Application du chiffrement homomorphe à la protection de la vie privée

Cédric Lefebvre

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Informatique et Télécommunication

Présentée et soutenue par :

M. CÉDRIC LEFEBVRE

le vendredi 10 décembre 2021

Titre :

Application du chiffrement homomorphe à la protection de la vie privée

---

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (IRIT)

Directeurs de Thèse :

M. CARLOS AGUILAR MELCHOR

M. MARC-OLIVIER KILLIJIAN

Rapporteurs :

MME CAROLINE FONTAINE, CNRS PARIS

M. PATRICK LACHARME, ENSICAEN

Membres du jury :

M. ANDRE LUC BEYLOT, TOULOUSE INP, Président

M. CARLOS AGUILAR MELCHOR, ISAE-SUPAERO, Membre

M. MARC-OLIVIER KILLIJIAN, UNIVERSITE DU QUEBEC A MONTREAL, Membre

MME MARION VIDEAU, QUARKSLAB, Membre

Remerciements

Je tiens en premier lieu à remercier mes deux directeurs de thèse Carlos Aguilar Melchor et Marc-Olivier Killijian. Le sujet de recherche qui m'a été proposé était passionnant. J'ai pu pendant ma thèse rencontrer de nombreuses personnes, collaborer avec beaucoup de gens et cela grâce à vous. De plus, les travaux et discussions que nous avons menés ensemble étaient très fructueux et intéressants. J'ai pu grâce à vous découvrir le métier de chercheur de manière agréable. Je leur suis reconnaissant de m'avoir accompagné et encouragé dans des moments pas toujours faciles.

Je suis très reconnaissant à Caroline Fontaine et Patrick Lacharme qui ont rédigé des rapports réfléchis et pertinents de mon manuscrit de thèse malgré un emploi du temps chargé, les échanges que nous avons eu étaient très intéressants et les remarques fondées. Je tiens également à remercier André-Luc Beylot et Marion Videau pour avoir accepté d'être membre du jury de ma thèse et pour les discussions intéressantes que nous avons pu avoir.

Je remercie l'équipe RMESS de l'IRIT de m'avoir accueilli pour réaliser ma thèse. L'ambiance au seins de l'équipe est très agréable et c'était un bonheur pour moi d'avoir partagé du temps avec vous. Merci à Sylvie pour son efficacité et sa gentillesse en tant que secrétaire. Je tiens également à remercier André-Luc pour diriger et faire vivre cette équipe. Je remercie les doctorants de l'équipe avec qui j'ai pu partagé des moments conviviaux. Je garde de très bons souvenirs des repas et séminaires d'équipe, des journées au vert.

Je tiens à remercier l'équipe de Jean-Pierre Hubeaux à l'EPFL de m'avoir accueilli pendant 1 mois et demi. Les travaux que nous avons réalisés ensemble au début de ma thèse m'ont permis de découvrir d'une des meilleure façon possible mon sujet de thèse.

Je remercie Eric pour m'avoir accueilli au sein d'Airbus Defense and Space pendant 2 mois à Ottobrunn. Même si les travaux de recherche ont abouti un peu trop tard pour permettre d'être publiés, j'ai appris beaucoup de choses auprès de toi et tu as su faire preuve de pédagogie.

Merci à Chloé, Etienne, Elie et Ida pour l'expérience REDOCS, travailler avec vous

était vraiment intéressant, chacun a pu apporter sa connaissance pour mener à bien le projet. Merci également aux organisateurs des REDOCS, c'est un évènement que j'ai trouvé très pertinent !

Je remercie l'INSA et l'équipe TSF du LAAS-CNRS pour m'avoir accueilli lors de mon poste ATER permettant de finaliser la rédaction de ma thèse. L'ambiance de travail était très bonne et les enseignements intéressants.

Je remercie chaleureusement ma famille, mes parents qui malgré la distance m'ont soutenu et motivé pendant la thèse. Je remercie infiniment ma femme qui en plus de m'avoir soutenu pendant la thèse, a pris le temps de relire mon manuscrit et m'aider à préparer ma soutenance.

Table des matières

Table des figures

Liste des tableaux

Organisation du manuscrit

Le premier chapitre du manuscrit présentera dans un premier temps quelques éléments fondamentaux en cryptographie ainsi qu'un bref historique non exhaustif des schémas. Une fois ces éléments introduits, nous présenterons la cryptographie homomorphe basée sur les réseaux euclidiens. Ce type de cryptographie permet d'effectuer des opérations directement sur les chiffrés qui seront reportés sur les clairs et permet des algorithmes utiles dans le domaine de la protection de la vie privée. Quelques algorithmes seront présentés à la fin du premier chapitre.

Le second chapitre du manuscrit sera une étude et une présentation des principales librairies permettant d'effectuer du chiffrement homomorphe. Cette étude permet de choisir au mieux la librairie en fonction du cas d'usage et de choisir un paramétrage cohérent.

Les trois derniers chapitres du manuscrit représentent chacun un cas concret de l'utilisation du chiffrement homomorphe. Le chapitre 3 présente une solution permettant d'externaliser les données génomiques dans une base de données et de vérifier la présence de manière privée. Le chapitre 4 présente une solution permettant de réaliser un calcul particulier sur une base de donnée génomique de manière privée. Le chapitre 5 présente une solution permettant de réaliser une détection de fraude de manière privée lors d'un paiement.

Chapitre 1

État de l'art

1.1 Cryptographie

1.1.1 Histoire de la cryptographie

Le terme cryptographie vient des mots grecs *kruptos* « caché » et *graphein* « écrire ». Un des premiers documents connus recelant un secret enfoui dans l'écriture est une tablette en argile réalisée par un potier qui y avait dissimulé sa recette de fabrication en jouant sur l'orthographe des mots et en supprimant des consonnes. Elle a été retrouvée en Irak, et date du XVIème siècle avant Jésus Christ.

La cryptographie a pour but de sécuriser un message pour lequel on veut assurer des propriétés telles que la confidentialité (un adversaire ne peut pas lire un message) ou l'intégrité (un adversaire ne peut pas créer ou modifier un message valide). Parmi les algorithmes cryptographiques on peut distinguer les algorithmes de chiffrement, qui permettent d'assurer la confidentialité d'une donnée en combinant un message et une clé pour rendre le résultat (appelé chiffré) incompréhensible. Les premiers algorithmes de chiffrement étaient des algorithmes qui réalisaient des substitutions. Historiquement, on peut discerner trois types de chiffrement par substitution : mono-alphabétique (une lettre remplace une autre lettre), poly-alphabétique (une lettre est remplacée par une autre en fonction d'un état interne à l'algorithme) et poly-gramme (un groupe de lettres remplace un autre groupe de lettres).

Le chiffré de César est un système simple par substitution mono-alphabétique consistant à décaler les lettres de l'alphabet. Le chiffrement est limité par le nombre de lettres de l'alphabet. Il est assez facile de retrouver le message initial en essayant tous les décalages.

Blaise de Vigenère présenta, en 1586, une technique de chiffrement par substitution

poly-alphabétique. Ce chiffrement utilise une clé qui permet de remplacer chaque caractère. Plus la clé est grande et diversifiée, plus le chiffrement est solide. Des passages entiers de certaines œuvres littéraires pouvaient être utilisés. Ce chiffrement n'a été cassé qu'au milieu du XIXème siècle.

La cryptographie, par le passé, a souvent eu un rôle important lors des guerres. Le chiffrement de César a été par exemple utilisé par l'armée soviétique lors de la première guerre mondiale. Les connaissances en cryptographie des alliés lors de la première guerre mondiale seraient un élément de victoire essentiel. C'est dans ce contexte qu'est née Enigma, utilisée par les allemands lors de la seconde guerre mondiale. Il s'agit d'une machine portable utilisant des rotors sur cylindres afin de chiffrer et déchiffrer des messages. Il s'agit d'un chiffrement poly-gramme. Le chiffrement d'Enigma a été cassé par une équipe composée entre autres d'Alan Turing, qui est considéré comme le père de l'informatique. L'ouvrage "Les codes secrets décryptés" [1] de Didier Müller est extrêmement riche en détails et références vers ces événements historiques.

1.1.2 Cryptographie symétrique

Dans les cryptosystèmes abordés dans la section précédente, les deux entités qui communiquent doivent partager un même secret. Ces cryptosystèmes sont dits symétriques car ce secret est le même pour chiffrer et déchiffrer les messages.

Définition 1 (Chiffrement symétrique)**.** *Un schéma de chiffrement symétrique est composé de deux algorithmes*

Chiffrement(sk,m) : génère un chiffré c du message m en utilisant la clé sk.

Déchiffrement(sk,c) : retourne le clair m correspondant au chiffré c en utilisant la clé sk.

On distingue deux types de systèmes de chiffrement, les chiffrements par blocs, qui prennent n bits en entrée et en ressortent n pour un paramètre n appelé taille de bloc, et les chiffrements à flots, qui chiffrent bit par bit.

Le *Data Encryption Standard* DES [2] est un algorithme de chiffrement symétrique (chiffrement par bloc) utilisant des clés de 64 bits. Le premier standard DES est publié en 1977. Bien qu'il n'y ait pas d'attaque connue contre DES, son emploi n'est plus recommandé aujourd'hui, du fait de sa lenteur d'exécution et de son espace de clés trop petit, seuls 56 bits de la clé apportent de la sécurité.

L'AES (*Advanced Encryption Standard*) [3] est un standard de chiffrement symétrique qui de nos jours a remplacé le DES. Le développement de l'AES a été instigué par le NIST (*National Institue of Standards and Technology*) en 1997. C'est l'algorithme Rijndael, du

nom de ses deux concepteurs Vincent Rijmen et Joan Daemen, qui a été retenu, rebaptisé AES lors de la standardisation en 2001. L'AES utilise des clés allant de 128 à 256 bits ce qui est suffisant pour les sécurités visées aujourd'hui et sa vitesse d'exécution est satisfaisante. L'AES n'a pour l'instant pas été cassé, même théoriquement, au sens où il n'existe pas d'attaque significativement plus efficace que la recherche exhaustive quand le chiffrement est correctement utilisé.

1.1.3 Cryptographie asymétrique

Un chiffrement asymétrique utilise une paire de clés composée d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer. Les deux clés sont créées par une personne qui souhaite que lui soient envoyées des données confidentielles par les gens possédant sa clé publique, mais elle seule peut déchiffrer grâce à sa clé privée.

Définition 2 (Chiffrement asymétrique)**.** *Un schéma de chiffrement asymétrique est composé de quatre algorithmes.*

Configuration(1^λ) : à partir du paramètre de sécurité λ, génère les paramètres param du système.

GénérationClé(param) : à partir des paramètres du système, génère la clé publique pk et privée sk.

Chiffrement(pk,m) : génère un chiffré c du message m en utilisant la clé publique.

Déchiffrement(sk,c) : retourne le clair m correspondant au chiffré c.

Le chiffrement RSA (nommé par les initiales de ses trois inventeurs) est le premier algorithme de chiffrement asymétrique. Cet algorithme a été décrit en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman [4].

Ce cryptosystème fut une révolution dans le sens où il permit à tout le monde d'envoyer des messages chiffrés à une personne, sans que personne hormis le destinataire ne puisse déchiffrer.

1.1.4 Indistingabilité

Une propriété importante pour un chiffré est l'indistingabilité [5]. Un chiffrement possédant cette propriété permet d'envoyer des chiffrés sans que l'adversaire puisse apprendre d'information sur ce qui a été chiffré. Notamment un adversaire ne peut pas savoir si des chiffrés correspondent au même clair ou à des clairs différents.

Définition 3 (Fonction négligeable)**.** *Une fonction négligeable est une fonction qui minore asymptotiquement l'inverse de tout polynôme.*

Définition 4 (Indistingabilité face à une attaque à clairs choisis)**.** *Considérons le jeu suivant :*

- *Le challengeur génère (sk, pk) pour un paramètre λ et donne pk à l'adversaire.*
- *L'adversaire fait des opérations de son choix et soumet deux clairs distincts m_0 et m_1 au challengeur.*
- *Le challengeur sélectionne de manière uniformément aléatoire un bit b et envoie le challenge $c = Chiffrement(pk, m_b)$.*
- *L'adversaire fait des opérations de son choix et renvoie un bit b'.*
- *L'adversaire gagne si $b = b'$*

Un cryptosystème est dit indistingable pour une attaque à clairs choisis si tout adversaire faisant un nombre polynomial (en λ) d'opérations a un avantage négligeable (en λ). Un adversaire a un avantage négligeable si sa probabilité de gagner le jeu ne dépasse pas $1/2 + \epsilon(\lambda)$ où ϵ est une fonction négligeable.

1.2 Chiffrement homomorphe

Le chiffrement homomorphe permet de réaliser des opérations sur des données chiffrées. Cette propriété permet de confier des calculs à un agent externe, sans que les données ni les résultats ne soient accessibles à cet agent. Un client peut chiffrer avec une fonction homomorphe ses données. Il envoie les données chiffrées au serveur qui effectue des opérations directement sur les données chiffrées, sans passer par un déchiffrement. Le serveur renvoie le résultat au client qui lui seul peut déchiffrer.

1.2.1 Définition

Définition 5 (Schéma de chiffrement homomorphe)**.** *Soit un schéma de chiffrement (symétrique ou asymétrique). Le schéma est dit homomorphe s'il existe une opération $*$ sur l'espace ambiant des chiffrés et une opération $!$ sur l'espace des clairs telles que :*

$$Chiffrement_{\approx}(m_1) * Chiffrement_{\approx}(m_2) = Chiffrement_{\approx}(m_1 ! m_2)$$

pour toute clé et tous clairs m_1 et m_2, où $Chiffrement_{\approx}(m)$ correspond à un élément de l'espace ambiant des chiffrés, pour une clé donnée, qui se déchiffre en m avec une probabilité d'erreur :

- *négligeable en λ quand les éléments de l'espace de chiffrés utilisés pour faire l'opération $*$ sont issus de la fonction de chiffrement ;*
- *pouvant augmenter arbitrairement lors d'applications successives de $*$.*

Cette définition prend en compte les systèmes de chiffrement homomorphe permettant plus ou moins d'opérations avant de donner une erreur de déchiffrement, mais permettant au moins de réaliser une opération tout en gardant un déchiffrement (probabilistiquement) correct.

Il est commun d'utiliser la même notation pour l'opération dans l'espace des chiffrés et dans l'espace des clairs : $\text{Chiffrement}_{\approx}(m_1) * \text{Chiffrement}_{\approx}(m_2) = \text{Chiffrement}_{\approx}(m_1 * m_2)$, même si en général il ne s'agit pas de la même opération (ni du même espace). Un chiffrement est dit simplement homomorphe s'il permet de réaliser une opération sur l'espace des clairs. S'il permet de faire deux opérations différentes (en général on considère que ça doit être les opérations arithmétiques classiques d'addition et multiplication) et que la probabilité d'erreur au déchiffrement reste négligeable quel que soit le nombre d'opérations subies, on dit qu'il est complètement homomorphe.

Le concept de chiffrement complètement homomorphe a été décrit dès 1978, par Rivest, Adleman et Dertouzos [6]. Pourtant, le premier schéma (sûr) permettant de réaliser des multiplications et des additions n'a été découvert qu'en 2005 par Boneh Goh et Nissim [7], mais il ne pouvait évaluer que des polynômes de degré deux et sur un espace des clairs de petite taille. Quelques années plus tard, les premiers schémas permettant d'évaluer des polynômes de degré arbitraire ont vu le jour [8–13]. Parmi ces travaux, le schéma proposé par Gentry [9] a été le premier à permettre de réaliser un nombre d'additions et de multiplications illimité pour un paramétrage donné.

L'avancée rapide dans la recherche liée au chiffrement homomorphe a permis de repenser les possibilités pour sécuriser un système dans un environnement qui n'est pas de confiance. L'utilisation d'un schéma de chiffrement homomorphe permet d'évaluer une fonction arbitraire sur des données chiffrées [9, 10]. Depuis l'invention des premiers schémas, le domaine d'application du chiffrement homomorphe s'est étendu. Des prototypes d'applications existent pour le traitement du signal [14], les diagnostics de santé [15], les statistiques génomiques [16], les requêtes sur une base de données [17], le vote en ligne [18], etc.

Les schémas les plus prometteurs sont fondés sur les réseaux euclidiens [12, 19, 20].

1.2.2 Les réseaux euclidiens

Dans le reste du manuscrit nous noterons les vecteurs en gras et en minuscules. Les matrices seront notées en gras et majuscules. Pour un vecteur $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$ la norme euclidienne est notée $||v||$ et définie par $||v|| = \sqrt{\sum_{i=0}^{n} v_i^2}$. Pour deux vecteurs $\mathbf{v} = (v_1, \ldots, v_n)$ et $\mathbf{w} = (w_1, \ldots, w_n)$, on note $< \mathbf{v}, \mathbf{w} >= \sum_i v_i w_i$ leur produit scalaire. Pour deux entiers a, b, $a \wedge b$ correspond au pgcd des deux nombres. Enfin $\mathbb{Z}_q$ est une

notation simplifiée correspondant au quotient $\mathbb{Z}/q\mathbb{Z}$ (les entiers modulo q) et nous utiliserons de façon extensive les notation de Landau [21].

Les réseaux euclidiens sont les sous-groupes discrets de $\mathbb{R}^n$, n étant un entier positif. Bien qu'il soit possible de définir la notion de réseau de façon formelle directement, il est pratique de les définir en partant d'une base de vecteurs.

Définition 6 (Réseau euclidien)**.** *Soient $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ n vecteurs à m coordonnées dans $\mathbb{R}$ et $\mathbf{B} \in \mathbb{R}^{n \times m}$ la matrice ayant pour vecteurs colonnes les $\mathbf{b}_i$. On note $L(\mathbf{B})$ le sous-groupe discret de $(\mathbb{R}^m, +)$ engendré par les colonnes de B*

$$L(\mathbf{B}) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i | x_i \in \mathbb{Z}\} = \{\mathbf{Bx} | \mathbf{x} \in \mathbb{Z}^n\}$$

et on l'appelle le réseau euclidien de base $\mathbf{B}$, dimension m et rang n. Quand $m = n$ on dit qu'il est de rang plein.

On considérera dans ce manuscrit principalement des réseaux de rang plein. Il est important de noter qu'une base d'un réseau n'est pas unique. En effet, en multipliant la base par une matrice de $\mathbb{Z}^{n \times n}$ dont le déterminant vaut 1 on obtient une autre base du réseau.

La cryptographie fondée sur les réseaux base sa sécurité sur des problèmes difficiles sur les réseaux. Définissons d'abord la norme euclidienne puis la notion de plus petite norme dans un réseau. La norme minimale d'un réseau est un élément important pour les problèmes calculatoires des réseaux.

Définition 7 (Norme minimale)**.** *La distance minimale d'un réseau L est définie par*

$$\lambda_1(L) := \min(\{||v|| \mid v \in L \backslash \{0\}\}).$$

L'un des problèmes les plus fondamentaux de ce domaine est SVP (*Shortest Vector Problem*) qui consiste à trouver un vecteur de norme minimale pour un réseau. Le problème SVP_γ (*Approximate Shortest Vector Problem*) consiste à trouver un vecteur au plus γ fois plus grand que le plus petit vecteur. Le facteur multiplicatif γ est appelé le facteur d'approximation et peut dépendre de la dimension du réseau. Enfin, il est important également d'introduire la version décisionnelle de SVP_γ, le problème GapSVP_γ (*Decisional Approximate Shortest Vector Problem*), et SIVP_γ, équivalent de SVP_γ où il est demandé de trouver une base approximant la plus petite base à un facteur γ près. Nous concluons cette section par les définitions formelles de ces problèmes.

Définition 8 (*Shortest Vector Problem* SVP)**.** *Soit L un réseau euclidien. Trouver $z \in L$ tel que $\|z\| = \lambda_1(L)$.*

Définition 9 (*Approximate Shortest Vector Problem* SVP_γ)**.** *Soit L un réseau euclidien. Trouver $z \in L$ tel que $\|z\| \neq 0$ et $\|z\| \leq \gamma\lambda_1(L)$.*

Définition 10 (*Decisional Approximate Shortest Vector Problem* GapSVP_γ)**.** *Soit L un réseau euclidien avec $\lambda_1(L) \leq 1$ ou $\lambda_1(L) > \gamma$. Déterminer quel est le cas.*

Définition 11 (*Approximate Shortest Independent Vector Problem* SIVP_γ)**.** *Soit L un réseau euclidien, et $\mathbf{B}_{min}$ une base de ce réseau telle que $\|\mathbf{B}_{\min}\| \leq \|\mathbf{B}'\|$ pour toute autre base $\mathbf{B}'$ de L, en notant $\|\mathbf{B}\|$ le maximum des normes des vecteurs composant une base. Trouver une base $\mathbf{B}'$ telle que $\|\mathbf{B}'\| \leq \gamma\|\mathbf{B}_{min}\|$.*

1.2.3 Cryptographie fondée sur les réseaux

En 2005 Regev [22] a introduit le problème LWE (*Learning With Errors*), qui s'est révélé particulièrement utile pour construire des algorithmes de chiffrement.

Définition 12 (Distribution LWE)**.** *La distribution LWE est définie pour deux entiers q, n une distribution χ sur $\mathbb{Z}_q$, et un élément fixe $\mathbf{s} \in \mathbb{Z}_q^n$ par les couples $(\mathbf{a}, b)$ avec $\mathbf{a}$ choisi uniformément dans $\mathbb{Z}_q^n$ et $b =< \mathbf{a}, \mathbf{s} > +e$ avec e choisi dans $\mathbb{Z}_q$ en suivant la distribution χ.*

La valeur b est une combinaison linéaire des coordonnées de s dont le résultat est perturbé par le terme e qui est vu comme un bruit.

Le problème de recherche associé à une distribution LWE consiste à retrouver s en ayant un certain nombre d'échantillons suivant une loi LWE.

Définition 13 (Search-LWE)**.** *Soit une distribution LWE définie par des paramètres q, n, χ et un secret $\mathbf{s}$. Trouver $\mathbf{s}$ en utilisant un nombre polynomial en n d'échantillons issus de cette distribution.*

Le problème décisionnel associé à une distribution LWE consiste à distinguer les échantillons issus de cette distribution de ceux issus d'une distribution uniformément aléatoire.

Définition 14 (Decisional-LWE)**.** *Soit une distribution LWE définie par des paramètres q, n, χ et un secret $\mathbf{s}$. Décider à partir d'un nombre polynomial en n d'échantillons si ceux-ci viennent de la distribution LWE ou d'une distribution uniforme sur $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*

Ces deux problèmes LWE ont été réduits à des problèmes sur les réseaux euclidiens.

Théorème 1 ([22]). *Pour $q \leq 2^{poly(n)}$ et σ une distribution gaussienne discrète de paramètre $\alpha q \geq 2\sqrt{n}$ avec $\alpha < 1$, résoudre le problème Decisional-LWE est au moins aussi complexe que résoudre $GapSVP_\gamma$ et $SIVP_\gamma$ pour un réseau arbitraire de dimension n et $\gamma = \tilde{O}(n/\alpha)$.*

Il existe une version cyclique de LWE appelée RLWE (*Ring Learning With Errors*) [23]. RLWE est défini sur un anneau polynomial R avec des opérations sur les coordonnées qui sont définies modulo q, un entier positif, ce qui permet de définir l'anneau $R_q = R/qR$. De manière générale, R est un anneau cyclotomique pour faciliter les calculs. On note Φ_m, le m-ième polynôme cyclotomique usuel.

Définition 15 (Polynôme cyclotomique). *Soit m un entier. Le polynôme cyclotomique associé à cet entier m est le polynôme dont les racines complexes sont les racines primitives m-ièmes de l'unité*

$\Phi_m(x) = \prod_{k=1,k\wedge m=1}^{m} \left(X - e^{2ik\pi/m}\right)$.

La distribution RLWE est définie de manière similaire à la distribution LWE.

Définition 16 (Distribution RLWE). *La distribution RLWE est définie pour deux entiers q, m une distribution χ sur $R_q = R/qR$, avec $R = \mathbb{Z}[X]/\langle\Phi_m\rangle$, et pour un élément fixe $\mathbf{s} \in R_q$ par les couples $(\mathbf{a}, \mathbf{b})$ avec $\mathbf{a}$ choisi uniformément dans R_q et $\mathbf{b} = \mathbf{as} + \mathbf{e}$ avec $\mathbf{e}$ choisi dans R_q en suivant la distribution χ.*

Il est habituel de définir la distribution RLWE à partir de paramètres q, n avec n le degré de Φ_m pour un m donné. Ceci permet d'avoir des paramètres similaires pour RLWE et LWE car R_q est isomorphe à $\mathbb{Z}_q^n$. Nous ferons ceci dans la suite.

Il est bien sûr possible de définir des problèmes de recherche et décisionnel pour la distribution RLWE comme cela a été fait pour LWE.

Définition 17 (Search-RLWE). *Soit une distribution RLWE définie par des paramètres q, n, χ et un secret $\mathbf{s}$. Trouver $\mathbf{s}$ en utilisant un nombre polynomial en n d'échantillons issus de cette distribution.*

Définition 18 (Decisional-RLWE). *Soit une distribution RLWE définie par des paramètres q, n, χ et un secret $\mathbf{s}$. Décider à partir d'un nombre polynomial en n d'échantillons si ceux-ci viennent de la distribution RLWE ou d'une distribution uniforme sur $\mathbb{R}_q^2$.*

À nouveau ces problèmes peuvent être réduits à des problèmes dans les réseaux euclidiens.

