCRYPT 2000*. Sous la dir. de Bart Preneel. T. 1807. Springer Berlin Heidelberg, 2000, p. 122–138. url : `http://dx.doi.org/10.1007/3-540-45539-6_10`.

[12] Whitfield Diffie et Martin E Hellman. "New directions in cryptography". In : *Information Theory, IEEE Transactions on* 22.6 (1976), p. 644–654.

[13] Yarkın Doröz, Aria Shahverdi, Thomas Eisenbarth et Berk Sunar. "Toward practical homomorphic evaluation of block ciphers using prince". In : *Financial Cryptography and Data Security*. Springer, 2014, p. 208–220.

[14] Taher ElGamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In : *Advances in cryptology*. Springer. 1985, p. 10–18.

[15] Craig Gentry et Ramzan Zulfikar. “Single-Database Private Information Retrieval with Constant Communication Rate”. In : *Proceedings of the 32nd International colloquium on Automata, Languages and Programming* 3580 (2005), p. 803–815.

[16] Oded Goldreich. *Computational Complexity : A Conceptual Perspective*. 1re éd. New York, NY, USA : Cambridge University Press, 2008. isbn : 052188473X.

[17] Norman Göttert, Thomas Feller, Michael Schneider, Johannes Buchmann et Sorin Huss. “On the design of hardware building blocks for modern lattice-based encryption schemes,” in : *CHES* (2012), p. 512–529.

[18] Tim Güneysu, Tobias Oder, Thomas Pöppelmann et Peter Schwabe. “Software Speed Records for Lattice-Based Signatures”. In : *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*. 2013, p. 67–82.

[19] Tim Güneysu, Tobias Oder, Thomas Pöppelmann et Peter Schwabe. “Software speed records for lattice-based signatures”. In : *Post-Quantum Cryptography*. Springer, 2013, p. 67–82.

[20] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi et Michael Walfish. “Scalable and private media consumption with Popcorn”. In : (2015). `http://eprint.iacr.org/`.

[21] Shai Halevi et Victor Shoup. “Algorithms in HElib”. In : *Advances in Cryptology–CRYPTO 2014*. Springer, 2014, p. 554–571.

[22] Shai Halevi et Victor Shoup. “Design and implementation of a homomorphic-encryption library”. In : *IBM Research (Manuscript)* (2013).

[23] William Hart. *Fast Library for Number Theory*. `http://www.flintlib.org`. 2014.

[24] David Harvey. “Faster arithmetic for number-theoretic transforms”. In : *J. Symb. Comput.* 60 (2014), p. 113–119.

[25] David Harvey. “Faster arithmetic for number-theoretic transforms”. In : *Journal of Symbolic Computation* 60 (2014), p. 113–119.

[26] Jeffrey Hoffstein, Jill Pipher et Joseph H Silverman. “NTRU : A ring-based public key cryptosystem”. In : *Algorithmic number theory*. Springer, 1998, p. 267–288.

[27] Alex Iliev et Sean Smith. “Privacy-enhanced credential services”. In : *2nd Annual PKI Research Workshop*. Citeseer. 2003.

[28] International Telecommunication Union. *Key ICT indicators for developed and developing countries and the world (totals and penetration rates)*. 2015. url : `https://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2015/ITU_Key_2005-2015_ICT_data.xls`.

[29] Gene Itkis. *Forward security, adaptive cryptography : Time evolution*. 2004.

[30] Bi Jingguo, Liu Mingjie et Wang Xiaoyun. “Cryptanalysis of Homomorphic Encryption Scheme From ISIT 2008”. In : *IEEE Symposium on Information Theory (ISIT)* (2012), p. 2152–2156.

[31] Eyal Kushilevitz et Rafail Ostrovsky. “Replication Is Not Needed : Single Database, Computationally-Private Information Retrieval”. In : *FOCS* (oct. 1997).

[32] Arjen Klaas Lenstra, Hendrik Willem Lenstra et László Lovász. “Factoring polynomials with rational coefficients”. In : *Mathematische Annalen* 261.4 (1982), p. 515–534.

[33] Tancrède Lepoint et Michael Naehrig. “A Comparison of the Homomorphic Encryption Schemes FV and YASHE”. In : *Progress in Cryptology - AFRICACRYPT 2014, Marrakesh, Morocco, May 28-30, 2014. Proceedings*. 2014, p. 318–335.

[34] Richard Lindner et Chris Peiker. “Better Key Sizes (and Attacks) for LWE-Based Encryption”. In : *CT-RSA* (2011), p. 319–339.

[35] Richard Lindner et Chris Peikert. “Better key sizes (and attacks) for LWE-based encryption”. In : *Topics in Cryptology–CT-RSA 2011*. Springer, 2011, p. 319–339.