Théorème 2 ([23], simplifié)**.** *Pour un entier n donné, l'anneau cyclotomique R sur $\mathbb{Z}$ de degré n, un choix approprié pour q et χ, résoudre le problème Decisional-RLWE est au moins aussi complexe que résoudre SVP_γ pour un réseau arbitraire sur R avec $\gamma = poly(n)$.*

1.2.4 Construction d'un schéma

Nous allons nous intéresser aux schémas homomorphes dont la sécurité est fondée sur RLWE. En 2011 Brakerski et Vaikuntanathan (BV) [13] ont introduit le premier de ces schémas, qui a été amélioré en 2012 par le schéma appelé de nos jours BGV [10]. Dans son expression la plus simple, le schéma BGV permet de chiffrer symétriquement un bit et de faire des additions ou multiplications modulo 2. Ici nous le décrirons pour un module p potentiellement supérieur à 2. Ce schéma se généralise facilement au cas asymétrique, ou pour des données sur plusieurs coordonnées.

Soit une distribution RLWE de paramètres q, n, χ pour un secret $\mathbf{s} \in R_q$, et un entier p. Un chiffré symétrique $\mathbf{c} \in R_q \times R_q$ de $m \in \mathbb{Z}_p$ est donné par $(\mathbf{a}, \mathbf{b})$ avec $\mathbf{a}$ tiré uniformément dans R_q, $\mathbf{b} = -\mathbf{as} + p\mathbf{e} + \mathbf{m}$, et $\mathbf{m}$ le polynôme constant de valeur m. Pour déchiffrer il suffit de calculer $\mathbf{b} + \mathbf{as}$ et de réduire le résultat modulo p. Tant que $p\mathbf{e}$ ne dépasse pas q sur aucune de ses coordonnées, le résultat sera m.

Les opérations homomorphes possibles sur le chiffré sont l'addition et la multiplication (plus une opération dérivée de l'addition qu'on appellera absorption). Dans les propositions qui suivent les opérations sont à comprendre dans R_q sauf pour celles sur les clairs où il sera bien rendu explicite que les opérations sont modulo p.

Nous ne prouverons les propositions qui suivent dans cette section, celles-ci étant des applications directes des propositions et preuves présentes dans [10].

Proposition 1 (Addition)**.** *Soient $\mathbf{c_1}$ et $\mathbf{c_2}$ deux chiffrés de m_1 et m_2, sous des bruits $\mathbf{e_1}$ et $\mathbf{e_2}$. L'élément $\mathbf{c_1} + \mathbf{c_2}$ est un chiffré de $m_1 + m_2 \mod p$ sous un bruit $p(\mathbf{e_1} + \mathbf{e_2})$.*

Il est possible qu'avec un grand nombre d'additions le bruit dépasse q ce qui rend le déchiffrement incorrect avec une grande probabilité. Il existe donc une limite maximum au nombre d'additions réalisable. Ceci étant dit, la croissance (en taille) du bruit est logarithmique, et il est donc possible de fixer un q tel que le nombre maximum d'additions ne soit pas un problème en pratique.

Il est bien sûr possible d'additionner un chiffré à lui même plusieurs fois. Souvent il existe une opération scalaire naturelle et rapide à calculer associée à l'addition qui permet d'éviter une addition itérative. Pour le cryptosystème décrit ici il s'agit de la

multiplication par un scalaire, et on dit que ce scalaire est *absorbé* par le chiffré quand on réalise cette opération.

Proposition 2 (Absorption)**.** *Soit* $\mathbf{c}$ *un chiffré de* m *sous un bruit* $\mathbf{e}$ *et* k *un scalaire. L'élément* $k \cdot \mathbf{c}$ *est un chiffré de* $k \cdot m \mod p$ *sous un bruit* $k \cdot p\mathbf{e}$*.*

Pour que le déchiffrement soit correct lors de l'absorption d'un scalaire k il faut bien évidemment avoir $\lceil \log_2 k \rceil$ bits de marge pour le bruit.

La dernière opération que nous traiterons est la multiplication, où les deux éléments à multiplier m_1 et m_2 sont chiffrés, contrairement à l'absorption (aussi appelée multiplication par un clair ou *plaintext multiplication*) où on multiplie un scalaire en clair k avec un chiffré de m.

Proposition 3 (Multiplication)**.** *Soient* $\mathbf{c_1} = (\mathbf{a_1}, \mathbf{b_1})$ *et* $\mathbf{c_2} = (\mathbf{a_2}, \mathbf{b_2})$ *deux chiffrés de* m_1 *et* m_2 *sous un bruit* $\mathbf{e_1}$ *et* $\mathbf{e_2}$*. En étendant l'espace des chiffrés aux triplets* $(\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3})$ *et en définissant un déchiffrement de ces triplets par* $\mathbf{v_3} + \mathbf{s}\mathbf{v_2} + \mathbf{s^2}\mathbf{v_1}$*, suivi d'une réduction modulo* p*, il est possible de calculer un chiffré de* $m_1 m_2 \mod p$ *avec*

$$(\mathbf{a_1 a_2}, \mathbf{a_1 b_2} + \mathbf{a_2 b_1}, \mathbf{b_1 b_2}).$$

Le bruit associé à ce chiffré est $p^2\mathbf{e_1 e_2} + m_1 p\mathbf{e_2} + m_2 p\mathbf{e_1}$*.*

Il est possible de définir la multiplication de façon plus générale par l'utilisation d'un produit tensoriel, mais dans ce manuscrit nous n'aurons pas besoin de cette description générique.

Il est important de noter que les schémas de chiffrement fondés sur RLWE disposent de deux opérations supplémentaires que nous détaillerons dans le chapitre suivant : relinéarisation et changement de module. La rélinéarisation permet de passer d'un chiffré à trois coordonnées vers un chiffré à deux coordonnées $(\mathbf{a}, \mathbf{b})$ qui se déchiffre tout simplement par $\mathbf{b} + \mathbf{as}$ puis une réduction modulo p. Ainsi il est habituel d'alterner les opérations de multiplication (qui font passer à trois coordonnées) et de relinéarisation (qui permettent de revenir sur deux coordonnées).

La réduction de module est utilisée pour réduire la norme du bruit. En effet un bruit de type $p^2\mathbf{e_1 e_2} + m_1 p\mathbf{e_2} + m_2 p\mathbf{e_1}$ a une magnitude deux fois plus importante que les bruits avant multiplication. Le changement de module permet de réduire ce bruit à quelque chose de proche du bruit en entrée de la multiplication par un remplacement du module q par un module plus petit q'. En alternant multiplications et changements de module on évite une croissance géométrique du bruit et on la remplace par une réduction linéaire du module q.

1.2.5 Représentation Double-CRT

Pour réduire les coûts calculatoires de la multiplication il est intéressant d'utiliser une représentation polynomiale particulière. Les outils de cette représentation sont aussi importants pour augmenter la quantité d'information transportée. Il est donc important de les présenter.

Une base CRT est un ℓ-uple de nombres premiers entre eux.

Définition 19 (Représentation CRT)**.** *Soit une base CRT* $(q_1, \ldots, q_\ell)$ *de dimension* ℓ *et* $q = \prod_{i=1}^{\ell} q_i$*. Un entier* $a \in \mathbb{Z}_q$ *en représentation CRT est donné par :*

$$CRT(a) = (a \bmod q_i)_{1 \leq i < \ell}\,.$$

Cette représentation est unique pour tout $a \in \mathbb{Z}_q$ par le théorème des restes chinois [24]. Elle est particulièrement utile pour réaliser des opérations sur des grands entiers car pour une base CRT dont le produit des premiers est q on a $\mathrm{CRT}(a) * \mathrm{CRT}(b) = \mathrm{CRT}(a * b \bmod q)$ où la première opération est un produit coordonnée à coordonnée. Ainsi avec une base ayant un nombre linéaire de premiers et en réalisant un nombre linéaire d'opérations il est possible de faire des produits modulo q qui auraient sinon eu un coût quadratique. Quand on réalise une multiplication dans l'espace des chiffrés, celle-ci sera donc beaucoup plus rapide si les coefficients des polynômes associés aux chiffrés sont en représentation CRT.

La *Number Theoretic Transform* (NTT), ou transformée de Walsh, permet de représenter un polynôme sous forme de valeurs en certains points au lieu de coefficients. Cette représentation permettra d'éliminer le coût quadratique (déjà éliminé pour les multiplication des scalaires par le CRT) dans la multiplication polynomiale.

Définition 20 (Représentation NTT)**.** *Soient* m, n *deux entiers tels que* Φ_m *soit de degré* n*. Soit* q *un entier tel que* $\mathbb{Z}/q\mathbb{Z}$ *contienne une racine* m^{eme} *de l'unité* ζ*. Soit* R_q *un anneau associé à une distribution RLWE de paramètres* q, n *(et donc isomorphe à* $\mathbb{Z}_q^n$*). La représentation NTT d'un polynôme* $\mathbf{a} \in R_q$ *est :*

$$NTT(\mathbf{a}) = \left(\mathbf{a}(\zeta^j)\right)_{j \in \mathbb{Z}_m^*}\,.$$

Quand un polynôme est représenté sous forme NTT avec ses coefficients sous forme CRT on parle de représentation DoubleCRT. [1]

1. Ce terme vient du fait que la représentation NTT peut être vue comme une représentation CRT ou le théorème des restes chinois est généralisé sur les polynômes facteurs de Φ_m modulo q.

Définition 21 (Représentation DoubleCRT)**.** *Soient m, n deux entiers tels que Φ_m soit de degré n. Soit q un entier tel que $\mathbb{Z}/q\mathbb{Z}$ contienne une racine m^{eme} de l'unité ζ. Soit R_q un anneau associé à une distribution RLWE de paramètres q, n (et donc isomorphe à $\mathbb{Z}_q^n$). Soit une base CRT $(q_1, \ldots, q_\ell)$ de dimension ℓ où $q = \prod_{i=1}^{\ell} q_i$. La représentation DoubleCRT d'un polynôme $\mathbf{a} \in R_q$ est donnée par :*

$$DoubleCRT(\mathbf{a}) = \big(\mathbf{a}(\zeta^j) \bmod p_i\big)_{0 \leq i < l,\ j \in \mathbb{Z}_m^*}.$$

1.2.6 Batching

Dans le schéma que nous avons présenté il est possible d'encoder dans un clair un vecteur dans $\mathbb{Z}_p^\ell$, en définissant le clair comme un polynôme sur $\mathbb{Z}_p$ (au lieu d'un scalaire) ayant ℓ valeurs en ℓ points fixés à l'avance (ce qui est réalisable par exemple par une interpolation). En chiffrant ce polynôme, les opérations homomorphes seront appliquées sur chacune des ℓ valeurs en parallèle. Cette fonctionnalité est le *batching*. Elle permet d'améliorer de manière significative les performances des opérations homomorphes. En effet, plusieurs milliers d'opérations peuvent être opérées en parallèle. On parle de *full batching* lorsque les opérations s'effectuent sur n valeurs pour un polynôme de degré $n - 1$.

1.2.7 Bootstrapping

Un schéma comme celui que nous avons décrit rajoutera du bruit au chiffré après une opération, ou réduira l'espace disponible pour celui-ci (lors d'une réduction de module). La profondeur calculatoire sera donc limitée à moins qu'une technique de *bootstrapping* [9] ne soit utilisée. Cette technique permet de réduire le bruit sans réduire pour autant l'espace disponible au delà de ce bruit pour faire des calculs supplémentaires. Il est donc possible de refaire augmenter le bruit par des calculs, puis de le réduire à nouveau par le *bootstrapping*. En faisant alternativement des calculs et des appels à cette technique il est donc possible de réaliser un nombre illimité d'opérations. L'idée derrière le *bootsrapping* est d'évaluer de façon homomorphe la fonction de déchiffrement à partir de chiffrés de la clé. L'approche précise est complexe et nous n'avons pas besoin de la décrire dans ce manuscrit. Il nous suffit juste de noter que cette opération est coûteuse et n'est en pratique pas utilisée dans de nombreuses applications du chiffrement homomorphe. En effet, d'un point de vue calculatoire, lorsqu'il faut réaliser des calculs complexes il est généralement plus intéressant d'augmenter la taille des paramètres (et ainsi la capacité de calcul) plutôt que de faire appel au *bootstrapping*.

1.3 Protection des choix lors d'un accès

Lorsqu'il est nécessaire dans un protocole d'accéder en lecture à un élément parmi plusieurs sur un serveur distant, il faut en général donner un index qui peut se révéler privé. Diverses techniques permettent de protéger le choix réalisé. Nous les présentons dans cette section.

1.3.1 PIR

Le retrait d'information privé (*Private Information Retrieval* PIR) est un protocole de type client-serveur. Il permet à un client de récupérer un élément dans une base de données distante sans révéler à cette dernière l'élément récupéré. Le PIR ne protège pas nécessairement la confidentialité des données stockées dans la base de données, il est possible pour le client de récupérer plus d'une donnée s'il le souhaite. Un protocole PIR qui garantit également que le client ne récupère qu'un seul élément est appelé *Symmetric PIR* ou SPIR.

Il est bien sûr possible de réaliser un PIR naïf en téléchargeant tout le contenu de la base de données puis en choisissant localement de ne lire que l'élément souhaité. Le premier PIR avec un coût sous-linéaire (par rapport à la taille de la base de données) a été introduit par Chor, Goldreich, Kushilevitz, and Sudan en 1995[25]. Ils proposent une manière de réaliser le PIR en utilisant des bases de données répliquées ne communiquant pas entre elles. La sécurité de ce PIR repose sur la théorie de l'information (autrement dit, il est sûr même face à un attaquant ayant une capacité de calcul illimitée).

Il est possible de définir un protocole PIR avec un coût de communication sous-linéaire en la base de donnée et sans que celle-ci soit répliquée au prix de ne résister qu'aux attaquants ayant une capacité calculatoire limitée. On appelle ceux-ci les protocoles cPIR pour *computationally-Private Information Retrieval* [26]. Une manière simple d'obtenir un tel protocole est d'utiliser un système de chiffrement homomorphe ayant la propriété d'indistingabilité face aux attaques à clair choisi.

Absorptions

Avant de décrire un protocole cPIR commençons par décrire comment absorber un clair de taille arbitraire dans un chiffrement homomorphe disposant d'une fonction d'absorption. Supposons que cette fonction prend en entrée un entier k et un chiffré $\mathbf{c}$ associé à un message m et retourne $\text{Absorption}(k, \mathbf{c})$, un chiffré de $k \cdot m \mod p$ pour un entier p donné. Le comportement d'une telle fonction est remarquable pour les valeurs

Algorithme 1 : Absorption$(e, \mathbf{c}, A)$

Input : e élément d'une base de taille l, $\mathbf{c}$ chiffré, A capacité d'absorption d'un chiffré
Output : Vecteur de chiffrés $\mathbf{C}$ correspondant à l'absorption de e par $\mathbf{c}$
1 Découper e en entiers de A bits $e_1, \ldots, e_{\lceil l/A \rceil}$ (en ajoutant un padding si l/A n'est pas un entier)
2 Retourner $\mathbf{C} = (\text{Absorption}(e_1, \mathbf{c}), \ldots, \text{Absorption}(e_{\lceil l/A \rceil}, \mathbf{c}))$

$m = 1$ et $m = 0$. En effet, si $m = 1$ le résultat de l'absorption sera un chiffré de $k \mod p$ (on dira que k a été retenu) alors que si $m = 0$ le résultat sera un chiffré de 0 (on dira que k a été effacé). Pour que k soit correctement retenu il faut que $k < p$ où sinon il y aura une perte d'information à cause du modulo p. Ainsi nous définissons pour cette fonction une borne supérieure pour l'absorption A que par convenance nous définirons en nombre de bits. Enfin il est important de noter que l'on peut définir $A < \log_2 p$ pour d'autres raisons, en particulier à cause d'une capacité de calcul homomorphe limitée (comme pour le schéma décrit dans les sections précédentes où on disposait d'un espace limité pour le bruit).

Si un chiffré ne permet pas d'absorber entièrement un élément de la base de données (car l'élément est trop grand en taille et provoquera une erreur lors du déchiffrement), alors le chiffré envoyé par le client est réutilisé pour absorber l'élément par morceaux en générant ainsi une suite de chiffrés ayant absorbé peu à peu l'élément complet. L'algorithme 1 montre comment réaliser l'absorption d'éléments plus grands que la capacité d'absorption.

Protocole simple

Soit une base de données ayant L éléments de l bits chacun. Un client crée une requête constituée de $L - 1$ chiffrés de 0 et un chiffré de 1. Les chiffrés sont ordonnés et le chiffré de 1 est à l'index correspond à l'élément que le client veut récupérer. Il envoie la requête au serveur. Grâce à la propriété d'indistingabilité le serveur ne peut pas savoir à quelle position correspond le chiffré de 1. Le serveur réalise L absorptions entre chaque élément de la base de données et chaque élément de la requête. Puis il va sommer le résultat de toutes les absorptions. Seulement l'élément de la base de données multipliant le chiffré de 1 sera retenu, les autres seront effacés, et donc la réponse contiendra uniquement l'élément souhaité. Un exemple pour une base de données à six éléments est donné dans la figure 1.1 et une description formelle des algorithmes associés dans la figure 1.2.

Agrégation

Le protocole décrit a une taille linéaire en L (pour la requête) et linéaire en l (pour la réponse). Les communications sont moindres qu'avec un téléchargement total qui a

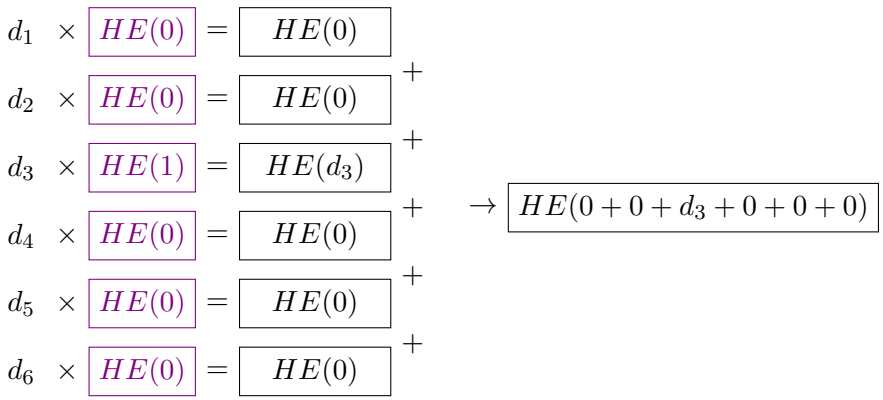

FIGURE 1.1 Génération d'une réponse PIR. Le client envoie un chiffré par élément dans la base de données. Le chiffré de 1 correspond à l'élément qui intéresse le client. La réponse contient un chiffré de d_3 mais il n'est pas possible de savoir ceci à partir de la requête car les chiffrés de 0 et de 1 sont indistingables. Il est important de remarquer que bien que tous les chiffrés de 0 soient notés $HE(0)$ ce sont tous des chiffrés différents appartenant à l'ensemble des chiffrés possibles de 0.

un coût en $L \times l$ si l est plus grand que la taille d'un chiffré. Les bases de données avec L grand et l petit sont courantes (e.g. bases de données du registre civil). Il est donc souvent intéressant de réduire la taille de la requête pour qu'elle représente moins de L chiffrés. Pour cela, il existe deux améliorations : l'agrégation et la récursion.

L'agrégation consiste à fusionner des éléments d'une base et d'obtenir par conséquent une base de données avec moins d'éléments plus grands. Prenons par exemple le cas où la taille des éléments est inférieure à A. Au lieu d'envoyer une requête de L chiffrés et recevoir un seul chiffré en réponse (puisque $l < A$), on peut agréger les éléments par groupes de $\sqrt{L}$ éléments (on considérera que ce nombre est un entier, sinon il est possible de l'arrondir). Grâce à cette agrégation notre requête ne fera plus que $\sqrt{L}$ chiffrés et la réponse sera au plus de $\sqrt{L}$ chiffrés (voir significativement inférieure si $l \ll A$ et il est possible d'envoyer plusieurs éléments agrégés dans chaque chiffré de la réponse). Ainsi on passe d'un protocole sans agrégation où on envoie $L + 1$ chiffrés à un protocole où on envoie au plus $2\sqrt{L}$ chiffrés.

Il est important de noter que l'agrégation révèle plus d'information que nécessaire au client. En effet, le client ne recevra pas uniquement l'élément voulu mais également tous les éléments qui lui ont été agrégés.

Algorithme 2 : Génération de requête cPIR

Input : Nombre d'éléments dans la base de données L, index i_0 de l'élément à obtenir, sk clé utilisée pour chiffrer

Output : Vecteur de chiffrés Q formant la requête

1 **for** $i \in [1 \dots L]$ **do**

2 $\quad Q_i = \text{Chiffrement}(sk, 1)$ si $i = i_0$; $Q_i = \text{Chiffrement}(sk, 0)$ sinon ;

3 Retourner $Q = (Q_1, \dots, Q_L)$;

Algorithme 3 : Génération de réponse cPIR

Input : Éléments de la base de données $(e_1, \dots, e_L)$, requête $Q = (Q_1, \dots, Q_L)$, capacité d'absorption des chiffrés A

Output : Vecteur de chiffrés R formant la réponse

1 Retourner $R = \sum_{i=1}^{L} \text{Absorption}(e_i, Q_i, A)$

Algorithme 4 : Extraction de la réponse cPIR

Input : Réponse PIR R formée des chiffrés $R_1, \dots, R_{\lceil l/A \rceil}$, clé de déchiffrement sk, capacité d'absorption des chiffrés A

Output : Élément de la base de données e

1 **for** $i \in [1 \dots \lceil l/A \rceil]$ **do**

2 $\quad \text{partie}_i = \text{Déchiffrement}(sk, R_i)$;

3 Concaténer les partie_i comme des éléments de A bits en éliminant le padding du dernier le cas échéant ;

4 Retourner le résultat ;

FIGURE 1.2 Algorithmes formant un PIR calculatoire fondé sur un chiffrement homomorphe.

Récursion

Une deuxième approche, complémentaire, pour améliorer le protocole simple est d'utiliser la récursion.

L'idée est de remplacer une base de données à L éléments par $\sqrt{L}$ bases de données à $\sqrt{L}$ éléments (nous supposerons pour simplifier que $\sqrt{L}$ est un entier). Si l'élément que veut obtenir le client est à l'indice i_0 de la base d'indice i_1 parmi les $\sqrt{L}$ bases, il envoie une requête $Q^{(1)}$ composée de $\sqrt{L}$ chiffrés où celui d'index i_0 seulement est un chiffré de 1 ainsi qu'une requête $Q^{(2)}$ composée de $\sqrt{L}$ chiffrés où celui d'index i_1 seulement est un chiffré de 1. La serveur génère une réponse PIR pour chacune des $\sqrt{L}$ bases en utilisant la requête $Q^{(1)}$ à chaque fois. Ensuite il génère une réponse PIR finale en utilisant $Q^{(2)}$ sur la liste de $\sqrt{L}$ réponses PIR intermédiaires. Il n'envoie que cette réponse finale. Le client obtiendra le bon élément car la première requête lui a permis d'obtenir des chiffrés des éléments d'index i_0 de chaque base, et la deuxième requête lui aura permis de choisir secrètement lequel de ces chiffrés l'intéressait. La récursion est illustrée dans la figure 1.3 avec une requête $Q^{(1)}$ de trois éléments utilisés trois fois, puis une requête $Q^{(2)}$ utilisée une fois sur les trois réponses intermédiaires.

Il est possible de généraliser ce mécanisme et d'envoyer une requête pour d récursions successives. Dans ce cas il faudra envoyer d requêtes chacune composée de $L^{1/d}$ chiffrés (on simplifie à nouveau en supposant que $L^{1/d}$ est un entier). Inversement, comme l'illustre bien la figure 1.3, les réponses issues du premier niveau de traitement sont des chiffrés homomorphes, puis les réponses issues du deuxième niveau de traitement sont des chiffrés de chiffrés, etc. Quand on réalise d récursions successives, la réponse a d couches de chiffrement. Or, chaque couche de chiffrement agrandit la réponse d'un facteur multiplicatif F (par exemple 6 dans la librairie de référence XPIR [27]). Ainsi la requête évolue en $d \cdot L^{1/d}$ et la réponse en F^d, il faut donc trouver un équilibre.

Utilisation d'un chiffrement additif et multiplicatif

Il est possible de réaliser un PIR extrêmement efficace en termes de communication en utilisant un chiffrement homomorphe additif et multiplicatif. Pour cela il suffit d'envoyer des chiffrés $\beta_1, \ldots, \beta_\eta$ des bits $b_1, \ldots, b_\eta$ de l'index i_0 nous intéressant pour $\eta = \lceil \log_2 L \rceil$ et un chiffré de 1 noté γ. Si on utilise un chiffrement homomorphe à clé publique, il n'est pas nécessaire d'envoyer un chiffré de 1 car le serveur peut l'obtenir lui même. Pour traiter un élément d'indice i le serveur va considérer les bits $b'_1, \ldots, b'_\eta$ de cet indice et va calculer les produits homomorphes des β_i quand $b'_i = 1$ et des chiffrés $\gamma - \beta_i$ quand $b'_i = 0$. Il est facile de voir que ceci ne donnera un chiffré de 1 que si l'index considéré est i_0. Une

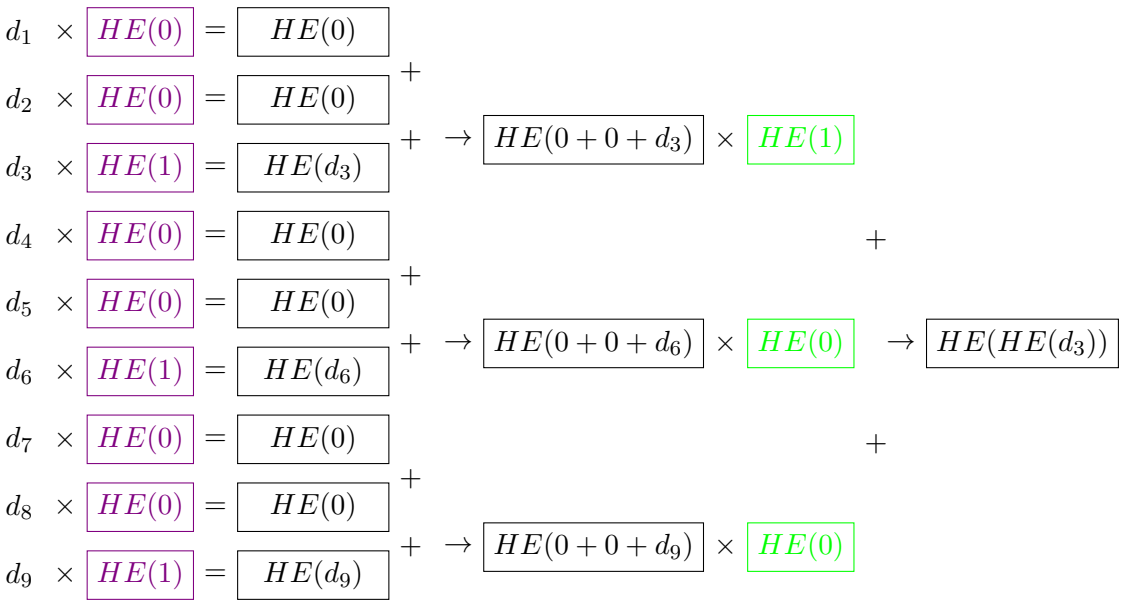

FIGURE 1.3 Protocole PIR récursif. La première requête, en violet, est composée de 3 éléments, réutilisés 3 fois. La deuxième requête est composée de trois éléments aussi. Ensemble elles permettent de récupérer un élément parmi neuf.

fois il a obtenu le produit des chiffrés correspondant à l'indice i il fait une absorption du i-ème élément comme dans un PIR classique et il renvoie la somme des résultats.

D'un point de vue des communications ce PIR sera extrêmement efficace. En effet il suffit d'envoyer $\lceil \log L \rceil$ chiffrés, et pour les meilleurs systèmes de chiffrement homomorphe actuels la taille du chiffré pouvant réaliser de tels produits va grandir en polylog$(\log L)$. Malheureusement d'un point de vue calculatoire il faudra réaliser $\log L$ produits homomorphes pour chacun des L éléments de la base ce qui sera extrêmement coûteux, et en pratique beaucoup plus long que d'envoyer toute la base par le réseau. En pratique il est donc souvent préférable de se contenter des opérations homomorphes additives quand on réalise un PIR.

1.3.2 OT

Un *Oblivious Transfer* (OT) est un protocole qui permet d'envoyer une donnée sans que l'expéditeur n'apprenne quelles informations ont été transmises. Contrairement aux protocoles PIR ces protocoles sont toujours symétriques, ils doivent garantir que l'utilisateur ne peut apprendre qu'une des données de l'expéditeur par transaction. Contrairement encore aux protocoles PIR, le coût des communications n'est pas un critère prioritaire et généralement on considère que la base entière (chiffrée) est envoyée

à l'utilisateur. Les objectifs de l'OT sont de donner des garanties le plus fortes possibles de sécurité (idéalement fondées sur les principes de la théorie de l'information) et de minimiser les coûts calculatoires.

Le premier schéma d'OT a été proposé en 1981 par Michael O. Rabin [28] mais la forme la plus utilisée des schémas OT correspond à un schéma ultérieur publié par Even et al. [29] en 1985, le 1-2 OT ou *1-out-of-2 oblivious transfer*. Dans ce protocole on envoie 2 éléments chiffrés et le destinataire ne peut en déchiffrer qu'un des deux.

Définition 22 (*1 out of 2 Oblivious Transfer* 1-2 OT)**.** *Un schéma d'1-2 OT est donné par deux algorithmes interactifs : l'expéditeur S et le destinataire R. L'expéditeur prend en entrée deux messages m_0 et m_1, le destinataire un bit b. Le destinataire reçoit m_b mais n'apprend pas d'information sur $m_{b+1 \bmod 2}$. Les interactions pour $b = 0$ et $b = 1$ sont indistingables. L'expéditeur n'apprend pas d'information sur b et ne sait donc pas quel message le destinataire a reçu.*

L'OT a été généralisé à une situation où le destinataire choisit parmi n éléments. L'utilisateur récupère toujours exactement un élément d'une base de données sans que le serveur apprenne quel élément il a récupéré et sans que l'utilisateur n'apprenne les autres éléments [30].

Les protocoles d'OT n'utilisent pas forcément le chiffrement homomorphe et nous les utiliserons dans ce manuscrit en boîte noire uniquement leur fonctionnement interne n'ayant pas d'impact sur nos protocoles.

1.4 Plan

Dans le chapitre 2 nous nous intéresserons aux librairies de référence permettant de réaliser du chiffrement homomorphe. Les librairies SEAL, FV-NFLlib et HElib seront présentées et les performances de ces librairies seront étudiées pour différents paramétrages. L'étude de ces librairies nous permettra de choisir, dans les chapitres ultérieurs, la librairie donnant des résultats optimaux dans chaque situation. Les schémas utilisés par les librairies seront détaillés et les impacts de certains choix algorithmiques sur les performances analysés.

Dans le chapitre 3 nous proposerons des protocoles améliorant la protection de la vie privée dans le contexte de l'utilisation des données génomiques. Ces données étant fortement privatives, il est important de les protéger. Nous présenterons un algorithme qui permet d'externaliser les données génomiques chiffrées dans une base de données tout en laissant la possibilité de tester à distance, efficacement et de manière privée, la présence d'un élément donné.

Dans le chapitre 4 nous étudierons une solution permettant de calculer un ensemble maximal de manière privée entre un élément et une base de donnée. Ce calcul est souvent utilisé pour connaître la corrélation entre un génome individuel et les génomes d'une base de données.

Pour finir, dans le chapitre 5 nous verrons un algorithme permettant de détecter une fraude lors d'un paiement en ligne en utilisant le calcul homomorphe et un algorithme s'apparentant à un *Oblivious Transfer*. La librairie SEAL a été utilisée pour implémenter l'algorithme.

Chapitre 2

Analyse des performances de SEAL, FV-NFLlib, HElib

Les algorithmes de chiffrement homomorphe se sont considérablement améliorés au cours de ces dix dernières années. La découverte de schémas plus efficaces a permis de réaliser des applications intéressantes à un coût de calcul acceptable. Plusieurs implémentations de ces schémas ont été rendues publiques, permettant aux chercheurs et acteurs du chiffrement homomorphe de mieux comprendre les propriétés liées à la performance de ces schémas. De nouvelles applications utilisant le chiffrement homomorphe ont ainsi pu être développées.

La plupart des applications utilisant le chiffrement homomorphe se basent sur une représentation des données en binaire et construisent un circuit booléen pour évaluer la fonction désirée sur les données binaires chiffrées. Les librairies telles que HElib [31, 32], SEAL [33] ou FV-NFLlib [34] sont adaptées à une telle représentation des données en binaire. Cependant il est également possible d'envisager d'autres représentations, avec un domaine des messages clairs plus grand, pour des situations où les fonctions sont représentées par un circuit arithmétique et potentiellement évaluées plus efficacement. Or, il semble qu'une telle représentation des données ne soit pas l'objectif premier de ces librairies et la question de ce qu'il est possible de faire, ou non, dans ce domaine avec celles-ci est une question ouverte.

Pouvoir représenter les données dans un domaine de clairs de taille plus importante ouvrirait pourtant la voie à de nombreuses et prometteuses applications. Cela pourrait, par exemple, être utile pour le calcul de précision sur des données à virgule fixe ou pour permettre des opérations liées au logarithme discret ou à la factorisation de grands nombres pour de l'externalisation de données.

Le travail présenté dans ce chapitre consiste en une étude comparative des performances des principales librairies de chiffrement homomorphe dans ce contexte, avec une taille de clairs allant jusqu'à 2048 bits, des schémas de chiffrement partiellement homomorphes où l'opération limitante est la multiplication. L'objectif principal de ce travail est de proposer un guide de sélection des librairies et des paramètres cryptographiques en fonction des besoins d'une application en termes de taille du domaine des données claires et du nombre d'opérations homomorphes nécessaires, ainsi que de mieux identifier et comprendre les limites, en ce domaine, de chacune de ces librairies.

2.1 Librairies et schémas de chiffrement homomorphe

Dans cette section, nous présentons les schémas de chiffrement homomorphes implémentés dans les librairies que nous étudions : BGV, utilisé par HElib, et FV, utilisé par FV-NFLlib et SEAL.

2.1.1 HElib, basée sur BGV

La librarie HElib est basée sur une variante de BGV [12]. Elle est définie sur l'anneau des polynômes de la forme $R = \mathbb{Z}[x]/\Phi_m(x)$, où m est un paramètre et Φ_m est le $m^{ème}$ polynôme cyclotomique. Pour simplifier, nous allons considérer m de la forme $m = 2 * n$ avec n une puissance de 2. Cela implique que $\Phi_m(x) = X^n + 1$. L'espace des clairs est l'anneau $R_p = R/pR$ pour un entier p.

Un polynôme clair $\mathbf{a} \in R_p$ est chiffré comme un vecteur sur $R_q = R/qR$, où q est un module public. Plus spécifiquement, BGV définit q comme étant une chaîne de modules de taille décroissante : $q_0 > q_1 > \cdots > q_L$. Lorsque l'on chiffre un polynôme, le chiffré est en premier lieu définit modulo q_0. Après chaque multiplication homomorphe, on procède à une commutation de module, du module courant vers le module suivant, plus petit. Il en va ainsi jusqu'à arriver au module q_L, le stock de modules est épuisé et le chiffré ne peut alors plus être multiplié. L est donc la profondeur multiplicative maximale du circuit.

Notons q_i un module quelconque entre $\{q_0, \ldots, q_L\}$. La variante de BGV dans HElib utilise également un autre ensemble de nombres premiers dont le produit est q' et qui sont utilisés afin de limiter le bruit introduit lors de la relinéarisation. Le bruit suit une distribution gaussienne $D_{\mathbb{Z}^n,\sigma}$ d'écart-type σ sur le réseau euclidien $\mathbb{Z}^n$. En plus des paramètres habituels pour la génération des clés (degré n, variation σ, module des clairs p, module des chiffrés q), il est aussi possible de choisir un paramètre de relinéarisation ℓ

qui offre un compromis entre la croissance du bruit et le coût de l'opération. En pratique ce paramètre vaut toujours 3 dans HElib.

Le schéma global de la variante de BGV utilisée dans HElib est détaillé ci-dessous.

- HElib.GenClé :
  - sk Tire un secret aléatoire $\mathsf{sk} := \mathbf{s} \leftarrow R_{q_0}$ avec les coefficients dans $\{-1, 0, 1\}$, dont exactement h valent 0.
  - pk Génère une clé publique $\mathsf{pk} := (\mathbf{b}, \mathbf{a}) \in R_{q_0}^2$, avec $\mathbf{a} \leftarrow R_{q_0}$ tiré aléatoirement uniforme, et $\mathbf{b} := p\mathbf{e} - \mathbf{a} \cdot \mathbf{s}$, où $\mathbf{e} \leftarrow R_{q_0}$ suit $D_{\mathbb{Z}^n,\sigma}$.
  - rk Génère une clé de relinéarisation dans $R_{q_0 \cdot q'}^{2\times\ell}$. Divise q en ℓ facteurs de même taille $B_1, \ldots, B_\ell$. Définit $\mathsf{rk}_i := (\mathbf{a_i}, \mathbf{b_i})^t \in R_{q_0 q'}^2$, avec $\mathbf{a_i} \leftarrow R_{q_0 q'}$ tiré uniformément et $\mathbf{b_i} := \left(\prod_{j=0}^{i-1} B_j\right) \mathbf{s}^2 + p\mathbf{e_i} - \mathbf{a_i} \cdot \mathbf{s}$, où $\mathbf{e_i} \leftarrow R_{q_0 q'}$ suivent $D_{\mathbb{Z}^n,\sigma}$. Sortie $\mathsf{rk} := (\mathsf{rk}_1, \ldots, \mathsf{rk}_\ell)$.

- HElib.Chiffrement$(\mathsf{pk}, \boldsymbol{\mu})$: Génère un nouveau chiffré $\mathsf{ct} \in R_{q_0}^2$ à partir d'un clair $\boldsymbol{\mu} \in R_p$, chiffré en utilisant la clé $\mathsf{pk} := (\mathbf{b}, \mathbf{a}) \in R_{q_0}^2$. Nous avons $\mathsf{ct} := (\mathbf{c_0}, \mathbf{c_1})$ avec $\mathbf{c_0} := \mathbf{u} \cdot \mathbf{b} + p\mathbf{e_0} + [q_0\boldsymbol{\mu}]_p$ et $\mathbf{c_1} := \mathbf{u} \cdot \mathbf{a} + p\mathbf{e_1}$, où $\mathbf{u} \leftarrow R_{q_0}$ est tiré uniformément dans $\{-1, 0, 1\}^n$ et $\mathbf{e_0}, \mathbf{e_1}$ suivent $D_{\mathbb{Z}^n,\sigma}$.

- HElib.Addition$(\mathsf{ct}_0, \mathsf{ct}_1)$: Additionne deux chiffrés $\mathsf{ct}_0 := (\mathbf{c_{00}}, \mathbf{c_{01}}) \in R_{q_i}^2$ et $\mathsf{ct}_1 := (\mathbf{c_{10}}, \mathbf{c_{11}}) \in R_{q_i}^2$ en un chiffré $\mathsf{ct}_+ := (\mathbf{c_0}, \mathbf{c_1}) \in R_{q_i}^2$, avec $\mathbf{c_0} = \mathbf{c_{00}} + \mathbf{c_{10}}$ et $\mathbf{c_1} = \mathbf{c_{01}} + \mathbf{c_{11}}$.

- HElib.Multiplication$(\mathsf{ct}_0, \mathsf{ct}_1)$: Multiplie deux chiffrés $\mathsf{ct}_0 := (\mathbf{c_{00}}, \mathbf{c_{01}}) \in R_{q_i}^2$ et $\mathsf{ct}_1 := (\mathbf{c_{10}}, \mathbf{c_{11}}) \in R_{q_i}^2$ en un chiffré $\widetilde{\mathsf{ct}}_\times := (\mathbf{c_0}, \mathbf{c_1}, \mathbf{c_2}) \in R_{q_i}^3$, avec $\mathbf{c_0} = \left[q_i^{-1}\right]_p \mathbf{c_{00}} \cdot \mathbf{c_{10}}$, $\mathbf{c_1} = \left[q_i^{-1}\right]_p (\mathbf{c_{00}} \cdot \mathbf{c_{11}} + \mathbf{c_{01}} \cdot \mathbf{c_{10}})$ et $\mathbf{c_2} = \left[q_i^{-1}\right]_p \mathbf{c_{01}} \cdot \mathbf{c_{11}}$.

- HElib.ChangementModule(ct, q) : Retire des premiers du module courant pour en obtenir un nouveau q et réduire le chiffré ct par un facteur Δ
  1. Réduit les coefficients de ct pour obtenir $\mathsf{ct}' = \mathsf{ct} \bmod \Delta$,
  2. Ajoute ou soustrait des multiples de Δ à chaque coefficient de ct' jusqu'à ce qu'il soit divisible par p,
  3. $\mathsf{ct}^* = \mathsf{ct} - \mathsf{ct}'$, // ct^* divisible par Δ, et $\mathsf{ct}^* \equiv \mathsf{ct} \pmod p$
  4. Sortie ct^*/Δ.

- HElib.Relin$(\mathsf{rk}, \widetilde{\mathsf{ct}}_\times)$: Relinéarise un chiffré $\widetilde{\mathsf{ct}}_\times := (\mathbf{c_0}, \mathbf{c_1}, \mathbf{c_2}) \in R_{q_i}^3$ en un chiffré

$\mathsf{ct}_\times \in R_{q_i}^2$ en utilisant la clé de relinéarisation $\mathsf{rk} := W \in R_{q_0 q'}^{2\times\ell}$. Dans une première étape $\mathbf{c_2}$ est divisé en ℓ polynômes de plus petite norme $\mathbf{c_2}^{(i)}$:

1. $\mathbf{d_1} \leftarrow \mathbf{c_2}$
2. Pour $i \leftarrow 1, \ldots, \ell$:
3. $\quad \mathbf{c_2}^{(i)} \leftarrow \mathbf{d_i} \bmod B_i$
4. $\quad \mathbf{d_{i+1}} \leftarrow \left(\mathbf{d_i} - \mathbf{c_2}^{(i)}\right)/B_i$

La matrice représentant la clé de relinéarisation est réduite modulo $q_i q'$, on ajoute tous les premiers spéciaux correspondants à $q_i q'$ à tous les $\mathbf{c_2}^{(i)}$, puis on calcule le chiffré

$$\overline{\mathsf{ct}}_\times := \left(\mathbf{c_0} + \sum_{i=1}^{\ell} W_{i,0} \cdot \mathbf{c_2}^{(i)},\ \mathbf{c_1} + \sum_{i=1}^{\ell} W_{i,1} \cdot \mathbf{c_2}^{(i)}\right) \in R_{q_i q'}^2$$

Finalement en utilisant la fonction de changement de module, on obtient $\mathsf{ct}_\times := \mathsf{HElib.ModSwitch}(\overline{\mathsf{ct}}_\times, q_i) \in R_{q_i}^2$.

- $\mathsf{HElib.Déchiffrement}(\mathsf{sk}, \mathsf{ct})$: Déchiffre un chiffré $\mathsf{ct} := (\mathbf{c_0}, \mathbf{c_1}) \in R_{q_i}^2$ en un clair $\boldsymbol{\mu} := \left[\left[q_i^{-1}\right]_p \left[\mathbf{c_0} + \mathbf{c_1} \cdot \mathbf{s}\right]_{q_i}\right]_p \in R_p$.

2.1.2 Schéma de Fan et Vercauteren

Le schéma de Fan et Vercauteren [20] (FV) est proche de celui de BGV. La différence principale réside dans l'utilisation d'une échelle invariante pour limiter l'accroissement du bruit plutôt qu'un changement de module avec BGV. De plus, il place les bits utiles sur les bits de poids fort de chaque coefficient du polynôme contrairement à BGV.

Il est possible de choisir un paramètre pour la relinéarisation noté ω qui représente une base dans laquelle la clé de relinéarisation est décomposée. Le choix d'ω a un impact sensible sur les performances. Deux distributions différentes sont utilisées : χ_{key} et χ_{err}. La librairie FV-NFLlib définit $\chi_{\mathsf{key}} = D_{\mathbb{Z}^n, \sigma_{\mathsf{key}}}$ et $\chi_{\mathsf{err}} = D_{\mathbb{Z}^n, \sigma_{\mathsf{err}}}$, pour σ_{key} et un σ_{err} donnés. La librairie SEAL, de son côté, définit χ_{key} comme la distribution uniforme sur $\{-1, 0, 1\}$ et χ_{err} de façon similaire à FV-NFLlib.

Le schéma global de FV utilisé dans FV-NFLlib et dans SEAL est détaillé ci-dessous.

- $\mathsf{FV.GénérationClé}$:

  sk Soit $\mathsf{sk} := \mathbf{s} \leftarrow \chi_{\mathsf{key}}$

  pk Soit $\mathbf{a} \leftarrow R_q$ tiré uniformément et $\mathbf{e} \leftarrow \chi_{\mathsf{err}}$.
  Soit $\mathsf{pk} := ([-\mathbf{a} \cdot \mathbf{s} - \mathbf{e}]_q, \mathbf{a})$.

rk Génère une clé de relinéarisation $\mathsf{rk} := (\mathsf{rk_1}, \ldots, \mathsf{rk}_\ell)$ avec $\ell = \lfloor \log_\omega(q) \rfloor + 1$ et $\mathsf{rk_i} := (\mathbf{s^2}\omega^i - (\mathbf{a_i} \cdot \mathbf{s} + \mathbf{e_i}), \mathbf{a_i})$, avec $\mathbf{a_i} \leftarrow R_q$ uniformément aléatoire et $\mathbf{e_i} \leftarrow \chi_{\mathsf{err}}$.

- $\mathsf{FV.Chiffrement}(\mathsf{pk}, \mu)$: Génère un nouveau chiffré $\mathsf{ct} \in R_q^2$ à partir d'un clair $\boldsymbol{\mu} \in R_p$, en utilisant la clé de chiffrement $\mathsf{pk} := (\mathbf{b}, \mathbf{a}) \in R_q^2$. Nous avons $\mathsf{ct} := (\mathbf{c_0}, \mathbf{c_1})$ avec $\mathbf{c_0} := \mathbf{u} \cdot \mathbf{b} + \mathbf{e_0} + \Delta[\mu]_p$ et $\mathbf{c_1} := \mathbf{u} \cdot \mathbf{a} + \mathbf{e_1}$, où $\mathbf{u}$ suit χ_{key} et $\mathbf{e_0}, \mathbf{e_1}$ suivent χ_{err} et $\Delta = q/p$.

- $\mathsf{FV.Addition}(\mathsf{ct}_0, \mathsf{ct}_1)$: Additionne deux chiffrés $\mathsf{ct}_0 := (\mathbf{c_{00}}, \mathbf{c_{01}}) \in R_q^2$ et $\mathsf{ct}_1 := (\mathbf{c_{10}}, \mathbf{c_{11}}) \in R_q^2$ en un chiffré $\widetilde{\mathsf{ct}}_+ := (\mathbf{c_0}, \mathbf{c_1}) \in R_q^2$, avec $\mathbf{c_0} = \mathbf{c_{00}} + \mathbf{c_{10}}$ et $\mathbf{c_1} = \mathbf{c_{01}} + \mathbf{c_{11}}$.

- $\mathsf{FV.Multiplication}(\mathsf{ct}_0, \mathsf{ct}_1)$: Multiplie deux chiffrés $\mathsf{ct}_0 := (\mathbf{c_{00}}, \mathbf{c_{01}}) \in R_q^2$ et $\mathsf{ct}_1 := (\mathbf{c_{10}}, \mathbf{c_{11}}) \in R_q^2$ en un chiffré $\widetilde{\mathsf{ct}}_\times := (\mathbf{c_0}, \mathbf{c_1}, \mathbf{c_2}) \in R_q^3$, avec $\mathbf{c_0} = \lfloor p/q(\mathbf{c_{00}} \cdot \mathbf{c_{10}}) \rceil$, $\mathbf{c_1} = \lfloor p/q(\mathbf{c_{00}} \cdot \mathbf{c_{11}} + \mathbf{c_{01}} \cdot \mathbf{c_{10}}) \rceil$) et $\mathbf{c_2} = \lfloor p/q(\mathbf{c_{01}} \cdot \mathbf{c_{11}}) \rceil$.

- $\mathsf{FV.Relin}(\mathsf{rk}, \widetilde{\mathsf{ct}}_\times)$: Relinéarise un chiffré $\widetilde{\mathsf{ct}}_\times := (\mathbf{c_0}, \mathbf{c_1}, \mathbf{c_2}) \in R_q^3$ en un chiffré $\mathsf{ct}_\times \in R_q^2$. Soient les éléments de la clé de relinéarisation $\mathsf{rk_i} = (\mathsf{rk_{i,0}}, \mathsf{rk_{i,1}}) \in R_q^2$, et $\mathbf{c_2}^{(i)}$ la décomposition de $\mathbf{c_2}$ en chiffre de base ω, sortie

$$\mathsf{ct}_\times := \left(\mathbf{c_0} + \sum_{i=0}^{\ell} \mathsf{rk}_{i,1} \cdot \mathbf{c_2}^{(i)}, \mathbf{c_1} + \sum_{i=1}^{\ell} \mathsf{rk_{i,1}} \cdot \mathbf{c_2}^{(i)} \right)$$

- $\mathsf{FV.Déchiffrement}(\mathsf{sk}, \mathsf{ct})$: Déchiffre un chiffré $\mathsf{ct} := (\mathbf{c_0}, \mathbf{c_1}) \in R_q^2$ en un clair $\boldsymbol{\mu} := [\lfloor p/q(\mathbf{c_0} + \mathbf{c_1} \cdot \mathbf{s}) \rceil]_p$

2.2 Paramétrage des librairies et organisation des tests

Dans l'objectif d'étudier le comportement et les performances des librairies SEAL, HElib, et FV-NFLlib, certains paramétrages et modifications sont nécessaires. C'est notamment le cas pour HElib, qu'il a fallu modifier afin que la librairie soit capable de traiter des opérations avec des modules en multi-précision. Cette version de la librairie a été appelée HElib-MP [35]. En ce qui concerne la librairie SEAL, nous avons utilisé deux versions selon l'expérience effectuée : la version 2.1 pour les expériences qui nécessitent

un module dans le domaine des clairs de taille 60 bits ou moins et la version 2.3 pour les expériences avec un module plus grand. La librairie FV-NFLlib a, quant à elle, été utilisée telle quelle.

Notons qu'il est envisageable de réaliser des opérations en multi-précision dans le domaine des clairs avec les librairies de base en utilisant plusieurs instances des schémas de chiffrement et en travaillant avec des modules premiers, puis en utilisant le théorème des restes chinois pour travailler sur un module de grande taille. Cependant, cette technique ne permet pas de travailler avec des modules arbitraires qui ne se factorisent pas en petits premiers, ce qui exclut les modules spécifiques tels que ceux utilisés avec ECDSA ou RSA.