[36] Helger Lipmaa. “An Oblivious Transfer Protocol with Log-Squared Communication”. In : *The 8th Information Security Conference* (2004).

[37] Vadim Lyubashevsky, Chris Peikert et Oded Regev. “On Ideal Lattices and Learning with Errors over Rings”. In : *Advances in Cryptology – EUROCRYPT 2010 : 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 – June 3, 2010. Proceedings*. Sous la dir. d’Henri Gilbert. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, p. 1–23. isbn : 978-3-642-13190-5. doi : `10.1007/978-3-642-13190-5_1`. url : `http://dx.doi.org/10.1007/978-3-642-13190-5_1`.

[38] Carlos Aguilar Melchor et Philippe Gaborit. “A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol”. In : *WEWORC* (2007).

[39] Niels Moller et Torbjörn Granlund. “Improved Division by Invariant Integers”. In : *IEEE Trans. Computers* 60.2 (2011), p. 165–175.

[40] Victor Shoup. *Number Theory Library (Version 8.1)*. 2015. url : `%5Curl%7Bhttp://www.shoup.net/ntl%7D`.

[41] J-R Ohm, Gary J Sullivan, Holger Schwarz, Thiow Keng Tan et Thomas Wiegand. “Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC)”. In : *Circuits and Systems for Video Technology, IEEE Transactions on* 22.12 (2012), p. 1669–1684.

[42] Pascal Paillier. “Public-Key Cryptosystems based on Composite Degree Residue Classes”. In : *EUROCRYPT* (1999), p. 223–238.

[43] Pascal Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In : *Advances in cryptology—EUROCRYPT’99*. Springer. 1999, p. 223–238.

[44] Michael O. Rabin. *How To Exchange Secrets with Oblivious Transfer*. Rapp. tech. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005. Aiken Computation Lab, Harvard University, 1981. url : `http://eprint.iacr.org/2005/187`.

[45] Oded Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In : *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. STOC '05. Baltimore, MD, USA : ACM, 2005, p. 84–93. isbn : 1-58113-960-8. doi : 10.1145/1060590.1060603. url : http://doi.acm.org/10.1145/1060590.1060603.

[46] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In : *Journal of the ACM (JACM)* 56.6 (2009), p. 34.

[47] Ronald L Rivest, Len Adleman et Michael L Dertouzos. "On data banks and privacy homomorphisms". In : *Foundations of secure computation* 4.11 (1978), p. 169–180.

[48] Ronald L Rivest, Adi Shamir et Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In : *Communications of the ACM* 21.2 (1978), p. 120–126.

[49] Benjamin Ray Seyfarth. *Introduction to 64 Bit Intel Assembly Language Programming for Linux*. CreateSpace Independent Publishing Platform ; 2 edition (June 23, 2012), 2013.

[50] Bo Shen, Sung-Ju Lee et Sujoy Basu. "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks". In : *Multimedia, IEEE Transactions on* 6.2 (2004), p. 375–386.

[51] Victor Shoup. "A new polynomial factorization algorithm and its implementation". In : *Journal of Symbolic Computation* 20.4 (1995), p. 363–397.

[52] Simon Singh. *Histoire des codes secrets*. 1999. isbn : 9782253150978.

[53] Radu Sion et Bogdan Carbunar. "On the computational practicality of private information retrieval". In : *In Proceedings of the Network and Distributed Systems Security Symposium*. 2007.

[54] Damien Stehlé et Ron Steinfeld. "Making NTRU as secure as worst-case problems over ideal lattices". In : *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2011, p. 27–47.

[55] Julien P. Stern. "A New Efficient All-Or-Nothing Disclosure of Secrets Protocol". In : *ASIACRYPT* (1998), p. 357–371.

[56] "The GNU Multiple Precision Arithmetic Library". In : *https ://gmplib.org* (2014). url : https://gmplib.org.

[57] Jonathan Trostle et Andy Parrish. "Efficient Computationally Private Information Retrieval from Anonymity or Trapdoor Groups". In : *Information Security*. Sous la dir. de Mike Burmester, Gene Tsudik, Spyros Magliveras et Ivana Ilić. T. 6531. Springer Berlin Heidelberg, 2011, p. 114–128. url : http://dx.doi.org/10.1007/978-3-642-18178-8_10.

[58] Liam Vaughan, Gavin Finch et Ambereen Choudhury. "Traders said to rig currency rates to profit off clients". In : *Bloomberg.com* (2013). url : https://www.bloomberg.com/news/articles/2013-06-11/traders-said-to-rig-currency-rates-to-profit-off-clients.