HElib-MP est donc une extension de HElib pour la multi-précision qui se base également sur la librairie NTL pour implémenter le schéma BGV. Originellement, HElib permet de traiter des modules dans l'espace des clairs de la forme p^r, avec p et r des entiers en simple précision. Cette représentation ne permet pas de travailler avec un module choisi. HElib-MP modifie donc la génération des clés, le chiffrement, le déchiffrement, l'addition et la multiplication afin de permettre de travailler avec un module dans l'espace des clairs de taille arbitraire. Il a fallu adapter les paramètres en conséquence, notamment ceux relatifs au changement de module.

Nous avons analysé les performances des librairies et comparé l'impact des stratégies utilisées dans chaque librairie pour limiter la progression du bruit et les changements de représentations (tels que l'entrée et la sortie de la représentation en Double-CRT).

Les tests ont été réalisés pour deux versions de SEAL : v2.1 [36] et v2.3 [33].

SEAL v2.1 permet nativement de travailler avec des modules de taille arbitraire, mais la librairie n'est pas optimisée pour un choix de module de taille supérieure à 64 bits. En effet, la librairie considère un chiffré comme sa représentation en grands entiers plutôt qu'utiliser une transformation CRT ou Double-CRT, cf. 1.2.5. Cela permet de garantir une grande liberté et simplicité dans le choix des paramètres mais la multiplication polynomiale sur $\mathbb{Z}[x]$ est moins performante qu'en utilisant la représentation CRT, particulièrement lorsque les paramètres grandissent.

C'est pourquoi, pour de grands modules nous avons préféré SEAL v2.3 qui, quant à elle, utilise la représentation CRT en plus de la représentation dite *full-RNS* (Residue Number scheme) de FV proposée dans [37]. Il nous a semblé intéressant d'inclure les deux versions dans l'étude afin d'illustrer l'impact de la représentation et de mesurer l'impact des travaux réalisés dans [37].

Tous les tests ont été réalisés sur un Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz utilisant un seul coeur. Les paramètres ont été sélectionnés en utilisant le script *sagemath*

de Albrecht-Player-Scott [38] pour assurer au minimum 128 bits de sécurité (qui est la sécurité par défaut de SEAL v2.3). Le script retourne la sécurité des attaques les mieux étudiées contre la cryptographie basée sur LWE, et nous faisons l'hypothèse que le résultat est le même pour RLWE. En utilisant les heuristiques classiques, doubler la sécurité multiplierait les coûts par un facteur d'environ deux, ce qui permet d'étendre nos conclusions à des choix de sécurité plus importants.

Les tests des librairies ont été réalisés pour des clairs de taille 1 bit, 64 bits, 256 bits et 2048 bits. Le cas 1 bit se comporte de manière très différente des autres cas. Il est intéressant de l'aborder dans l'étude car c'est un choix largement répandu dans les applications et il permet également de comprendre les changements nécessaires pour passer à un grand module.

Pour chaque taille de module dans le domaine des clairs p, les tests consistent à incrémenter la profondeur multiplicative jusqu'à ce qu'une expérience dure plus de douze heures. Lors d'une expérience, un chiffré est mis au carré jusqu'à atteindre la profondeur multiplicative donnée. Nous nous intéressons au temps d'exécution (moyenné sur trois séries) divisé par la profondeur, ce qui donne le temps moyen d'une multiplication. Concernant le choix des paramètres, le module des chiffrés q est choisi assez grand pour réaliser les opérations tout en conservant le bruit suffisamment petit pour pouvoir déchiffrer, puis le degré n est choisi comme une puissance de 2 afin d'assurer au moins 128 bits de sécurité. Nous vérifions à la fin de l'expérience que le déchiffrement s'est correctement effectué.

La factorisation du polynôme cyclotomique $\Phi_m(x) = \prod_{j=0}^{\ell-1} F_j(x) \pmod p$ dépend du choix de m et p. Si le choix de p est contraint par une application donnée alors la capacité de *batching* (cf. 1.2.6) sera différente selon le choix de m et dépendra de la façon dont se factorise $\Phi_m(x)$ modulo p. Puisque le *batching* dépend de la valeur exacte de p et non seulement de sa taille, nous ne l'avons pas pris en compte. Le temps est donc le coût moyen pour une multiplication pour un clair qui permet le *batching*. Le fait que HElib permette une meilleure capacité de *batching* grâce à un choix plus libre de m pourra également être pris en considération.

2.3 Résultats

Dans cette section, nous présentons les résultats des expérimentations décrites dans la section précédente.

2.3.1 Taille des chiffrés

Ici, nous analysons l'évolution de la taille des chiffrés ($\log q$) en fonction de la profondeur multiplicative, et ceci dans deux cas : lorsque $\log p = 1$ et $\log p = 64$.

La figure 2.1 (haut) montre l'évolution de la taille des chiffrés pour $\log p = 1$. Lorsque la profondeur multiplicative augmente, toutes les librairies convergent vers une taille de chiffré similaire. Lors de la phase de relinéarisation, HElib-MP inclut des premiers spéciaux ce qui fait croître le chiffré d'un facteur multiplicatif quasi-constant.

FV-NFLlib a, asymptotiquement, des chiffrés plus grands d'un facteur multiplicatif que les deux autres librairies. Cette différence est due au choix des distributions du bruit, plus grandes dans FV-NFLlib que dans les autres librairies.

La figure 2.1 (bas) montre l'évolution de la taille des chiffrés lorsque $\log p = 64$. L'écart entre SEAL et FV-NFLlib, qui apparaît lorsque $\log p = 1$, est également visible. Lorsque la profondeur multiplicative augmente, un facteur multiplicatif important (environ deux) apparaît en défaveur de HElib-MP. Ce facteur est d'environ trois lors de la phase de relinéarisation. Cet écart est similaire lorsque $\log p = 256$ et $\log p = 2048$. Les courbes semblent montrer que le bruit généré par une multiplication augmente de $2 \log p$ ($3 \log p$ avec les premiers spéciaux) alors que pour SEAL et FV-NFLlib, il augmente de $\log p$.

2.3.2 Coût calculatoire

Nous nous intéressons ici au coût calculatoire d'une multiplication homomorphe dans le cadre des expérimentations décrites précédemment.

La figure 2.2 représente le coût d'une multiplication pour le cas $\log p = 1$, pour les 4 librairies, en fonction de la profondeur multiplicative désirée. On constate que les coûts pour SEAL v2.1 sont largement plus importants que pour les autres librairies, ce qui montre que l'utilisation de la représentation CRT est essentielle avec un niveau multiplicatif élevé. Les coûts pour SEAL v2.3 et FV-NFLlib s'entrecroisent régulièrement. Même si SEAL est régulièrement plus performante que FV-NFLlib, l'écart de performance entre les deux librairies est peu significatif.

Les coûts pour HElib, par contre, semblent croître de façon plus lente que pour les autres librairies et on remarque un écart qui se creuse à partir d'une profondeur de 25. Cet écart augmente jusqu'à atteindre un facteur 4 pour la profondeur 60. Asymptotiquement, HElib semble donc plus performante que les autres librairies.

La figure 2.3 représente le coût d'une multiplication pour $\log p$ valant 64, 256, et 2048, sauf pour SEAL où $\log p = 64$ est remplacé $\log p = 60$ dans le but d'utiliser SEAL v2.3. Les performances de SEAL v2.1, comme constaté pour $\log p = 1$, montrent un coût de

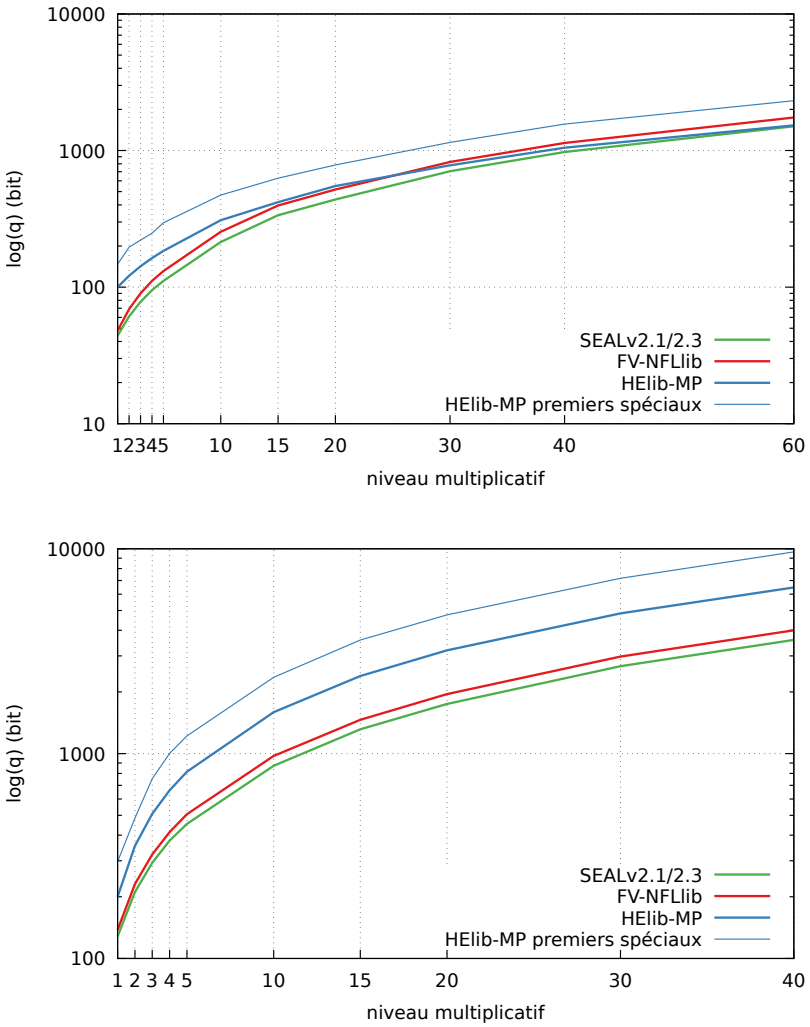

FIGURE 2.1 Évolution de n et $\log q$ qui permettent de déchiffrer en fonction de la profondeur multiplicative pour $\log p = 1$ (haut) et $\log p = 64$ (bas). Pour $\log p = 256$ et $\log p = 2048$, les résultats sont similaires au cas $\log p = 64$. Deux résultats sont présentés pour HElib-MP, l'un avec les premiers spéciaux utilisés lors de la relinéarisation et l'autre sans.

calcul qui augmente de manière significative comparé aux autres librairies. La courbe correspondant à SEALv2.1 s'arrête plus tôt que pour les autres librairies pour $\log p$ égal à 64 ou 256, au moment où les tests durent plus de 12 heures. Pour $\log p = 2048$ SEALv2.1 est incapable de faire une seule multiplication en moins de 12 heures.

SEALv2.3 et FV-NFLlib ont, encore ici, un comportement similaire et montrent des

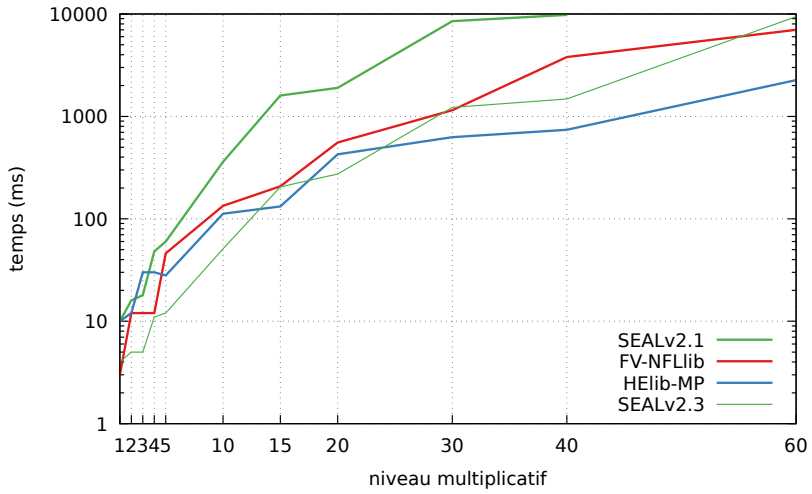

FIGURE 2.2 Temps moyen d'une multiplication en fonction du niveau multiplicatif avec $\log p = 1$.

performances proches pour un module de 64 bits (60 pour SEAL). C'est un résultat inattendu puisque SEALv2.3 implémente la version *full-RNS* de FV contrairement à FV-NFLlib. On aurait donc pu s'attendre à de meilleures performances de la part de SEAL.

Le coût d'une multiplication pour HElib est plus important (jusqu'à un facteur 2 au départ) jusqu'à une profondeur d'environ 40 pour $\log p = 64$ et pour une profondeur 7 pour $\log p = 256$. Au dessus de ces profondeurs multiplicatives, HElib est plus performante que les autres librairies (même si les tests pour $\log p = 64$ et une profondeur au dessus de 40 durent plus de 12 heures, il est cohérent de penser que les librairies garderont le même comportement). Le point de croisement semble intervenir lorsque $\log q$ vaut environ 2500 bits pour FV-NFLlib et SEAL, après ce point, les performances de HElib creusent l'écart avec les performances des autres librairies. Pour l'utilisation d'un module de 2048 bits, HElib est la plus performante (d'un facteur atteignant 2.5).

Soulignons que les résultats présentés ici concernent uniquement la multiplication et non l'addition ou l'absorption. Ce choix se justifie par le fait que la multiplication est l'opération limitante car la plus coûteuse mais également car il y a peu de choix dans l'implémentation d'une addition ou une absorption en homomorphe. Une série de tests a été réalisée et les coûts des additions et des absorptions sont semblables pour SEAL v2.3, FV-NFLlib et HElib-MP pour des paramètres égaux.

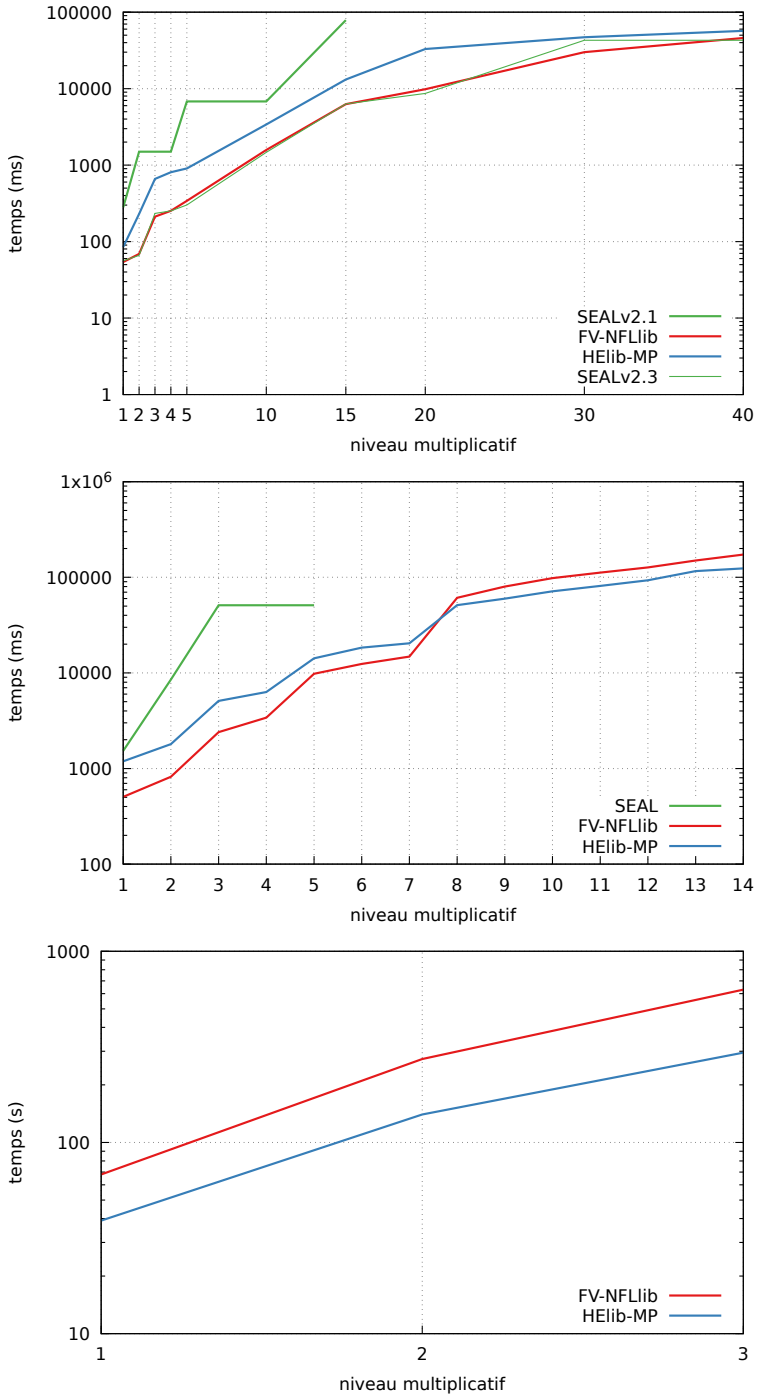

FIGURE 2.3 Temps moyen pour une multiplication en fonction du niveau multiplicatif lorsque $\log p = 64$ ($\log p = 60$ pour SEAL), $\log p = 256$ et $\log p = 2048$.

2.4 Evolution du bruit

Notre objectif dans cette section est d'étudier l'évolution du bruit dans les différentes expériences conduites, afin de mieux comprendre l'évolution de $\log q$. Le but de cette section n'est pas de fournir une stratégie pour sélectionner les paramètres optimaux mais de justifier les observations des sections précédentes. Les analyses suivantes proposent donc une majoration du bruit. Il est intéressant de souligner que l'analyse réalisée dans le document présentant HElib [31] propose une estimation de la variance du bruit, alors que les travaux présentant FV [39] fournissent une majoration du bruit, en supposant un seuil sur les gaussiennes utilisées.

Nous utiliserons les notations suivantes, $\delta = \mathsf{sup}(\|\mathbf{a} \cdot \mathbf{b}\|_\infty / \|\mathbf{a}\|_\infty \|\mathbf{b}\|_\infty : a, b \in R)$, δ étant le facteur d'expansion de l'anneau polynomial. Notons B_{key} et B_{err} les bornes supérieures associées à χ_{key} et χ_{err}.

2.4.1 Evolution du bruit dans FV-NFLlib et SEAL

L'analyse faite dans [40] montre qu'un chiffré sortant de la fonction FV.Encrypt a un bruit initial qui peut être majoré par $V = B_{\mathsf{err}}(1 + 2\delta B_{\mathsf{key}})$.

Après L niveaux de multiplications et relinéarisations, le bruit résultant est majoré par

$$U_L = C_1^L V + L C_1^{L-1} C_2 = L C_1^L (C_2/C_1 + V/L),$$

où

$$C_1 = 2\delta^2 p B_{\mathsf{key}}, \quad C_2 = \delta^2 B_{\mathsf{key}}(B_{\mathsf{key}} + p^2) + \delta\omega \log_\omega(q) B_{\mathsf{err}}.$$

Dans SEAL, $B_{\mathsf{key}} = 1$. En utilisant cette substitution et en remplaçant les constantes dans l'expression on obtient

$$U_L = L\left(2\delta^2 p\right)^L \left(\frac{1+p^2}{2p} + \frac{\omega \log_\omega(q) B_{\mathsf{err}}}{2\delta p} + o_L(1)\right),$$

où $o_L(1)$ est un terme asymptotiquement petit en L. La première fraction est plus petite (bien que proche) de p et la seconde plus petite que 1 pour toutes les valeurs considérées de ω (le nombre de parties utilisées lors de la relinéarisation). Cette borne supérieure peut être arrondie à l'expression $U_L \simeq Lp(2\delta^2 p)^L$. On peut donc s'attendre, pour SEAL, à ce que $\log q$ soit proche de $L \log(2\delta^2 p)$ lorsque L grandit.

Pour FV-NFLlib, on considère que $B_{\mathsf{key}} = B_{\mathsf{err}}$, ce qui est le cas dans tous les exemples utilisés pour présenter la librairie. En utilisant la même approche que pour SEAL on obtient $U_L \simeq LpB_{\mathsf{err}}(2\delta^2 p B_{\mathsf{err}})$. On peut donc s'attendre à ce que $\log q$ soit proche de

$L \log(2\delta^2 p B_{\mathsf{err}})$ lorsque L grandit.

Cela explique la légère différence multiplicative entre FV-NFLlib et SEAL lorsque $\log p = 1$, qui disparaît pour des valeurs plus importantes de $\log p$ puisque $\log(2\delta^2 p)$ et $\log(2\delta^2 p B_{\mathsf{err}})$ sont similaires pour $\log p \in \{64, 256, 2048\}$.

Notons qu'il est envisageable de modifier les paramètres de sécurité de FV-NFLlib afin d'utiliser un plus petit bruit ($B_{\mathsf{key}} = 1$) et de faire disparaître la différence dans l'évolution du bruit entre FV-NFLlib et SEAL pour $\log p = 1$.

2.4.2 Evolution du bruit dans HElib

L'analyse de l'évolution du bruit dans HELib est plus complexe. En effet, avec cette librairie, un chiffré ct possède un bruit $\mathbf{v} \in R$ de norme minimale $\|\mathbf{v}\|_\infty$, tel que $\mathbf{c_0}+\mathbf{c_1}\cdot\mathbf{s} = p\mathbf{v}+[q_i\boldsymbol{\mu}]_p$. Si la norme est sous un seuil $\|\mathbf{v}\|_\infty < q_i/p$, alors le déchiffrement est correct, et le clair retourné est $\boldsymbol{\mu}$. Un chiffré généré par la fonction $\mathsf{HElib.Encrypt}$ possède un bruit $\mathbf{v}_{\mathsf{enc}} = \mathbf{u}\cdot\mathbf{e} + \mathbf{e_0} + \mathbf{s}\cdot\mathbf{e_1}$ qui peut être majoré par $\|\mathbf{v}_{\mathsf{enc}}\|_\infty < V = (1+2\delta)B_{\mathsf{err}}$.

Une étude détaillée sur l'évolution du bruit lors des opérations est disponible dans [31], qui analyse la variance du bruit, probablement la meilleure approche pour avoir une borne précise. Cependant les résultats issus de cette étude sont trop précis et particulièrement complexes à utiliser pour obtenir une borne supérieure du bruit.

Nous donnons ici une analyse simple estimant une borne supérieure du bruit après une opération composée de : une multiplication, un changement de modules et une relinéarisation.

Multiplication. Soit $\mathsf{ct} := (\mathbf{c_0}, \mathbf{c_1})$ que l'on multiplie par lui même (la mise au carré permet de réduire le nombre de variables) avec

$$\mathbf{c_0} := \mathbf{u}\cdot\mathbf{b} + p\mathbf{e_0} + [q_i\boldsymbol{\mu}]_p \text{ et } \mathbf{c_1} := \mathbf{u}\cdot\mathbf{a} + p\mathbf{e_1}.$$

La première partie résultante de l'opération est $[q_i^{-1}]_p \mathbf{c_0}\cdot\mathbf{c_0}$. C'est la partie du chiffré qui contient l'information sur le clair $\boldsymbol{\mu}$. Afin de conserver l'invariance sur le clair et avoir un déchiffrement correct, il est nécessaire de calculer $\left[q_i^{-1}\right]_p \mathbf{c_0}\cdot\mathbf{c_0}$, et non $\mathbf{c_0}\cdot\mathbf{c_0}$, multiplié (*modp*) par le module dans l'espace des chiffrés. Le terme résultant de $\mathbf{c_0}\cdot\mathbf{c_0}$ est donc $[q_i^2\boldsymbol{\mu}^2]_p$ et non $[q_i\boldsymbol{\mu}^2]_p$.

Lorsque l'on multiplie le terme de bruit par lui même on obtient $p^2\mathbf{e_0}\cdot\mathbf{e_0}$. Puisqu'il est également nécessaire de multiplier tous les termes par $[q_i^{-1}]_p$, qui est de la même taille que p, nous pouvons majorer ce terme par p^3 fois le bruit mis au carré. Si l'on note U le bruit précédent on obtient une borne qui est $\delta p^3 U^2$ après multiplication.

Changement de module Lors du changement de module, l'opération qui permet de

passer d'un module q_i à un module q où $q < q_i$, le bruit résultant diminue d'un facteur inverse $\Delta = q_i/q$, puis grossit d'un terme additif dû à l'erreur d'arrondi. Le bruit résultant est donc majoré par $\|\mathbf{v}_{\mathsf{mod}}\|_\infty < \|\mathbf{v}\|_\infty/\Delta + B_{\mathsf{err}}$, où B_{err} vaut au moins p puisqu'il est nécessaire d'avoir un élément divisible par p lors du changement de module. Cela implique que le changement de module ne peut pas réduire le bruit en dessous de p.

L niveaux de multiplications. Analysons le bruit résultant après L multiplications, en ignorant pour le moment la relinéarisation. Un chiffré sortant de la fonction de chiffrement a un bruit majoré par $V \sim 2\delta B_{\mathsf{err}}$. Après une multiplication, la majoration du bruit est $4p^3\delta^2 B_{\mathsf{err}}^2$. Lorsque p est grand, pour éviter un accroissement géométrique du bruit, le facteur Δ utilisé lors du changement de module devra être au moins $\Delta \sim p^2$. Ainsi pour permettre L multiplications, $\log q$ grossira en $2\log p$ ce qui justifie les observations qui montrent que l'évolution de $\log q$ est deux fois plus rapide pour HElib-MP comparée à FV-NFLlib et SEAL.

Relinéarisation. Si l'on considère une multiplication suivie par une opération de relinéarisation, quand $\|\mathbf{v_0}\|_\infty\|, \|\mathbf{v_1}\|_\infty\| < V$, le bruit après relinéarisation est majoré par $\|\mathbf{v}_{\mathsf{mul}}\|_\infty < U = tV^2 + \delta B_{\mathsf{err}} \sum_{i=1}^{\ell} B_i/2q'$. Puisque, dans HElib, la taille de B_i est fixée à $1/3\log q_i$, l'analyse effectuée précédemment reste valable tant que $\log q' \sim 1/3\log q_i$. Cela valide le raisonnement simplifié précédent et la taille observée pour HElib-MP avec les premiers spéciaux.

2.5 CRT

Comme nous l'avons vu dans les sections précédentes, notamment lorsque nous avons comparé les performances de SEAL v2.1 à celles des autres librairies, les choix de représentation des grands nombres ont une importance capitale en termes de coûts calculatoires, mais également en termes de taille maximale des paramètres. Dans cette section, nous nous penchons particulièrement sur les représentations utilisées pour les clairs et les chiffrés, i.e. les polynômes de R_p et R_q, les coûts associés aux changements de représentation et à la compatibilité des opérations liées à ces changements.

2.5.1 Coût du changement de représentation

Passer d'une représentation Double-CRT (un polynôme sous forme de valeurs en CRT) à une représentation CRT (un polynôme sous forme de coefficients en CRT), ou vice-versa, est peu coûteux. Il suffit de réaliser une transformation NTT, ou NTT inverse respectivement, dont le coût est $O(n \log n \log q)$.

Sortir de la représentation CRT, ou projeter des coefficients dans une représentation CRT, est plus couteux. En effet, le coût pour chaque coefficient est en $O(n \log^2 q)$ et $\log q$ peut être conséquent lorsque l'on travaille avec un grand module dans l'espace des clairs.

En règle générale, pour garder une sécurité constante dans la cryptographie basée sur les réseaux, n doit grandir linéairement en fonction de $\log q$ et donc le ratio $\frac{\log q}{\log n}$, qui est le ratio du coût de la transformation CRT sur celui de la transformation NTT, grandit en $O(\frac{\log q}{\log \log q})$.

Etant donné ces couts, la solution à privilégier est donc de rester en Double-CRT ou en CRT afin d'éviter les transformations CRT tout en se permettant des transformations NTT et NTT inverses. Examinons maintenant quelles opérations sont possibles tout en restant en Double-CRT, ou en CRT, et lesquelles ne le sont pas.

2.5.2 Opérations compatibles avec la représentation CRT

Les opérations suivantes peuvent être réalisées directement en Double-CRT :

— L'addition polynomiale
DoubleCRT($\mathbf{a} + \mathbf{b}$) = DoubleCRT($\mathbf{a}$)+ DoubleCRT($\mathbf{b}$)

— La multiplication polynomiale
DoubleCRT($\mathbf{a} \cdot \mathbf{b}$) = DoubleCRT($\mathbf{a}$)$\cdot$ DoubleCRT($\mathbf{b}$)

— La multiplication par un scalaire
DoubleCRT($p \cdot \mathbf{a}$) = $p$$\cdot$DoubleCRT($\mathbf{a}$)

— L'échantillonnage uniforme
DoubleCRT($\mathbf{a}$) pour $\mathbf{a} \xleftarrow{U} R_q \sim \mathbf{a} \xleftarrow{U} R_q$

Le coût de ces opérations dans R_q est en $O(n \log q)$.

Si le polynôme est en représentation CRT, les coûts asymptotiques sont équivalents sauf pour la multiplication polynomiale qui, elle, est en $O(n \log n \log q)$ puisqu'il est nécessaire de réaliser une transformation NTT, la multiplication en Double-CRT puis la NTT inverse.

Les opérations présentes dans HElib.ChangementModule sont la soustraction multiple d'une constante et la vérification de divisibilité par un entier pour un polynome sous forme de coefficients. Ces opérations sont moins coûteuses que le $O(n \log^2 q)$ de la transformation CRT. Pour soustraire la constante, il est suffisant de la projeter en représentation CRT, une seule fois pour tout le polynôme, pour un coût en $O(\log^2 q)$ puis de réaliser le soustraction en forme CRT le nombre nécessaire de fois. Pour vérifier la divisibilité par p, il suffit d'étendre la représentation CRT des coefficients à $q \times p$ et vérifier si la coordonnée associé à p est zéro. Cette extension a un coût en $O(n \log q \log p)$, qui ne grandit pas de manière quadratique en $\log q$.

Finalement, si un entier d_1 en représentation CRT est divisible par une constante d_2, et que d_2 est un facteur de la représentation CRT de d_1, alors il est possible de calculter d_1/d_2 sans quitter la représentation CRT. Il suffit en effet, de supprimer la coordonnée associée à d_2 et de multiplier les autres par l'inverse de d_2.

Les opérations décrites dans cette section suffisent à réaliser l'addition, la multiplication, le changement de module et la relinéarisation dans HElib. Il est nécessaire de projeter les coefficients dans la représentation CRT ou d'en sortir uniquement lors de la génération de clé, le chiffrement et le déchiffrement dans HElib.

2.5.3 Opérations incompatibles avec la représentation CRT

La plupart des opérations incompatibles avec la représentation CRT, par exemple l'échantillonnage gaussien, sont uniquement nécessaires lors de la génération de clés et du chiffrement et du déchiffrement.

Il y a cependant une opération clé dans FV qui, lors de la multiplication de chiffré, est incompatible avec la représentation CRT. En effet, il est nécessaire pour réaliser une multiplication de calculer $[\lfloor p(\mathbf{a} \cdot \mathbf{b})/q \rceil]_q$ avec $\mathbf{a}$, $\mathbf{b}$ des éléments de R_q.

Le principal problème est le suivant : la multiplication $p(\mathbf{a} \cdot \mathbf{b})$ doit être réalisée sur $\mathbb{Z}$ et non modulo (q). Il est donc nécessaire, lors de cette opération, de sortir de la représentation CRT. De plus, la division par q et l'arrondi doivent être calculés sur les coefficients, mais il est impensable de réaliser la multiplication $\mathbf{a} \cdot \mathbf{b}$ dans une représentation par coefficient car cela impliquerait un coût quadratique en n. Ces contraintes mènent à l'algorithme 5.

SEAL v2.1 emploie une approche triviale de multiplication rapide pour multiplier les polynômes dans une représentation par coefficients qui ne sont pas en représentation CRT. Une telle approche a un coût en $K_c n^{1+c} \log^2 q$ avec K_c une constante qui n'est pas négligeable lorsque c est inférieur à 0.58 pour la multiplication de Karatsuba.

Dans FV-NFLlib, l'idée est d'étendre la base CRT à un module $q' > q^2 n$, puis réaliser la multiplication modulo (q'). Puisque ce module est plus petit que la norme infinie du polynôme résultat, il n'y a pas de réduction $mod q'$ et donc le résultat est le même que si l'on travaille sur $\mathbb{Z}$. Avec cette stratégie, il est nécessaire de réaliser plusieurs projections dans la base CRT et d'en sortir avec, à chaque fois, un coût en $n \log^2 q$. En prenant $\log q = O(L \log p)$ on obtient une complexité de $O(nL^2 \log^2 p)$ pour FV-NFLlib et $O(nL \log^2 p)$ pour HElib ce qui explique les différences observées asymptotiquement.

SEAL v2.3 implémente la version *full-RNS* de FV proposée par Bajard et al. [37]. Cet article propose de nombreuses pistes de réduction des coûts en effectuant un maximum d'opérations en double-CRT. L'idée principale est donc de rester en représentation CRT,

Algorithme 5 : Multiplication dans FV-NFLlib

L'approche naïve
Input : **a**, **b** des éléments de $R_q = R/qR$ with $R = \mathbb{Z}[X]/\Phi_{2n}(X)$, q et $p \in \mathbb{Z}$ et n une puissance de 2
Output : $[\lfloor t(\mathbf{a} \cdot \mathbf{b})/q \rceil]_q$ sont effectués sur les coefficients
begin
  $\mathbf{a_1} = CRT_q^{-1} \circ NTT_{R_q}^{-1}(\mathbf{a})$, $\mathbf{b_1} = CRT_q^{-1} \circ NTT_{R_q}^{-1}(\mathbf{b})$;
  $\mathbf{c} = FastMul(\mathbf{a_1}, \mathbf{b_1}))$;
  $\mathbf{c_1} = [\lfloor p \cdot \mathbf{c}/q \rceil]_q$;
  $\mathbf{res} = NTT_{Rq} \circ CRT_q(\mathbf{c_1})$;
  return res;

Algorithme amélioré
Input : **a**, **b** éléments de $R_q = R/qR$ avec $R = \mathbb{Z}[X]/\Phi_{2n}(X)$, q et $t \in \mathbb{Z}$, $q' > q^2 n$ avec n une puissance de 2
Output : $[\lfloor t(\mathbf{a} \cdot \mathbf{b})/q \rceil]_q$ où $\mathbf{a} \cdot \mathbf{b}$ s'effectue sur les entiers et la division et l'arrondi sur les coefficients
begin
  $\mathbf{a_1} = CRT_q^{-1} \circ NTT_{R_q}^{-1}(\mathbf{a})$, $\mathbf{b_1} = CRT_q^{-1} \circ NTT_{R_q}^{-1}(\mathbf{b})$;
  $\mathbf{a_2} = NTT_{Rq'} \circ CRT_{q'}(\mathbf{a_1})$, $\mathbf{b_2} = NTT_{Rq'} \circ CRT_{q'}(\mathbf{b_1})$;
  $\mathbf{c} = \mathbf{a_2} \cdot \mathbf{b_2}$;
  $\mathbf{c_1} = CRT_{q'}^{-1} \circ NTT_{R_{q'}}^{-1}(\mathbf{c})$;
  $\mathbf{c_2} = [\lfloor p \cdot \mathbf{c_1}/q \rceil]_q$;
  $\mathbf{res} = NTT_{Rq} \circ CRT_q(\mathbf{c_3})$;
  return res;

lors de la multiplication et du déchiffrement, en utilisant une base CRT étendue de manière efficace. Cependant, ils n'ont pas réussi à se passer du $O(n \log^2 q)$ de l'algorithme de multiplication. Cela semble être la raison pour laquelle SEAL se comporte de la même façon que FV-NFLlib asymptotiquement et est en dessous de HElib-MP.

2.6 Conclusion

Dans ce chapitre, nous avons comparé le comportement et les performances de différentes librairies de chiffrement homomorphe en présence de domaines de messages clairs de différentes tailles. Les librairies étudiées, FV-NFLlib, HElib et SEAL dans deux versions différentes, proposent des implémentations et des optimisations spécifiques qui ont un impact direct sur les performances de celles-ci, et que nous avons mis en lumière. Ici, nous essayons de tracer des tendances générales et de donner des pistes d'améliorations potentielles.

2.6.1 Implémentations possibles

Une première conclusion évidente, issue de l'analyse que nous venons de présenter ici, est que de se passer de la représentation CRT pour les entiers, même en utilisant un algorithme de multiplication avancé, tel que celui de Karatsuba, ne semble pas être une bonne option pour une implémentation efficace de schémas de chiffrement homomorphe.

Deuxièmement, on peut dire que BGV semble être supérieur à FV en présence de très grands modules, même en considérant la version *FullRNS*. Il serait intéressant de se pencher sur les performances d'une librairie implémentant BGV, plus simple et mieux optimisée, par exemple basée sur NFLlib. Cette librairie pourrait rivaliser avec une librairie basée sur FV pour un module plus petit.

L'utilisation de NFLlib ainsi que l'approche *FullRNS* semblent apporter un gain non négligeable aux performances de FV. Il serait intéressant de réaliser une implémentation qui utilise les deux approches pour exploiter au mieux le schéma.

2.6.2 Recommandation de librairies

Si on se base sur le critère du temps nécessaire pour réaliser une multiplication, il est facile, après cette étude, de choisir la librairie optimale pour une application donnée, à taille des domaines des clairs et nombre de multiplications fixés.

Pour $\log p = 1$, SEAL v2.3 est le meilleur choix jusqu'à une profondeur multiplicative d'environ 12 puis SEAL v2.3 et HElib ont des performances similaires jusqu'à une

profondeur d'environ 25. Au delà, HElib est le choix le plus optimal. Voir Figure 2.2

Pour $\log p = 60$, FV-NFLlib et SEAL v2.3 sont les plus performantes jusqu'à une profondeur multiplicative d'environ 40. Cependant, SEAL est plus facile d'accès et la librairie continue d'être activement améliorée. Il est donc conseillé de se tourner vers SEAL v2.3. Voir Figure 2.3

Pour $\log p$ supérieur à 60, si le module de l'espace des clairs peut être factorisé comme plusieurs modules de 60 bits, alors la conclusion précédente s'applique en utilisant le théorème des restes chinois dans l'espace des clairs. Sinon, pour un module RSA par exemple, FV-NFLlib est la meilleure approche (puisque SEALv2.3 n'est pas utilisable) pour $\log q < 2500$. Au dessus de ces paramètres HElib-MP est le meilleur choix.

Chapitre 3

Externalisation sécurisée de données génomiques

Les progrès de la génomique et du séquençage ADN sont rapides. La baisse du coût de séquençage entraîne la production d'une grande quantité de données. Ces données étant de nature très sensible vis à vis de la vie privée (il est possible avec ces données d'identifier une personne précisément et d'apprendre des informations cruciales sur elle notamment d'un point de vue médical), il convient de les stocker de façon sécuritaire en garantissant leur confidentialité et le contrôle de leur accès. Dans ce chapitre, nous nous intéressons au stockage sécurisé. Nous proposons une solution permettant de tester si une donnée est présente ou non dans une base de données chiffrée tout en occultant la position où se trouverait la donnée cherchée. Notre solution permet de réaliser ceci tout en réduisant fortement les coûts en communication par rapport à la solution triviale qui consiste à télécharger toute la base de données pour effectuer la recherche localement.

3.1 Contexte

3.1.1 Les bases des données génomiques

L'analyse ADN d'un individu est une pratique de plus en plus courante, la principale raison étant la décroissance exponentielle du prix. Un séquençage d'un génome coûtait environ 100 000 000$ au début des années 2000 et ne coûte plus que quelques centaines de dollars de nos jours [41]. Une autre raison à cette démocratisation est le temps que prend un séquençage : il faut maintenant moins d'un jour pour réaliser cette opération [42]. Cet accès aux données génomiques a créé une nouvelle approche médicale appelée *data-driven*

medicine. L'effondrement du coût du séquençage ADN ouvre la porte à une médecine personnalisée où chaque patient serait traité selon son profil génétique.

En effet, en fonction des gènes du patient, un traitement sera à privilégier par rapport à un autre. L'accès aux gènes permettant de mieux estimer la probabilité de déclenchement de certaines maladies, les politiques de préventions pourraient être améliorées, voire ciblées. Avant d'être utilisé comme outil de traitement, le séquençage ADN intervient d'abord lors du dépistage. On a pu le découvrir en 2013, lorsque l'actrice Angelina Jolie a révélé avoir subi une ablation des seins pour prévenir un risque élevé de cancer mis en évidence par un dépistage génétique [43]. Il est concevable que ce soit les patients eux-mêmes qui demandent l'utilisation de leur ADN pour profiter de meilleurs dépistages et d'une médecine personnalisée.

Enfin, l'avancement ces dernières années des performances en terme de débit des réseaux et de capacité de stockage a entraîné une rapide avancée dans la digitalisation des données génomiques. Cependant, il est important de remarquer qu'une séquence ADN peut avoir une taille importante (de l'ordre de 100Go) [44]. Ainsi, il n'est pas évident que des entités telles qu'un hôpital ou un cabinet médical puissent avoir accès, en interne, aux capacités de stockage et de traitement nécessaires pour une base de données génomique. C'est pour cette raison que de nombreux centres de recherches médicaux se sont tournés vers le stockage et le traitement en ligne de ces données [45].

Des bases de données publiques regroupant les données génomiques ont émergé en particulier pour faciliter la recherche. GenBank est la base de données de l'institut américain NIH (*National Institute of Health*) [1]. Dans le cadre d'une collaboration internationale pour la mise en commun de données génomiques, GenBank échange ses données quotidiennement avec des bases équivalentes au Japon et l'Union Européenne. Leur taille est multipliée par environ 2 tous les 10 mois, ce qui met en relief à quel point le volume de ces données augmente.

3.1.2 Problématique de la vie privée en génomique

Déporter des données génomiques vers un serveur distant n'est pas une tâche facile. Les données génomiques sont bien évidemment liées aux individus et soulèvent des problèmes de vie privée qui sont difficiles à contourner. Voici énumérés quelques uns de ces problèmes.

— Les données génomiques sont des données qui durent dans le temps, et n'évoluent que marginalement tout au long de la vie d'un individu. Ceci implique qu'une

1. https ://www.ncbi.nlm.nih.gov/genbank/

fuite de données aura un impact sur la vie privée d'un individu tout au long de sa vie.

— La persistance des données génomiques va au delà de la vie d'un individu. Ces données ont une certaine persistance par transmission. Ceci implique que les données génomiques ne révèlent pas des informations uniquement sur un individu, mais aussi sur sa famille [46] [47]. Si des personnes acceptent de livrer leur données génomiques à des bases publiques, et qu'un rapprochement familial est fait entre un individu et ces personnes, il est possible d'obtenir des informations sur l'ADN de cet individu. Un exemple remarquable d'une telle situation est donné par le cas du *Golden State Killer*, un tueur en série ayant commis des crimes dans les années 1970 et 1980 a été arrêté en 2018 [48]. Les enquêteurs ont utilisé des échantillons d'ADN recueillis sur les lieux des crimes, puis effectué des recherches dans une base de données en ligne. Ils sont arrivés ainsi à retrouver des membres de la famille du tueur, des cousins éloignés, qui avaient transmis leur ADN sur Internet, puis à reformer un arbre généalogique permettant de monter au tueur.

— Les êtres humains ont une grande partie d'ADN commune, mais la différence qu'ils ont est unique sauf dans le cas de vrais jumeaux. Il est donc possible de lier les données génomiques à une unique personne. Cette propriété, liée à l'identification d'un unique individu grâce à son ADN, est par exemple utilisée lors de tests ADN lors d'enquêtes policières [49]. Ainsi, non seulement les données génomiques sont liées à vie aux individus et aux membres de leurs familles, mais il est potentiellement possible de lier ces données aux identités des individus. L'anonymisation consistant à séparer les données de l'identité des individus n'est donc pas une protection suffisante.

— Même en allant au delà de la suppression des identités associées aux données génomiques il n'est pas évident d'assurer que le lien avec un individu ne pourra pas être fait. Les données ADN donnent des informations sur les caractéristiques visibles (le phénotype) d'un individu [50] (sexe /la couleur des yeux / des cheveux / de la peau ...). Il existe également de nombreux marqueurs non visibles à l'oeil nu qui pourraient être utilisés (groupe sanguin, lien de parenté avec quelqu'un ayant eu une maladie rare, etc.). Par ailleurs, les techniques qui permettraient de faire un portrait robot d'une personne en fonction des ses données ADN n'existent pas encore mais le recherche progresse en ce sens [51] [52]. Enfin, il est possible d'utiliser des bases de données publiques pour retrouver des informations sur le possesseur des données. Par exemple le chromosome Y chez les hommes est transmis quasiment intégralement. Par voie de conséquence, il est possible

de retrouver le nom de famille d'une personne si les données génomiques de ses ancêtres sont disponibles, par exemple dans une base de données publique. Cette technique a déjà été utilisée aux Etats-Unis lors d'un don anonyme de spermatozoïdes. L'enfant issu de ce don a réussi à retrouver son père puisqu'il était présent dans une base de données génomiques et avait le même chromosome Y [53].

— Enfin, un problème majeur associé à une potentielle divulgation de données génomiques est le lien entre ces données et des problèmes de santé ou comportementaux. Des études ont montrés que les données ADN pouvaient indiquer la probabilité d'avoir Alzheimer [54] ou d'être un jour atteint de démence. Une compagnie d'assurance malveillante qui tomberait sur ces données pourrait refuser d'assurer une personne qui a un risque plus important d'avoir un jour une maladie grave.

Les menaces décrites ci-dessus ne se cristallisent à priori que s'il y a une fuite d'information (accidentelle ou suite à un vol de données), ou une divulgation faite par un individu volontairement. Les divulgations volontaires sont un problème réel et il est inquiétant d'observer la prolifération de sites (par exemple Open-SNP.org) spécialisés dans le partage des données génomiques, parfois même avec un identifiant tel que le nom ou prénom.

En contrepartie, il y a des lois qui émergent pour essayer de protéger ces données. En 2007 le gouvernement des États-Unis a adopté une loi *Genetic Information Nondiscrimination Act (GINA),*[2] qui interdit les discriminations liées aux données génomiques pour les assurances ou lors de la phase de recrutement pour un travail. En 2012, toujours aux États-Unis, est paru un rapport, *Presidential Commission for the Study of Bioethical Issues*[3], sur les données génomiques qui étudiait les options techniques et politiques pour protéger la vie privée dans le domaine de la génomique. En 2008, l'Union Européenne a adopté une convention *Additional Protocol to the Convention on HumanRights and Biomedicine, concerning Genetic Testing for Health Purposes*[4] sur l'utilisation des tests génétiques pour la santé. Il y a cependant un grand nombre de systèmes judiciaires différents dans le monde et tous n'ont pas la même approche concernant la vie privée. Enfin, même quand les lois existent, leur application n'est pas toujours évidente. Par exemple, il n'est pas simple de prouver quelle est l'origine d'un refus de recrutement ou d'assurance.

2. https ://www.genome.gov/Pages/PolicyEthics/GeneticDiscrimination/SAPonHR493.pdf
3. https ://bioethicsarchive.georgetown.edu/pcsbi/sites/default/files/PrivacyProgress508_1.pdf
4. https ://rm.coe.int/CoERMPublicCommonSearchServices/DisplayDCTMContent ?documentId=0900001680084824

La loi seule ne permettant pas de prévenir l'utilisation malveillante des informations d'ordre privé, il est donc nécessaire d'utiliser des solutions techniques pour garantir la vie privée des individus. Utiliser de telles techniques pourrait être vécu comme un frein au développement économique et à la recherche en génomique. Mais il est important de comprendre que pour développer la génomique il faut que les individus soient encouragés à partager leurs données, et pour les encourager il est nécessaire de garantir que ces données ne seront pas divulguées ni utilisées de façon malveillante. Le meilleur moyen de donner des garanties est en combinant législation et techniques de protection de vie privée.

Dans ce chapitre, nous proposons une solution technique permettant de stocker les données génomiques de manière confidentielle. Il est ensuite possible pour le client de vérifier la présence d'une donnée de manière privée. Notre approche utilise comme fondations un retrait d'informations privé (PIR) et du chiffrement homomorphe.

3.2 Aspects techniques et outils

3.2.1 Notations

Un chiffré d'un schéma fondé sur RLWE est un polynôme de degré n. Le module dans le domaine des clairs est p, celui dans le domaine des chiffrés q. Une base de données contient L entrées. Il y aura `tailleligne` éléments par entrée dans la base de données. Chacun de ces éléments aura une taille l. Un élément k chiffré sera noté $E(k)$ pour un chiffrement classique et $HE(k)$ pour un chiffrement homomorphe. La requête utilisée lors d'un protocole de type PIR sera notée Q et la réponse R. Nous aurons besoin d'une requête dite de soustraction notée S. La valeur d'un haché sera représentée par h, le k-ème bit du haché h sera noté h_k.

3.2.2 Randomisation d'un chiffré

Dans ce chapitre nous avons besoin de multiplier les coordonnées d'un chiffré par un masque aléatoire. Soit un système de chiffrement homomorphe utilisant une NTT pour faire du batching (c.f. Section 1.2.6). Considérons un chiffré $HE((k_0, \ldots, k_{n-1}))$ tel qu'un des k_i soit égal à 0. Nous souhaitons conserver la présence du 0 et randomiser les autres coordonnées. Le batching nous permet de faire ceci en multipliant par un polynôme aléatoire car celui-ci aura des valeurs $(v_0, \ldots, v_{n-1})$ aléatoires qui viendront se multiplier à $(k_0, \ldots, k_{n-1})$ coordonnée à coordonnée.

3.2.3 Représentation des données génomiques

Un chromosome est représenté comme une suite d'allèles : adénine (A), cytosine (C), guanine (G), et thymine (T). Un génome représente l'ensemble des chromosomes. Il existe un génome de référence par rapport auquel il est possible de décrire un autre génome, en n'indiquant que les différences. L'intérêt de cela est qu'encoder l'ensemble des différences avec un génome est beaucoup moins lourd que d'encoder un génome tout entier car les génomes de deux humains quelconques sont identiques pour l'immense majorité des allèles. [55] En effet, comme déjà dit, pour stocker un génome explicitement il faut environ 100Gb. La taille nécessaire pour une représentation ne décrivant que les différences entre le génome d'un individu et un génome de référence dépendra du nombre de mutations que possède l'individu mais ne dépassera pas les quelques Mb. Il suffit d'indiquer le chromosome, la position au sein du chromosome et la nature de la différence : cette différence est appelée mutation. Il existe 4 types de mutations différentes : la substitution, la délétion, l'insertion et la substitution multiple.

— Une substitution correspond à une différence sur un allèle. Le client possède à une position donnée un allèle différent par rapport au génome de référence. Par exemple GAT**A**CA peut devenir GAT**T**CA.
— La délétion correspond à une coupure d'un ou plusieurs allèles à partir d'une position. Par exemple GAT**AC**A peut devenir GATA.
— L'insertion correspond à un ajout d'un ou plusieurs allèles à partir d'une position. Par exemple GATACA peut devenir GATA**GC**CA.
— La substitution multiple est une différence sur plusieurs positions consécutives. Par exemple GAT**AC**A peut devenir GAT**TG**A.

Le format VCF *Variant Call Format* est un format de fichier qui permet d'encoder les mutations d'un individu. Chaque ligne du fichier correspond à une mutation présentée sous un format précis : chromosome, position, identification, allèle de référence, type de mutation et allèles de l'individu. Ces informations permettent de définir la mutation quelle qu'elle soit.

3.3 Travaux précédents

Ces dernières années, de nombreux de travaux ont eu lieu sur le stockage et l'utilisation des données génomiques sur un serveur distant. Les plus populaires sont ceux basés sur le chiffrement homomorphe. Par exemple McLaren et al. [56] ont proposé une solution basée sur un chiffrement additivement homomorphe pour effectuer des tests pharmacogénétiques sur des données chiffrées. Lauter et al. [57] ont montré comment utiliser un

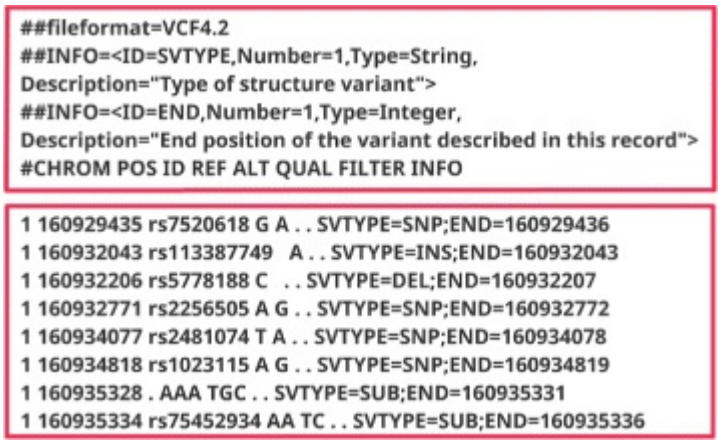

FIGURE 3.1 Un exemple de fichier VCF. La première partie représente l'en-tête qui décrit les méta-informations du fichier. La deuxième partie concerne les données : chaque ligne correspond à une mutation : le chromosome touché, la position de la mutation sur ce chromosome, la nature de la mutation, et les éléments qui composent cette mutation.

schéma homomorphe pour calculer des statistiques sur les données chiffrées, Wang et al. [58] en se basant sur ces travaux ont décrit une technique qui permet de calculer de manière privée une régression logistique.

Karvelas et al. [59] décrivent une solution permettant d'externaliser des données génomiques où un client va stocker dans une structure ORAM toutes ses données génomiques sous forme binaire. Une structure ORAM (pour *Oblivious RAM*) permet de réaliser des accès mémoire en lecture et écriture en occultant à cette mémoire où sont les données lues ou écrites). Il existes des méthodes très efficaces utilisant des arbres binaires (voir par exemple [60]). Dans la solution de Karvelas et al. il faut pour stocker les données en ORAM environ 2^{32} bits. Soit 2^{25} blocs de 128 bits. Ces blocs sont chiffrés par ElGamal. Les auteurs du papier ont décidé de stocker les données génomiques sous cette forme pour pouvoir directement y faire des opérations de manière privée. Ainsi un médecin peut faire des requêtes pour obtenir des données de manière privée. En moyenne une requête d'un bloc de 128 bits dure 12.39 secondes. Dans ce chapitre nous nous intéressons à comment tester la présence d'une donnée, sans avoir à la télécharger. Ceci nous permet d'obtenir un niveau de protection supplémentaire, comme on verra plus loin, et des performances meilleures.

3.4 Externalisation sécurisée de données génomiques

3.4.1 Le problème

Nous avons vu qu'il était important de se soucier de la confidentialité des données génomiques, ainsi que du contrôle de leur accès. Le contexte dans lequel nous nous plaçons est le suivant : un docteur possède les données issues du séquençage du génome d'un patient, sous la forme d'un fichier VCF, et veut le stocker en ligne sans que personne, y compris le serveur, ne puisse y accéder. Il est également nécessaire de cacher la façon dont on accède à ces données. En effet, en fonction de qui accède à une donnée et la fréquence d'accès, il est possible d'apprendre des informations. Par exemple un accès aux mutations ayant une influence sur l'efficacité de certains traitements pour le cancer de prostate peut donner des indications sur la maladie (cancer) sexe (masculin) et âge (probablement supérieur à 65 ans) du patient correspondant à un génome. Afin de pouvoir garantir un niveau élevé de sécurité, nous avons décidé d'avoir la possibilité de cacher le nombre de mutations que le client possède. On peut supposer qu'un nombre de mutations anormalement élevé pourrait entraîner des préjudices au client. Il est donc nécessaire de cacher cette information. Par ailleurs, il est nécessaire de cacher la taille des mutations afin de ne pas pouvoir faire de corrélation entre une mutation et sa taille.

Le médecin doit pouvoir réaliser une requête et recevoir une réponse sans échange intermédiaire. La requête identifie une mutation et la réponse est un booléen indiquant si le client possède ou non la mutation identifiée.

Pour récapituler :

— le serveur ne doit pas avoir accès aux données en clair ;
— le serveur ne doit pas apprendre quelles informations sont accédées ;
— le serveur ne doit pas apprendre quelle est la taille des données (nombre et taille des mutations) ;
— une requête doit permettre d'apprendre si une mutation est présente ou pas ;
— la réponse est un booléen qui indique la présence de la mutation.

3.4.2 La solution

Notre solution est fondée sur un système de chiffrement homomorphe et un PIR.

Le client a pour entrée un fichier VCF. Une ligne correspond à une mutation. Cependant l'encodage d'une mutation peut être particulièrement grand. En effet, il existe des insertions de plusieurs centaines d'allèles. Comme déjà expliqué, nous ne pouvons pas insérer des données de tailles différentes dans la base de données car cette information

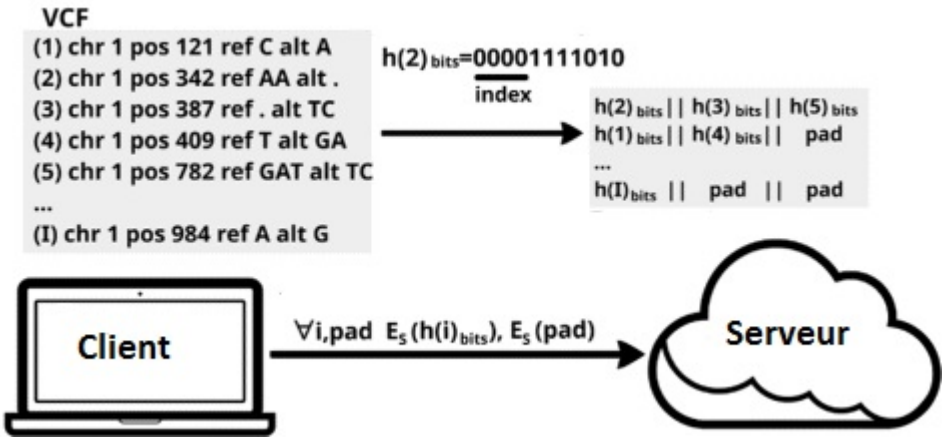

FIGURE 3.2 Lors de la phase d'initialisation, le client va hacher des informations de chaque ligne du fichier VCF. À chaque ligne sera associée une empreinte. Il va utiliser les premiers bits de cette empreinte pour définir l'index de la base de données où il enregistrera cette empreinte. Il va finalement chiffrer symétriquement chaque empreinte obtenue avant de l'enregistrer à son index. Il fera du bourrage pour simuler un fichier VCF de taille 5 millions de mutations puis fera du bourrage pour que chaque entrée de la base de données fasse la même taille. Pour simplifier, cette image ne décrit pas comment les hachés sont tronqués pour réduire la taille de la base de données.

peut permettre d'identifier partiellement ou totalement un client. C'est pour cette raison que le client applique une fonction de hachage sur chaque mutation.

Pour réduire la taille de la base de données le client ne va conserver qu'une partie du haché obtenu. Notons $h = h_1..h_y$, avec y la taille du haché. Les bits $h_1..h_x$ serviront d'index dans la base de données. Ensuite, $h_{x+1}..h_y$ est ajouté dans la base à l'index indiqué. S'il y a déjà des données stockées à cet index, le nouvel élément est ajouté au même index, de telle façon qu'à chaque index sera associée une liste d'éléments (chacun de ces éléments étant un hachage tronqué d'une mutation).

Une fois toutes les mutations insérées dans la base de données locale, le client va réaliser un bourrage pour simuler la présence de 5 millions de mutations dans la base de données (aucun être humain n'a autant de mutations connues pour le moment) [61], cela permet d'obtenir le même nombre de mutations pour chaque client. Puis il va réaliser l'empreinte cryptographique de ce bourrage et l'insérer avec le même processus. Pour que chaque ligne contienne le même nombre d'éléments, le client va ensuite réaliser un autre bourrage en ajoutant des éléments aléatoires jusqu'à ce que chaque index contienne une liste de taille identique. Il va chiffrer symétriquement, par bloc avec un mode compteur, ligne par ligne le fichier la base de données obtenue. Le client conserve en mémoire le compteur utilisé pour chaque position du chiffrement. Ce processus est décrit dans la figure 3.2 et formalisé dans l'algorithme 6.

Algorithme 6 : Initialisation de la base de données

Input : `mutations` vecteur de mutations issues d'un fichier VCF, x taille de l'index, `tailleligne` taille maximum d'une ligne, H fonction de hachage, E algorithme de chiffrement symétrique (mode, génération d'aléa et clé de chiffrement gérés en interne)
Output : $E(DB)$ Base de données initialisée

begin
  `sel` $\leftarrow \Omega$; Definir $h(\cdot) = H(\texttt{sel}||\cdot)$;
  for `mut` $\in$ `mutations` **do**
    $h_1..h_y = h(\texttt{mut})$; `index` $= h_1..h_x$; $DB^{h_1..h_x} = DB^{h_1..h_x} + h_{x+1}..h_y$;
  for $i \in [(\texttt{taille}(\texttt{mutations}) + 1)..5000000]$ **do**
    `alea` $\leftarrow \Omega$; $h_1..h_y = h(\texttt{alea})$; `index` $= h_1..h_x$; $DB^{h_1..h_x} = DB^{h_1..h_x} + h_{x+1}..h_y$;
  for $j \in [1..2^x]$ **do**
    Si `tailleligne` $<$ `taille`(DB^j) redémarrer l'algorithme;
    `alea` $\leftarrow \Omega$ tel que `taille`(`alea`) $=$ `tailleligne` $-$ `taille`(DB^j);
    $DB^j = DB^j +$ `alea`
  return $\underline{\mathbf{E}(\mathbf{DB})}$;

Le client envoie ensuite cette base de données chiffrée qui sera stockée par le serveur. Pour savoir si une mutation est présente ou non, le client va générer une requête PIR. Pour cela, il va reproduire le processus de la phase d'initialisation pour obtenir l'empreinte

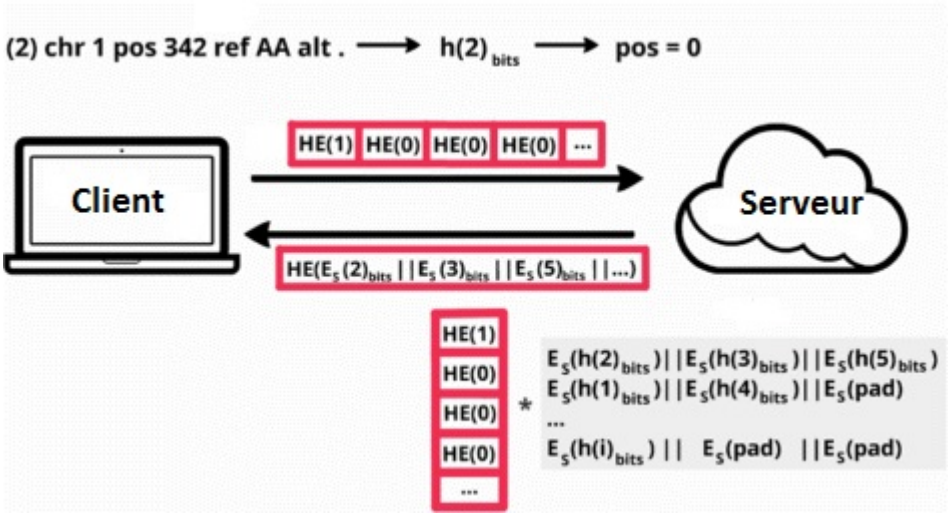

FIGURE 3.3 Lors de la phase de requête, le client forme une requête PIR pour savoir si une mutation est présente ou pas. Pour cela il va hacher la mutation qui l'intéresse puis réaliser une requête PIR pour récupérer la ligne ayant pour index les premiers bits de l'empreinte obtenue.

cryptographique de la mutation. Grâce à cela le client détermine la ligne qui peut contenir l'information concernant la mutation qu'il recherche. Le client va envoyer sa requête PIR pour télécharger toute la ligne qui contient l'information sur la mutation. Cependant, si l'on réalise un simple PIR, le client va récupérer beaucoup trop d'informations, ce que l'on ne veut pas nécessairement. En effet, puisque les données sont agrégées dans la base de données, si le client récupère une ligne entière, il va apprendre tous les éléments contenus sur cette ligne. Si on suppose par exemple que la personne qui fait la requête est un médecin et pas le patient lui même, il est préférable de cacher le plus d'informations possible.

Pour cela, en plus de la requête PIR, le client va également envoyer un chiffré, qu'on appellera requête de soustraction, qui va permettre d'obfusquer la réponse du serveur. Afin de comprendre cette obfuscation, il est nécessaire de donner quelques précisions sur le protocole PIR utilisé. Nous avons choisi d'utiliser un protocole PIR fondé sur RLWE en *full batching*. Comme vu précédemment (c.f Section 1.2.3), avec RLWE les clairs sont des polynômes (qu'il est possible de voir comme des vecteurs d'évaluations en différents points de ces polynômes). Dans notre protocole, les réponses PIR sont donc des vecteurs chiffrés, et nous paramétrons le protocole de telle façon qu'à chaque coordonnée de ces vecteurs est associé un unique haché (tronqué et chiffré) d'une mutation.

La requête de soustraction est un chiffré RLWE en *full batching* qui contient l'empreinte

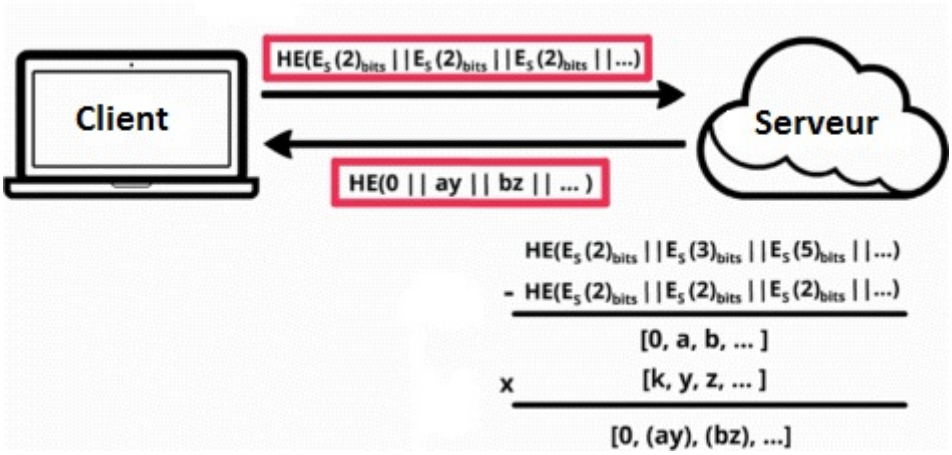

FIGURE 3.4 Un client réalisant une requête PIR classique sur la base de données va récupérer de nombreuses informations puisqu'une entrée dans la base de données correspond à de nombreuses mutations concaténées. Pour cacher ce surplus d'informations il va envoyer une requête de soustraction afin de faire apparaître un 0 dans le résultat du PIR si la mutation est présente. Il va ensuite faire en sorte que les résultats soient multipliés par des nombres aléatoires ce qui permettra d'éliminer toute autre information que si un résultat était nul ou pas.

chiffrée de la mutation recherchée à chaque coordonnée.

À la réception de ce chiffré le serveur fait une soustraction entre la réponse PIR et la requête de soustraction. Si la mutation est présente dans la réponse, cette opération va faire apparaître un zéro (voir Figure 3.4). Pour masquer les autres informations le serveur randomise chaque coordonnée (voir Section 3.2.2). Si l'une de ces coordonnées vaut zéro elle est inchangée par cette opération, mais toutes les positions qui ne sont pas à zéro (c'est-à-dire ne correspondant pas exactement à la mutation recherchée) seront obfusquées par un masque multiplicatif inconnu du client.

Une fois le résultat reçu, le client va le déchiffrer et chercher si la réponse possède un 0 à n'importe quelle position. Si c'est le cas, la mutation est présente.

Le client apprend également la position à laquelle la mutation a été stockée. Pour s'assurer que cela ne révèle pas d'informations il est possible de modifier ce protocole de façon à ce que le serveur change aléatoirement l'ordre des éléments au sein de chaque index.

3.4.3 Taille des communications

Comme décrit dans la Section 1.3.1, les protocoles PIR disposent de deux techniques génériques pour optimiser les coûts de communication : l'agrégation et la récursion. Pour

simplifier sa description, nous avons présenté notre solution sans ces techniques. Celles-ci viennent dans un deuxième temps réduire le coût des communications. En utilisant un facteur d'agrégation `agreg` et `dim` niveaux de récursion, et en notant `taillechiffre` la taille d'un chiffré et $L = 2^x$ le nombre de lignes dans la base de données, la taille des requêtes sera approximativement : $\texttt{taillechiffre} * \texttt{dim} * \lceil (L/\texttt{agreg})^{1/\texttt{dim}} \rceil$. En notant `tailleclair` la taille du message qui peut être absorbé par un chiffré du protocole PIR et `tailleligne` la taille d'une ligne dans la base de données, la taille de la réponse sera approximativement : $\texttt{tailleligne} * \texttt{agreg} * (\texttt{taillechiffre}/\texttt{tailleclair})^{\texttt{dim}}$. L'agrégation et la récursion réduisant la taille de la requête et augmentant celle de la réponse, un équilibre doit être trouvé pour réduire le temps total nécessaire aux communications.

La soustraction et randomisation sont effectuées par le serveur après le traitement PIR pour le premier niveau de récursion (après les niveaux ultérieurs la structure interne attribuant une mutation par coordonnée est généralement brisée). Cette requête devra avoir une taille suffisante pour pouvoir réaliser une soustraction à une position précise d'une ligne de la base de données et uniquement à cet endroit. La taille de la requête de soustraction sera donc : $\texttt{taillechiffre} * \lceil \texttt{tailleligne}/\texttt{tailleclair} \rceil$.

3.4.4 Sécurité

Objectif : Notre objectif est que l'utilisateur puisse tester la présence d'une mutation sans que le serveur puisse apprendre quoi que ce soit sur les données envoyées lors de la phase initiale, ou sur quels sont les éléments auxquels le client accède. Ceci peut en particulier être important si la récurrence des accès à une même donnée (e.g. par différents médecins) donne des indications sur une pathologie (e.g. maladie longue durée).

Modèle d'attaquant : On suppose que le serveur est honnête mais curieux, il suit le protocole mais peut essayer de tirer des informations sur les données du client. Ce modèle est souvent utilisé lorsque l'on travaille avec le calcul externalisé. En effet, un serveur malicieux pourrait être repéré lors d'audits de sécurité où le client enverrait un petit échantillon pour vérifier si le résultat qu'il reçoit est celui attendu. Nous considérons qu'il n'y a pas d'adversaire malhonnête sur le canal de communication, en supposant que les données sont protégées de bout en bout avec de l'authentification, de la confidentialité et de l'intégrité (par exemple en utilisant TLS 1.2). Le client qui possède les données, est supposé constamment authentifié et est supposé honnête.

Hypothèses cryptographiques : Nous supposons que le chiffrement symétrique ainsi que le chiffrement homomorphe utilisés sont IND-CPA pour des messages de taille arbitraire, pouvant dépasser un bloc (et donc composé de plusieurs sous-chiffrés). En utilisant

un argument hybride standard ceci revient à supposer l'IND-CPA pour un seul bloc chiffré. La fonction de hachage, quant à elle, n'a pas besoin d'être cryptographiquement sûre.

Proposition 1 (Indistingabilité des bases de données)**.** *Pour tout couple de génomes g_1, g_2 ayant moins de cinq millions de mutations par rapport à un génome de référence g, les distributions des bases de données chiffrées générées par l'algorithme 6 à l'initialisation sont indistingables.*

Démonstration. Soit A un algorithme qui distingue ces deux distributions avec un avantage non négligeable. Il est simple de construire un algorithme B qui contredit l'hypothèse d'IND-CPA pour le chiffrement symétrique. Pour cela il suffit de montrer que l'on peut construire deux messages m_1 et m_2 de même taille dont les chiffrés seront distingables avec une probabilité non-négligeable en utilisant A. Soit m_i obtenu en exécutant l'algorithme 6 pour le génome g_i en excluant la dernière étape de l'algorithme (le chiffrement symétrique final). On obtient m_1, m_2, deux clairs de même taille, que B propose pour un défi de type IND-CPA pour E, il reçoit un chiffré c d'une des deux bases de données et fournit ce chiffré à A. Si A répond g_i, B répond m_i au challenge c. L'existence de A est donc incompatible avec l'hypothèse de sécurité disant que le chiffrement symétrique est IND-CPA.

□

Proposition 2 (Indistingabilité des requêtes)**.** *Pour tout couple de mutations mut,mut′ dans un génome g, les distributions des requêtes générées pour tester la présence de la mutation sont indistingables par le serveur.*

Démonstration. Les requêtes générées pour tester la présence de *mut* et *mut′* sont des groupes de messages $m_1, ..., m_n, m'_1, ..., m'_n$ de même taille qui permettent de réaliser le PIR et la soustraction décrits dans la Section 3.4.2. En utilisant un argument hybride, l'avantage pour distinguer ces requêtes est le même avantage que pour distinguer des chiffrés de deux clairs divisé par n. L'existence d'un distingueur efficace pour les requêtes est donc incompatible avec l'hypothèse de sécurité disant que le chiffrement homomorphe est IND-CPA. □

Ces deux propositions assurent que le serveur ne peut exploiter les informations envoyées par le client ni pour apprendre des informations sur le génome, ni pour apprendre des informations sur les accès.

3.4.5 Exemple d'implémentation

Chiffrement symétrique : Nous avons choisi d'utiliser AES, standard actuel en chiffrement symétrique, avec une clé de 256 bits en mode compteur (CTR). Ce mode de chiffrement n'est pas authentifié, mais ceci correspond bien à notre modèle d'attaquant (honnête mais curieux) et à nos hypothèses cryptographiques (IND-CPA). Nous utilisons comme compteur un vecteur d'initialisation aléatoire de 64 bits concaténé avec un index décrivant la position de la mutation. Le vecteur d'initialisation est envoyé avec la base de données lors de l'initialisation et téléchargé par le client avant de faire une requête s'il ne l'a pas en cache.

Fonction de hachage : Bien que nous ne demandons pas de propriété cryptographique particulière à notre fonction de hachage, au vu du faible impact qu'elle a dans les coûts, nous avons utilisé la fonction de hachage SHA256.

Bourrage : Pour le bourrage nous avons utilisé le schéma PKCS 7 [5], où la valeur de chaque élément de bourrage est égale au nombre d'éléments. Par exemple si nous avons besoin de 3 éléments le bourrage sera "3 3 3".

Protocole PIR et chiffrement homomorphe : XPIR [27] est une librairie permettant d'effectuer un cPIR de manière efficace. La librairie utilise un schéma de chiffrement homomorphe basé sur RLWE et inspiré de BGV (voir section 1.2.4) implémenté sous NFLlib [62]. Ce schéma ne permet de faire que des additions et des absorptions. Sur un ordinateur portable MSI GT60 avec un processeur i7-3630QM 2.67GHz, le serveur est capable de traiter une base de données suite à une requête à un débit d'environ 20 Gbit/s. La base de données qu'on aura à traiter aura une taille très inférieure à 20Gbits, et donc le temps de calcul du PIR sera faible. La taille de la requête PIR en revanche sera significative et ce sera donc le temps de transfert réseau qui limitera principalement l'efficacité de notre protocole.

Nous avons choisi d'utiliser XPIR pour notre solution. Cependant, il est impossible avec XPIR en natif de réaliser la multiplication coordonnée par coordonnée que doit réaliser le serveur avant le renvoi final de la réponse. Nous avons dû ajouter un nouveau système de chiffrement sous XPIR. La librairie a été modifiée pour être basée sur le schéma de chiffrement homomorphe de Fan et Vercauteren (FV)[20], nous avons utilisé l'implémentation de FV-NFLlib[34].

Nous décrivons les différents paramètres du cryptosystème de la sorte FV-A-B-C-D, FV est le schéma utilisé,

— A représente le nombre de bits de sécurité,

5. https ://tools.ietf.org/html/rfc2315

— B représente le degré des polynômes,
— C représente la taille des coefficients dans le domaine des chiffrés,
— D représente le nombre de bits utiles que peut absorber un coefficient.

Pour obtenir A avons utilisé l'outil de référence actuel pour évaluer la sécurité d'un système de chiffrement basé sur LWE ou RLWE, développé par Martin Albrecht [63].La sécurité indiquée pour le schéma de chiffrement homomorphe est le retour (arrondi) de la fonction d'évaluation de cet outil.

Dans notre protocole, il est nécessaire que le traitement effectué par le serveur conserve dans la réponse un 0 si la mutation est présente après les phases de soustraction et multiplication par des nombres aléatoires. Pour cela, nous avons considéré jusqu'ici que $D = y$ mais il suffit de choisir D tel que $y = k * D$ pour un entier k. Si on choisissait D autrement, soit il y aurait des bits du clair non utilisés, soit il y aurait des coefficients contenant des informations sur plusieurs mutations. Pour ces coefficients il ne serait pas évident d'obtenir un 0 lors de la phase de soustraction.

Taille des empreintes : La taille y des empreintes stockeés dans la base de données joue un rôle très important. En effet plus la longueur du haché est petite, et plus nous augmentons les chances de faire apparaître une collision entre deux mutations. Si un client essaye de tester la présence d'une des mutations menant à la collision, alors que c'est l'autre mutation qui est présente, il obtiendra un faux positif. Par exemple si on suppose $y = 48$, la probabilité de faux positif, quand un client veut tester la présence d'une mutation donnée, est d'environ $5000000/2^{48} = 2^{-25}$. Cela nous paraît un compromis acceptable étant donné que la probabilité d'erreur lors d'un séquençage ADN est bien plus importante de nos jours (de l'ordre de 2^{-13}) [64]. Il serait possible d'augmenter la valeur de y à par exemple 96 pour avoir une probabilité de faux positif très faible. L'inconvénient serait que cela doublerait la taille de la base de données, et donc de la requête de soustraction et de la réponse, ce qui augmenterait le coût réseau significativement. Prendre $y = 48$ est un compromis que nous avons fait pour avoir une base de données relativement petite et ainsi avoir un protocole rapide.

Nombre d'entrées dans la base : Le choix du nombre d'entrées dans la base de données, 2^x, permet de jouer sur l'agrégation naturelle des données. En effet, plus x sera grand, moins il y aura d'empreintes par entrée dans la base, ce qui entraînera une taille de réponse plus petite. Inversement, plus x sera petit, plus la requête sera petite et plus la répartition des hachés dans les lignes de la base sera homogène ce qui permettra d'utiliser moins de padding. En pratique, nous avons choisi $x = 13$ valeur qui nous permet de stocker 5 millions d'entrées en garantissant par la loi des grands nombres qu'il n'y aura pas plus de 716 empreintes par ligne (probabilité d'environ 0.99). Suite à ce choix, nous

Configuration	**Sans soustraction**	**Etendue**	**Soustraction**	**Securisée**
y	48	48	48	48
x	13	16	13	13
$L = 2^x$	8192	65536	8192	8192
`tailleligne`	716	130	716	716
Chiffrement	FV-80-1024-62-14	FV-80-1024-62-14	FV-80-1024-62-12	FV-172-2048-62-12
Agrégation	3	15	3	3
Récursion	2 (53*52)	2(67*66)	2(53*52)	(53*52)

TABLE 3.1 Différents paramétrages possibles. Le premier permet d'illustrer un gain de performance possible quand la phase de soustraction finale n'est pas demandée au serveur. Le deuxième permet de mettre en relief qu'augmenter la valeur de x peut induire une perte de performance. Le troisième est le paramétrage nominal. Le quatrième est similaire au paramétrage nominal mais avec une sécurité plus élevée.

avons également choisi de fixer la taille du padding à `tailleligne` = 716. Si jamais une ligne contient plus de 716 empreintes il est nécessaire de reconstruire la base de données en utilisant un nouveau sel pour la fonction de hachage.

3.4.6 Résultats

Pour les expérimentations, nous avons implémenté le serveur et le client en C++ sur une machine virtuelle Ubuntu 64-bit avec 4Go de RAM et 250Go d'espace disque. La machine hôte est un MacBook Pro avec un processeur Intel Dual-Core i7 3.1GHz. Nous avons simulé un lien de 10Mb/s bidirectionnel. Chaque mesure est la moyenne de 10 tests indépendants.

Nous évaluons 4 types de configurations différentes présentées dans la table 3.1. La première configuration représente le cas où le client est le possesseur des données et veut vérifier s'il possède une mutation. Dans ce cas, puisque le client est directement le propriétaire des données, il n'y a pas fuite d'information associée à une réponse envoyant une ligne complète, et on peut supprimer la phase de soustraction. Nous avons choisi ces paramètres pour illustrer ce cas d'usage qui bénéficie des meilleures performances possibles puisqu'il induit un coût réseau moindre.

Nous avons utilisé l'optimiseur de paramètres de XPIR pour le choix des paramètres de chiffrement, de l'agrégation et de la récursion. Pour obtenir ces valeurs, XPIR lance des tests sur plusieurs jeux de paramètres et choisit le meilleur de façon empirique. Nous réalisons ici les mêmes hypothèses que l'optimiseur : les informations (requête, réponse) sont envoyées au fur et à mesure qu'elles sont générées et la base de données est supposée être pré-traitée par le serveur et en RAM.

La deuxième configuration est une configuration avec une base de données étendue

Configuration	Sans soustraction	Etendue	Soustraction	Securisée
Préparation des données (s)	19.2	21	19.6	21.3
Taille du fichier VCF (Mo)	35	51	35	35
Importation	0.6	1.08	0.71	0.8
Génération requête (s)	0.013	0.017	0.011	0.025
Envoi de le requête (s)	1.37	1.74	1.49	2.88
Génération de le réponse	0.38	0.55	0.46	0.56
Envoi de la réponse	1.03	1.04	1.26	1.39
Traitement de la réponse (s)	0.4	0.58	0.49	0.34
Temps total (s)	2.4	2.8	2.7	4.3

TABLE 3.2 Temps obtenus pour les différentes configurations décrites dans la table précédente. Par rapport à la configuration nominale nous pouvons remarquer que : éliminer la soustraction permet de réduire le temps total de dix pourcent ; la configuration étendue augmente légèrement le temps total ; et l'augmentation de la sécurité a un impact significatif mais raisonnable.

(avec plus de lignes concaténant moins d'empreintes chacune). Cette configuration montre qu'un x plus grand donne lieu à une taille de base de données $(y - x) * 2^x * \texttt{tailleligne}$ plus importante et à des coûts réseau plus importants. Globalement pour un x trop grand, la loi des grand nombres s'applique moins bien et il faut une quantité importante de padding, et pour un x trop petit la taille de la réponse grandit tellement qu'il n'est plus possible de faire varier les paramètres du protocole PIR pour équilibrer les communications.

La troisième configuration est la configuration nominale et représente le protocole avec la soustraction et multiplication par des nombres aléatoires pour cacher le surplus d'informations. La dernière configuration illustre le même cas que la troisième mais augmente la sécurité à 172 bits de sécurité au vu des attaques actuellement connues.

Les différentes configurations proposées sont évaluées en fonction du temps total de traitement de la requête en incluant les temps de transmission et en réalisant les mêmes hypothèses que l'optimiseur de XPIR : requête envoyée au fur et à mesure qu'elle est générée ; réponse envoyée au fur et à mesure qu'elle est générée ; réponse déchiffrée au fur et à mesure qu'elle est reçue. La Table 3.2 présente les temps nécessaires aux différentes opérations ainsi qu'un autre critère de performance : la taille du fichier VCF. Il est important de remarquer que cette dernière est la taille après les transformations réalisées par le client et représente donc le coût de stockage en RAM. Au niveau des temps il y a bien sûr les opérations nécessaires au traitement d'une demande client, mais également le temps nécessaire pour la génération et envoi de la base de données initiale

(ligne Préparation des données), ainsi que le temps nécessaire au serveur pour pré-traiter cette base de données et la mettre en mémoire (ligne Importation).

Le temps total représente le temps total de traitement observé par le client et peut être approché par le coût théorique prenant en compte les pipelines existants : génération/envoi de la requête d'un côté ; et génération/envoi/déchiffrement de la réponse de l'autre. Ainsi, par exemple pour la première colonne, il est possible de vérifier que $\max(0.013, 1.37) + \max(0.38, 1.03, 0.4) = 2.4$.

La configuration par défaut montre qu'un client peut récupérer une information sur une mutation en moins de 2.5s tout en protégeant ses données et la façon dont il y accède.

Dans la solution nominale, le coût est principalement dû à l'envoi de la requête et de la réponse. Ce temps peut être réduit considérablement avec une bande passante plus importante, comme le montre la Table 3.2. En effet, le temps de génération de la requête est dix fois plus court que le temps d'envoi et celui de la réponse environ trois fois plus court. En considérant une bande passante de 100MB le temps de traitement de la réponse représenterait la majeure partie des coûts, il faudrait un temps total d'environ $0.65s$ pour la configuration par défaut.

La configuration étendue donne lieu à une base de données plus grande que la configuration nominale, et donc les temps de génération de la réponse et de transmission empirent légèrement, ce qui mène à une diminution de performances dans les deux tables. Pour la configuration sans soustraction, il est possible d'utiliser un système de chiffrement absorbant plus d'information car la multiplication finale réalisée après la soustraction est évitée, et donc le coût des transmissions ainsi que le coût de génération de la réponse sont réduits ce qui à nouveau est observable dans les deux tables. Les performances obtenues avec la configuration sécurisée montrent que l'on peut choisir au déploiement une sécurité accrue d'un facteur environ 2 pour un cout qui ne croit que d'un facteur inférieur à 2.

3.4.7 Le challenge IDASH

IDASH [65] est une compétition internationale annuelle autour des techniques qui permettent de déporter des données génomiques vers un serveur distant tout en préservant la vie privée des utilisateurs. Ce challenge a déclenché nos recherches sur le sujet exposé sur ce chapitre. En 2016, le challenge IDASH proposait une tâche concernant le stockage sécurisé. Pour cette tâche, il existait plusieurs contraintes. La solution au challenge devait cacher les données génomiques, cacher l'élément accédé dans la base de données, utiliser le chiffrement homomorphe et récupérer moins de 20 informations sur les mutations (lors d'une requête on pouvait récupérer 19 autres informations). Cependant, le challenge n'imposait pas de cacher le nombre de mutations d'un client. C'est pour cette raison que

Configuration	**1a**	**2a**	**3a**
y	48	48	48
x	13	17	17
$L = 2^x$	8192	131072	131072
`tailleligne`	6	6	6
Chiffrement	FV-80-1024-62-12	FV-80-1024-62-12	FV-80-1024-62-12
Agrégation	4	4	4
Récursion	2 (46*45)	3(32*32*32)	3(32*32*32)
Configuration	**1b**	**2b**	**3b**
y	48	48	48
x	3	8	8
$L = 2^x$	8	256	256
`tailleligne`	1344	512	512
Chiffrement	FV-80-1024-62-14	FV-80-1024-62-14	FV-80-1024-62-14
Agrégation	1	1	1
Récursion	1 (8)	2(16*16)	2(16*16)

TABLE 3.3 Les différentes configurations utilisées pour le challenge. Les configurations finissant par la lettre b correspondent au protocole avec une étape de soustraction, et celles marquées par la lettre a correspondent au protocole sans cette étape. Les configurations commençant par 1 (respectivement 2 et 3) correspondent à un fichier VCF de 10 000 mutations (respectivement un fichier de 100 000 mutations et 50 fichiers de 10 000 mutations).

nous avons décidé pour ce challenge d'effectuer seulement le bourrage pour que chaque ligne de la base de données fasse la même taille et de ne pas simuler un fichier de taille 5 millions de mutations. Ce bourrage ne cache pas le nombre de mutations du client mais permet d'obtenir une base de données plus compacte qu'avec notre approche initiale. De plus, pour le challenge, seules les mutations du type substitution étaient considérées, dans notre solution nous avons décidé de garder le système de hachage et donc de considérer les 4 types de mutations.

Les Tables 3.3 et 3.4 montrent les différentes configurations utilisées dans le cadre du challenge et les résultats associés. Il y a 3 tests différents qui sont utilisés (1) Requête sur 4 mutations pour un fichier VCF de taille 10 000, (2) Requête sur 4 mutations pour un fichier VCF de taille 100 000, (3) Requête sur 4 mutations pour 50 fichiers VCF de taille 100 000. Nous avons opté pour deux scénarios différents, le premier a été celui soumis pour le challenge et utilise une base de données étendue pour ne récupérer qu'environ 20 éléments. Le second utilise une base de données avec moins de lignes cumulant plus de mutations avec l'ajout de l'étape de soustraction pour cacher les informations supplémentaires.

Configuration	**1a**	**2a**	**3a**
Préparation des données (s)	0.04	0.4	20.2
Taille du fichier VCF (Mo)	0.3	4.72	235.9
Importation	0.13	1.89	94.5
Génération requête (s)	0.04	0.05	0.67
Envoi de le requête (s)	4.77	5.03	62.9
Génération de le réponse	0.29	4.27	53.4
Envoi de la réponse	0.53	5.19	64.9
Traitement de la réponse (s)	0.34	4.41	55.1
Temps total (s)	5.5	13.3	128.1
Configuration	**1b**	**2b**	**3b**
Préparation des données (s)	0.037	0.36	19.2
Taille du fichier VCF (Mo)	0.06	0.79	39.5
Importation	0.002	0.03	1.4
Génération requête (s)	0.008	0.016	0.2
Envoi de le requête (s)	0.736	1.78	22.3
Génération de le réponse	0.008	0.1	1.27
Envoi de la réponse	0.31	1.15	14.5
Traitement de la réponse (s)	0.05	0.16	2.05
Temps total (s)	1.07	2.95	37

TABLE 3.4 Temps par opération et espace mémoire pour les différentes configurations.

Pour la configuration étendue, les lignes de la base de données sont complétées avec du bourrage pour contenir au plus 6 mutations à quoi vient s'ajouter une agrégation de 4 pour le PIR. Une analyse simple par la loi des grands nombres permet de montrer que le nombre moyen d'éléments par ligne est de 1 et qu'il est peut probable qu'en agrégeant 4 lignes le nombre de mutation contenu dépasse 20. Considérant la configuration 2a. La loi binomiale converge vers une loi normale. L'écart type est d'environ 0.87. Il faut donc que chaque ligne dépasse 6 fois l'écart type pour avoir au moins 5 éléments. La probabilité qu'une ligne contienne 5 éléments est d'environ 2^{-29}. En supposant que la fonction de hachage a de bonnes propriétés statistiques, et en se réduisant au cas où on atteint 20 éléments par agrégation de 4 lignes de 5 éléments, on peut déduire que la probabilité d'un tel événement est de 2^{-116}.

La Table 3.3 donne les paramètres pour les différentes configurations proposées, et la Table 3.4 présente les performances obtenues.

L'évaluation se fait dans la Table 3.4 de la même façon que précédemment : le critère principal est le temps total, et de façon annexe sont données la capacité de stockage requise et le temps nécessaire à l'importation.

Pour un fichier VCF de 10 000 mutations le temps total est de 5.5 secondes sans soustraction et 1.07 secondes avec. Pour un fichier VCF de 100 000 mutations il est de 13.3 secondes sans soustraction et 2.95 secondes avec. Enfin pour 50 fichiers VCF de 10 000 mutations chacun il est de 128 secondes sans soustraction et 37 secondes avec.

La grande différence de temps entre les deux méthodes (avec et sans soustraction) est due au fait que quand on réalise la soustraction nous pouvons choisir des paramètres optimaux pour réaliser le PIR puisque nous n'avons pas besoin de nous soucier de l'information supplémentaire, fournie au client, venant de l'agrégation.

Le scénario des 50 fichiers VCF montre que le coût de notre solution est essentiellement linéaire avec le nombre de mutations recherchées, puisque pour chaque mutation recherchée nous devons réaliser un PIR indépendant.

Nous avons été parmi les quatre finalistes du challenge. Une comparaison succincte des différentes approches proposée par les équipes est présentée dans la table ci-dessous.

Equipes	**Notre approche**	**A et B**	**C**
Requête multiples	Non	Oui	Oui
Taille de la base de données	=	=	+
Considération des mutations	Toutes	Substitutions	Toutes
Coût principal	Réseau	Calculatoire	Calculatoire
Temps total pour une requête	+	=	-
Temps pour une requête multiple	=	+	=

L'équipe (A) composée de Kristin Lauter, Kim Laine, Hao Chen, (Microsoft research), Gizem Cetin, (Worcester Polytechnic Institute), Peter Rindal, (Oregon State University), Yuhou (Susan) Xia, (Princeton University) [66] et l'équipe (B) composée de Jung Hee Cheon, Miran Kim, Yongsoo Song, (Seoul National University) [67] ont eu une approche similaire. Ils utilisent un schéma proche de FV (décrit dans la Section 2.1.2) pour faire un test d'égalité privé sur une base de données formatée différemment pour les deux équipes. Leurs formatages particuliers ne sont possibles que parce qu'ils considèrent les mutations de type substitution et non les autres. En effet, ces mutations ont la propriété de pouvoir être encodées sur un nombre court et fixe de bits suffisamment petit comme pour faire un test d'égalité homomorphe. Il est possible dans leur approche de réaliser une requête multiple, la forme de la base de données est dépendante du nombre maximal de mutations recherchées dans une requête.

L'équipe (C) composée de David Hellmanns, Martin, Henze, Jens Hiller, Ike Kunze, Sven Linden, Roman Matzutt, Jan Metzke, Marco Moscher, Jan Pennekamp, Felix Schwinger, Klaus Wehrle, Jan Henrik Ziegeldorf, (RWTH Aachen University, Germany) utilise une comparaison privée utilisant un filtre de Bloom [68].

3.4.8 Conclusion et perspectives

Nous avons présenté dans ce chapitre une solution à un des potentiels problèmes du stockage externalisé de données génomiques. Notre solution permet, en outre du stockage confidentiel, d'interroger le serveur sur la présence ou l'absence d'une mutation dans le génome stocké. Nous assurons la confidentialité des données stockées, mais également celle de la requête, vis-à-vis du serveur.

Avantages. Notre solution est avantageuse sur plusieurs points.

— Une protection de la vie privée optimale. En faisant du bourrage sur le fichier VCF et en utilisant une fonction de hachage pour réduire la taille des données, nous nous assurons que, peu importe le nombre de mutations (jusqu'à un maximum de 5 millions), les fichiers sont indistingables. En utilisant le PIR pour accéder aux données, nous pouvons cacher la manière dont nous accédons aux données, même si l'on accède deux fois de suite à la même donnée. Il est ceci dit possible de réduire les exigences de vie privée pour avoir de meilleures performances comme l'illustre l'exemple du challenge IDASH.

— La confidentialité des données. Chaque mutation est représentée par un empreinte de taille fixe, et le nombre d'empreintes stockées est également fixe, par conséquent il ne peut pas avoir d'attaque sur la taille des données. Ces données étant chiffrées avec un chiffrement symétrique, il n'est également pas possible d'obtenir des

informations sur quelles sont les empreintes stockées sans casser la sécurité du système de chiffrement.

— Elle s'adapte facilement à une sécurité plus grande. Grâce à l'utilisation de la cryptographie basée sur les réseaux, il est facile d'augmenter la sécurité, améliorer la sécurité par un facteur 2 réduit les performances d'un facteur légèrement inférieur à 2 seulement.

— Peu de capacité de stockage nécessaire. En ne stockant qu'une partie d'un haché, nous réduisons significativement la taille des données à stocker. En incluant le coût dû au bourrage un fichier VCF est stocké en environ 35 Mo.

— Un temps d'aller-retour très court. Notre approche était significativement plus rapide que toutes les autres approches présentées au challenge IDASH, quand il s'agissait de tester la présence d'une unique mutation.

— Minimisation de l'information. Le client apprend seulement s'il a ou non la mutation grâce à l'étape de soustraction rajoutée au PIR.

— Le fait de considérer les quatre types de mutations. La plupart des méthodes considèrent uniquement les substitutions.

Limites. Cependant, notre stratégie a aussi quelques limites. La première est le taux d'erreur associé au fait d'utiliser des empreintes de petite taille. Néanmoins, comme décrit dans la section paramétrage, il est possible de multiplier la taille de ces dernières par 2 pour avoir une forte garantie de non-collision tout en gardant de bonnes performances (augmentation d'environ 40% dans le temps total). La base de données serait alors deux fois plus grande. La seconde limitation est que si le nombre de fichiers VCF dans la base de données est trop grand alors il n'est plus possible de pré-importer les données dans la RAM. Dans ce cas, il faudrait lire les données directement sur un disque lors du traitement d'une requête ce qui pourrait ralentir la solution si le temps de génération de la réponse est plus lent que le temps d'envoi. Enfin, le principal défaut de la solution est que, si l'on s'intéresse à plusieurs mutations, il faut faire plusieurs requêtes. La complexité est linéaire en le nombre de mutations qui nous intéressent. Faire une requête multiple permettant de savoir si le client possède une première mutation ET une deuxième mutation pose beaucoup de problèmes avec notre solution. Il faudrait pour cela créer une requête PIR avec deux éléments valant 1.

— La récursion utilisée lors du PIR ne permet pas de faire une requête sur plusieurs éléments sans récupérer de l'information supplémentaire. En effet, supposons une récursion de 2, la première requête PIR contiendrait 2 éléments valant 1 et la seconde également. Un problème émerge, les deux éléments à 1 de la première requête se combinent aux deux éléments à 1 de la deuxième et le résultat contient

une somme correspondant aux quatres combinaisons possibles, dont deux qui n'intéressent pas a priori le client.

— L'aggrégation de multiples empreintes par ligne entraîne également des problèmes. En effet, quand une requête PIR contient plusieurs valeurs à 1 (une par ligne contenant une empreinte qui intéresse le client), le résultat du PIR sera la somme de deux lignes. Ainsi, les empreintes que l'utilisateur souhaite tester se voient ajoutées à des empreintes dont il ne connaît pas la valeur, rendant le résultat inutile.

Il serait donc nécessaire de ne pas utiliser d'agrégation ni de récursion. Pour cela la taille de l'index x devrait être aussi grand que la taille des données y ce qui implique que la base de données aurait 2^y éléments. Pour une base de données d'une telle taille, le protocole PIR aurait un coût exorbitant. Il faudrait pour pouvoir faire ce type de requête multiple une solution radicalement différente.

Améliorations. La première idée est de trouver un encodage pour les mutations plutôt que d'utiliser une fonction de hachage. Cela permettrait d'éliminer la possibilité d'avoir un faux positif, voir même utiliser d'autres opérations sur les données. Par exemple, si l'on considère uniquement les substitutions : chromosome (5 bits), position (28 bits), allèle altérée (2 bits). En considérant toutes les mutations possibles, il faudrait rajouter un champ mutation (2 bits), un champ taille et rendre le champ allèles altérés plus grand. S'il n'est pas possible de stocker des mutations de taille différente, il faut alors faire un bourrage pour que toutes les mutations fassent la même taille ce qui augmente significativement la taille de la base de données car certaines insertions peuvent introduire plus d'un millier allèles. Cependant si l'on ne considère que les substitutions, il est possible de réaliser un encodage sur 35 bits et stocker un chiffré de cet encodage. Cela permet un gain significatif en performance puisque la base de données est plus petite et en supplément cet encodage n'induit plus de faux positifs.

Une alternative envisageable en utilisant cet encodage de 35 bits est de ne stocker qu'un tableau booléen correspondant à l'ensemble des mutations possibles. Un 1 représente la possession de la mutation dans la base de données à la position correspond à l'encodage, le 0 la non possession. Le problème avec cette alternative est que la base de données contiendrait 2^{35} éléments ce qui implique que le protocole PIR générera significativement plus de communications que notre solution. En revanche, l'avantage de cette solution est qu'il est possible de faire des ET logiques en une seule requête, puisque si nous voulons vérifier si nous avons une première mutation ET une deuxième mutation il suffit de réaliser un PIR et vérifier si le résultat obtenu est 2.

La dernière solution envisageable est de ne stocker que les empreintes de taille 48 dans

la base de données, puis d'envoyer 48 éléments de requête homomorphe. Chaque élément correspond à 1 bit de la mutation que nous voulons vérifier. Soit a_i le bit i d'un élément dans la base de données et q_i le bit i de la requête. En faisant du calcul homomorphe modulo 2, on peut tester l'égalité bit à bit en faisant $t_i = 1 + q_i + a_i$. t_i est le résultat du test d'égalité pour le bit i. Ensuite, en faisant un ET entre tous les t_i on peut vérifier si le client possède la mutation. Cette opération doit être réalisée sous forme d'arbre binaire pour réduire la profondeur multiplicative.

Les avantages de cette stratégie sont que la réponse est binaire et ne dépend pas de la représentation envisagée. Il est facile de savoir si l'on possède une première mutation ET une deuxième mutation puisqu'il suffit de réaliser un niveau multiplicatif en plus à la fin. Il n'y a pas de bourrage nécessaire à part pour cacher le nombre de mutations. Le principal inconvénient de cette solution est que le temps de génération de la réponse est significativement plus important que le temps total de notre approche.

Chapitre 4

Ensemble maximal privé

Dans le chapitre précédent, nous avons présenté les problématiques liées à la protection de la vie privée dans le cadre du traitement de données génomiques, en particulier du point de vue de leur stockage. Dans ce chapitre, nous allons poursuivre nos travaux sur les données génomiques, en nous intéressant cette fois à leur traitement, et en prenant l'exemple des GWAS *Genome-Wide Association Studies*.

Les GWAS concernent l'étude de la corrélation statistique entre les mutations d'une personne et son appartenance à un groupe. Les GWAS sont probablement les études les plus fréquentes pour établir un lien entre les génomes et les phénotypes [51]. Le principe est de trouver des groupes de mutations qui ont une forte probabilité d'indiquer une caractéristique particulière d'un individu. Une telle caractéristique peut être physique, par exemple la couleur de ses yeux, de sa peau, etc. Elle peut également être mentale, telle que portant sur le comportement de la personne, ou concernant la probabilité d'avoir certaines maladies [54].

Dans le domaine de la bio-informatique, le calcul d'un ensemble maximal est souvent utilisé pour connaître la corrélation entre un génome individuel et les génomes d'une base de données [69]. En effet, il est intéressant de chercher quelles sont les longues séquences communes entre le génome individuel et les génomes d'une collection afin de déterminer les points communs, i.e. les corrélations, entre ces différents génomes. Par exemple, des sous-chaînes égales, suffisamment longues et nombreuses permettent de trouver des zones identiques significative d'une descendance. De la même manière, une longue séquence commune avec une collection dont les individus ont une caractéristique commune permet également d'inférer que le génome porte probablement également cette caractéristique.

4.1 Problématique du calcul des ensembles maximaux

L'objectif est donc ici de calculer l'ensemble des ensembles maximaux, d'une longueur supérieure à un seuil donné, entre un génome d'entrée (la requête) et une base de données. Les données sont représentées par des vecteurs booléens, dont le bit à la position i vaut 1 si le génome possède la substitution i. (voir Section 3.2.3) Soit une requête R et une base de données G, la liste de substitutions consécutives positionnées entre a et b ($a < b$) est un ensemble maximal si :

1. Il existe un génome G^j appartenant à la base de données tel que G^j et R sont égaux entre a et b, c'est-à-dire $G^j_i = R_i$ pour $i \in [a; b]$.
2. L'ensemble ne peut pas être étendu ni par la gauche ni par la droite, c'est-à-dire que $G^j_{a-1} \neq R_{a-1}$ et$G^j_{b+1} \neq R_{b+1}$
3. Il n'existe pas d'autre entrée dans la base de données qui peut étendre cet ensemble. En d'autre terme, il n'existe pas de génome G^k avec $k \neq j$ tel que $G^k_i = R_i$ pour $i \in [a; b]$ où $G^k_{a-1} = R_{a-1}$ ou $G^k_{b+1} = R_{b+1}$.

Nous souhaitons calculer l'ensemble des sets maximaux de manière privée et en minimisant les interactions.

4.2 Travaux précédents

Dans cette section, nous présentons les solutions de l'état de l'art au problème du calcul des ensembles maximaux, dans un premier temps les solutions non respectueuses de la confidentialité des données, puis les solutions privées.

4.2.1 De manière non privée

PBWT *Positional Burrows–Wheeler Transform* [69] est un algorithme développé par Richard Durbin pour calculer des ensembles maximaux de manière non privée. Il s'agit actuellement de l'algorithme le plus rapide de l'état de l'art. La stratégie de l'algorithme consiste en l'utilisation d'un tri distinct pour chaque sous-chaîne de la collection de taille M. Pour chaque position k entre 0 et N l'ordonnancement est différent. Pour une position k, chaque sous-chaîne de la collection est triée de façon à ce que le préfixe en ordre inverse (allant de $k-1$ à 0) soit trié de façon naturelle. Prenons un exemple avec 4 chaînes : $e_0 = 0010$, $e_1 = 1101$, $e_2 = 1010$, $e_3 = 0001$. Pour la position $k = 2$ (la première position est $k = 0$) le tri serait e_3, e_1, e_0, e_2 puisque l'ordre inverse du préfixe à partir de la position 2 est 100 pour e_0, 011 pour e_1, 101 pour e_2 et 000 pour e_3. Pour chaque

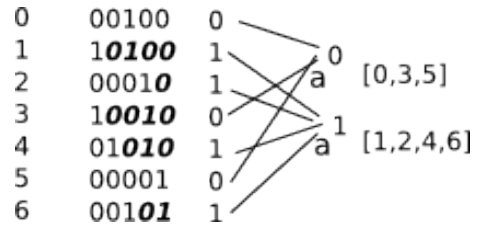

FIGURE 4.1 Exemple de PBWT, en italique et gras nous avons ce que représente d, le plus grand préfixe commun avec l'entrée précédente. La base de données est bien triée. Si l'élément suivant est 0 on stocke l'index dans le tableau a^0 sinon on le stocke dans a^1. On voit que le passage d'une position à une autre est élémentaire. L'ordre à la fin de ce passage sera 0351246.

position, l'ordre du tri est conservé dans un tableau a_k. Ici a_2 vaut $[3, 1, 0, 2]$. Calculer a_{k+1} à partir de a_k est rapide. Pour cela, l'algorithme va utiliser 2 sous-tableaux a^0 et a^1. Pour chaque élément de a_k, dans l'ordre du tri, on va intégrer l'élément à la fin d'un sous-tableau en fonction de sa valeur en $k + 1$. Pour finir, il suffit de concaténer les 2 sous-tableaux.

Dans l'exemple, e_3 est le premier élément, à la position 3 il vaut 1 il sera donc dans le sous-tableau des 1, $a^1_{k+1} = [e_3]$. Puis $a^1_{k+1} = [e_3, e_1]$. Puis $a^0_{k+1} = [e_0]$ et $a^0_{k+1} = [e_0, e_2]$. En concaténant on obtient $a_{k+1} = [e_0, e_2, e_3, e_1]$.

Ce tableau est construit en début d'algorithme. De la même façon PBWT construit un tableau d, qui contient pour chaque position k la séquence commune avec l'élément antérieur. Par exemple avec $e_1 = 0010$ et $e_2 = 1010$ à la position 3, la séquence commune de e_2 avec e_1 sont les 3 derniers éléments. On conserve dans d uniquement la position de départ de cette séquence, dans l'exemple $d_3[1] = 1$. Il existe également un moyen rapide pour calculer d_{k+1} à partir de d_k

L'algorithme utilisé ensuite pour trouver les ensembles maximaux entre une requête Q et la collection se base sur l'algorithme suivant où il est rapide de passer de k à $k + 1$. Soit e_k la position de départ de la plus grande correspondance entre Q et un élément de la collection se terminant à la position k. Soient f_k et g_k les extrémités de tous les indices qui correspondent à cette égalité. C'est-à-dire que pour chaque élément y_i dans la base de données avec i compris entre f_k et g_k, on a une égalité entre Q et y_i entre e_k et k mais pas en $e_k - 1$. La mise à jour de ces valeurs utilisent les tableaux pré-calculés et la requête du client.

Transformer l'algorithme pour l'utiliser de manière privée n'est pas intuitif. En effet, l'algorithme a besoin d'utiliser des tableaux pré-calculés que le client ne devrait pas

connaître puisqu'ils dépendent de la base de données ainsi que de la requête du client que le serveur ne devrait pas connaître. En utilisant des échanges entre le client et le serveur on pourrait toutefois imaginer une solution interactive à base de transferts privés.

4.2.2 De manière privée

L'algorithme PBWT a été adapté de façon à prendre en compte les problématiques de confidentialité des données d'entrée et de la base de données, cet algorithme privé s'appelle PBWT-sec [70]. Son but est d'envoyer une requête de taille l et une position au serveur pour récupérer le préfixe maximal commun entre la requête et toutes les entrées de la base de données à partir de cette position. Le serveur ne doit pas apprendre ce que contient la base de données du serveur et le client ne doit récupérer que la taille du préfixe maximal. Pour cela PBWT-sec utilise l'algorithme de PBWT permettant de passer de k à $k+1$ mais de manière privée. Il est donc nécessaire que le client puisse accéder à des valeurs pré-calculées par le serveur sans les apprendre et que le serveur mette à jour des valeurs en fonction de la requête du client. Pour cela PBWT-sec utilise des transferts équivoques (OT *Oblivious Transfer*) (voir Section 1.3.2). Pour que le serveur mette à jour une valeur le client peut lui faire sélectionner cette valeur (en fonction de sa requête) de manière privée grâce à un OT puis le serveur peut réaliser les opérations nécessaires à la mise à jour de façon homomorphe.

Cet algorithme possède toutefois d'importantes limites :

— Chaque mise à jour nécessite un OT, l'algorithme est donc très interactif.
— Il ne s'applique qu'à une position donnée, il est donc nécéssaire de le lancer k fois pour l'appliquer à l'ensemble du génome individuel.
— Son efficacité dépend fortement de la taille des requêtes, et ne passe pas à l'échelle.

4.3 Une solution privée au calcul d'ensembles maximaux basée sur le chiffrement homomorphe

Dans cette section, nous présentons notre contribution au calcul d'ensembles maximaux privés. Cette solution repose sur un calcul complètement homomorphe des ensembles communs entre la requête et la base de données et utilise comme brique de base la librairie TFHE présentée dans la sous-section suivante.

4.3.1 TFHE

TFHE est une librairie qui implémente un schéma de chiffrement totalement homomorphe. La librairie est également basée sur le problème LWE (cf. Section 1.2.3). Un problème important des schémas basés sur LWE concerne l'évolution du bruit. En effet, les opérations effectuées sur les chiffrés (additions, multiplications) font grandir le bruit qu'ils contiennent. Si le bruit, ou plus exactement sa taille, n'est pas contrôlé, il peut grandir jusqu'à écraser le message clair contenu dans le chiffré, tel que présenté en section 1.2.4.A cette fin, existe une opération de réinitialisation du bruit nommée *bootstraping*. Cette opération est coûteuse dans les schémas présentés jusqu'à maintenant. Si l'on est capable de réinitialiser le bruit à sa valeur initiale, il est alors possible d'effectuer autant d'opérations que l'on souhaite sur le chiffré. TFHE permet d'évaluer un circuit booléen arbitraire, composé de portes binaires qui opérent sur des données chiffrées. Le principe de THFE repose donc sur la réinitialisation du bruit après chaque porte binaire. La librairie offre une bibliothèque d'une dizaine portes binaires qui permettent donc de réaliser un circuit arbitraire. Chaque opération coute environ $13ms$.

Les paramètres varient selon les opérations, le degré du polynôme utilisé varie entre 500 et 2048. Le modulo dans l'espace des clairs est 2, celui dans l'espace des chiffrés est de taille 16 à 128. Nous avons utilisé les paramètres par défaut de la librairie qui offrent 152 bits de sécurité.

Nous avons créé une librairie au dessus de TFHE qui offre une API transparente pour des portes de plus haut niveau (comparaisons, recherche de maximum, addition sur plusieurs bits, etc). La librairie a été créée pour fonctionner en parallélisant les opérations et permettre à l'utilisateur d'avoir un contrôle sur la gestion de la mémoire.

4.3.2 Notre algorithme

Nous avons imaginé deux algorithmes permettant de calculer les ensembles maximaux de manière privée. Le premier algorithme a une complexité en $O(M.N.\log(N))$ avec M le nombre de génomes dans la base de données et N leur taille, c'est-à-dire le nombre de substitutions considérées. Le second algorithme est une optimisation du premier et permet de réduire la composante N de la complexité. Sa complexité est similaire mais sur un N' réduit (de l'ordre de $N' = N/50$ en général).

Pour chacun des algorithmes, le client envoie une requête R chiffrée avec TFHE au serveur qui effectue le calcul de la réponse L et la renvoie au client. Cette réponse contient le vecteur des tailles des composantes communes entre R et la base de données qui sont plus grandes que le seuil t et correspondent à un ensemble maximal.

L'idée principale de ces algorithmes est de calculer l'égalité bit à bit entre la requête et chaque entrée de la base de données dans une matrice d'égalité I. Ensuite on construit la matrice J qui contient la taille des séquences de 1 de la matrice I. Par exemple, si une des lignes de la matrice d'égalité I est 111011, dans J, le vecteur correspondant sera 321021 puisqu'à la première position il y a une suite de 3 1, ainsi de suite. Ce calcul est répété pour chaque entrée de la base de données, i.e. pour chaque ligne des matrices I et J. Après cela, le serveur cherche la valeur maximale de chaque colonne de J, on obtient ainsi le vecteur L. Le serveur vérifie si la taille de chaque ensemble est plus grande que le seuil et si l'ensemble ne peut pas être étendu. La valeur est conservée dans ce cas et mise à 0 sinon. Le vecteur L est enfin randomisé, afin que l'on ne puisse apprendre la position de chaque ensemble maximal, et cela constitue la réponse qui sera renvoyée au client.

L'algorithme que nous venons de décire peut se formaliser comme suit.

Algorithme 7 : Fonction Ensemble Maximal Privé : calcul de la matrice d'égalité I

1 Fonction Ensemble Maximal Privé (R, DB, t);
  Entrée : Requête $R = \{r_i\}$ avec $i \in [1, N]$ et $r_i = \{0, 1\}$
  Entrée : Base de données $DB = \{G^j\}$ avec $j \in [1, M]$, $G^j = \{g_i^j\}$ et $g_i^j = \{0, 1\}$
  Entrée : Seuil $t \in [1, N]$
  Sortie : Résultat $L = \{L_i\}$ avec $i \in [1, N]$ et $L_i \in [t, N - t]$
2 **for** $j \in [1, M]$ **do**
3 $I^j = \neg G^j \oplus R$ // I^j est l'égalité bit à bit entre G^j et R

Dans la première partie de l'algorithme, le serveur reçoit la requête R envoyée par le client. Il calcule la matrice d'égalité I entre R et les M entrées de la base G. La requête a la même taille qu'un élément de la base de données qui est de taille N. Puisque nous utilisons TFHE, chaque bit est représenté par un chiffré. La requête correspond donc à N chiffrés TFHE. En réalité, nous envoyons $N * \lceil log(N) \rceil$ chiffrés TFHE, puisque dans la suite de l'algorithme nous voulons représenter des entiers entre 0 et $N - 1$. Le résultat I que l'on obtient est une matrice M lignes, N colonnes de chiffrés TFHE.

Dans la seconde partie de l'algorithme le serveur calcule la matrice J des tailles de séquences égales dans I. La matrice J contient des nombres entiers qui correspondent au nombre de 1 consécutifs dans la matrice I à partir de la position courante. Pour chaque ligne de la matrice I, on se place à la dernière position et on garde la valeur de I dans J. Ensuite on parcourt cette ligne en arrière, de la fin vers le début. Soit i la position actuelle. Si l'on fait $(I_i + J_{i+1}) * I_i$, dans le cas où I_i vaut 0 on met la valeur à 0, si I_i vaut 1 on va incrémenter la valeur présente dans J_{i+1} de 1. J_{i+1} correspond au nombre

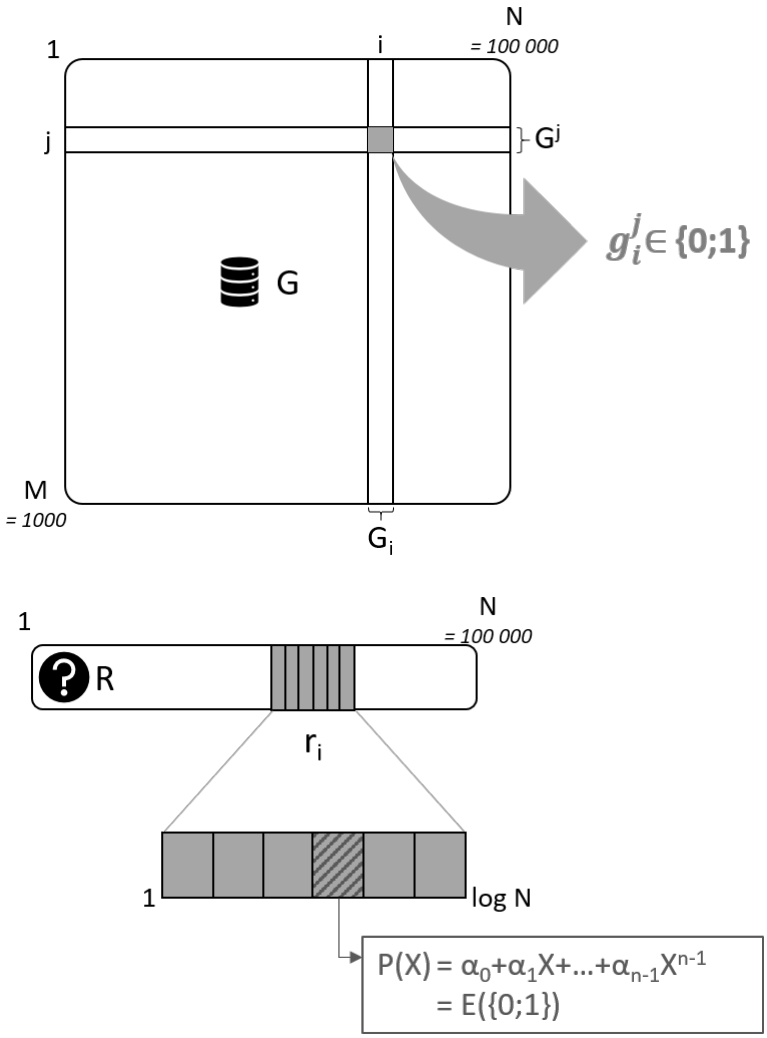

FIGURE 4.2 Description des notations avec un exemple de paramétrage classique. Le client envoie une requête R de taille $N * log(N)$ chiffrés TFHE, $log(N)$ chiffrés sont nécessaires pour chiffrer un entier qui peut valoir de 0 à N. La base de données G a M lignes et N colonnes.

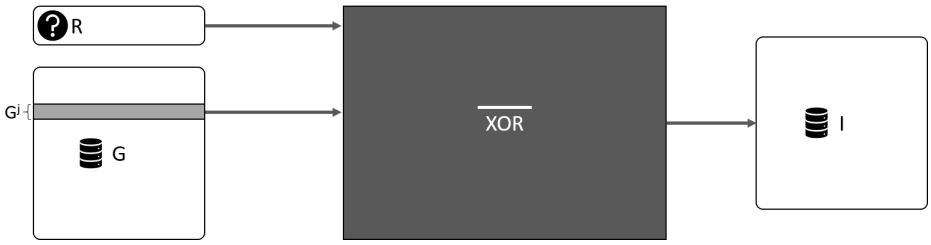

FIGURE 4.3 Le serveur reçoit la requête R du client, il va effectuer pour chaque ligne un NON XOR (qui est l'égalité bit à bit) entre la requête et chaque entrée de la base de données G^j. La nouvelle base de données obtenues est de la même taille que G et chaque coordonnées est représentée par $log(N)$ chiffrés TFHE, dont un seul est utilisé pour stocker l'égalité bit à bit entre $g_i^j et R_i$

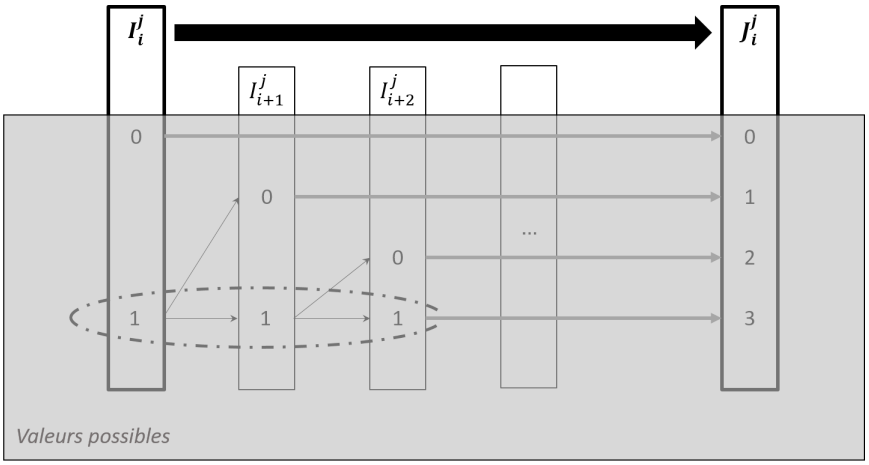

FIGURE 4.4 Le serveur transforme la matrice I en une matrice J. La matrice I contient l'égalité bit à bit entre la base de données initiale G et la requête R. Une coordonnée de la matrice J représente le nombre de 1 successifs à partir de la position i. Ce calcul est fait de droite à gauche, i.e. des positions N à 1, en calculant $I_i^j * (I_i^j + J_{i+1}^j)$.

Algorithme 8 : Fonction Ensemble Maximal Privé : calcul de la matrice de tailles d'égalité J

```
1  for i ∈ [N, 1] do
2      // Pour chaque position dans l'ordre inverse
3      for j ∈ [1, M] do
4          // Pour chaque élément
5          if i == N then
6              // La dernière position la taille est soit 0 soit 1
7              J_N^j = I_N^j
8          else
9              // Dans les autres positions la taille est incrémentée de 1 s'il y a un 1
                 et est mise à 0 sinon
10             J_i^j = (J_{i+1}^j + I_i^j) * I_i^j
```

de 1 consécutifs à partir de la position $i+1$. J_i correspond donc bien à ce nombre à partir de la position i. Dans TFHE nous travaillons avec des coefficients modulo 2 dans l'espace des clairs. C'est pour cela que la représentation d'un nombre correspond à $\lceil log(N) \rceil$ chiffrés. L'addition est une addition sur $\lceil log(N) \rceil$ bits.

Nous remarquons que pour cette opération, nous avons une profondeur multiplicative de taille N. Une telle profondeur multiplicative peut poser de sérieux problèmes de croissance du bruit et nécessite une relinéarisation fréquente, c'est essentiellement cette raison qui nous a poussé à choisir TFHE comme librairie sous-jacente.

Algorithme 9 : Fonction Ensemble Maximal Privé : calcul du vecteur des préfixes maximaux L

```
1  ∀i ∈ [1, N] ∀j ∈ [1, M] L_i = (MAX[J_i^j]) //C'est le préfixe maximal à partir de la
     position i
```

Nous allons chercher la valeur maximale sur chaque colonne de la matrice J. Elle correspond au préfixe maximal commun à cette position. A la fin de cette boucle, nous avons un vecteur L de $\lceil log(N) \rceil$ chiffrés TFHE par coordonnée.

La dernière partie de l'algorithme supprime les ensembles qui ne sont pas maximaux. Il est inutile de vérifier si l'ensemble peut être étendu par la droite puisque c'est un préfixe maximal (qui n'est pas extensible par la droite par définition). S'il peut être étendu c'est uniquement par la gauche. Si l'ensemble peut être étendu par la gauche alors la coordonnée précédente est supérieure de 1 à la coordonnée courante. Soit la position n, si $L_n + 1$ est égal à L_{n-1} alors L_n n'est pas un ensemble maximal et sa valeur est mise

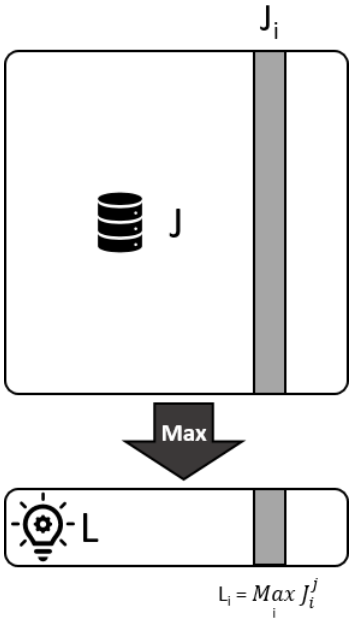

FIGURE 4.5 Le serveur recherche le plus grand préfixe à chaque position. Pour cela il doit récupérer, dans un vecteur, le maximum sur chaque colonne de la matrice J. La coordonnée L_i contient la taille du plus grand préfixe commun entre la requête du client et l'ensemble des entrées de la base de données à partir de la position i

Algorithme 10 : Fonction Ensemble Maximal Privé : filtrage des ensembles non-maximaux ou de longueur inférieure au seuil

// Si l'ensemble peut être étendu par la gauche, on le supprime puisqu'il est inclus dans l'ensemble qui démarre à une position antérieure
$\forall i \in [N-t, 1]\ L_i = L_i - L_i * (L_i + 1 == L_{i-1})$
// Si l'ensemble est plus petit que le seuil on le supprime.
$\forall i \in [N-t, 1]\ L_i = L_i * (L_i > t - 1)$
// On envoie L_i au client, potentiellement en y appliquant une permutation aléatoire pour cacher la position de départ des ensembles.

à 0. Pour finir, nous vérifions que l'ensemble est plus grand que le seuil. Si nécessaire, nous effectuons une permutation aléatoire au sein des coordonnées de L et renvoyons le vecteur au client.

La complexité de l'algorithme est de $O(M.N.\log N)$ pour le calcul de J et du max.

Cette complexité peut paraître élevée, en particulier en regard de N qui est le facteur dominant en pratique. C'est pourquoi nous proposons un second algorithme qui troque un certain niveau de précision (paramétrable) contre une plus grande efficacité. L'idée principale de cet algorithme est de réduire la taille de la base de données pour la calcul de J et du max afin que l'opération la plus coûteuse soit le calcul de I (l'égalité bit à bit) dont la complexité est $O(M.N)$.

Dans cet algorithme, le but est de réduire la taille N et de travailler sur des blocs plutôt que sur des éléments individuels. Un bloc représente *taille_bloc* positions consécutives. La valeur d'un bloc est binaire, 1 signifie que chaque élément de l'entrée vaut 1, à l'inverse un 0 dans le bloc indique que tous les éléments de l'entrée n'étaient pas à 1.

Algorithme 11 : Fonction Ensemble Maximal Privé par bloc

for $j \in [1, M]$ **do**
  $I^j = \neg G^j \oplus R$ // I^j est légalité bit à bit entre G^j et R
for $j \in [1, M]$ **do**
  // Pour chaque entrée
  for $i \in [0, N/taille\_bloc - 1]$ **do**
    // Pour chaque position
    $Iblock_i^j == 1$
    for $l \in [1, taille\_bloc]$ **do**
      $Iblock_i^j = I^j_{l+i*taille\_bloc} \wedge Iblock_i^j$
Ensuite on applique le premier algorithme sur $Iblock_i^j$ au lieu de l'appliquer sur I_i^j

La complexité totale de cette version de l'algorithme, avec l'appel du premier algorithme sur la base de données réduite est $O(M.N + M.N/block\_size. \log(N/taille\_bloc))$. Cet algorithme renvoie un résultat approché qui peut contenir trois types d'erreurs différentes :

Erreur sur la longueur de l'ensemble. Le serveur ne renvoie pas la longueur exacte des ensembles maximaux mais une valeur approchée avec une erreur de plus ou moins la taille d'un bloc : $Error\_max = taille\_bloc - 1$. Par exemple avec une taille de bloc de 50, si le client reçoit 20 (c'est à dire 20 blocs de taille 50), il convertit cela en 1049 et la taille réelle de l'ensemble se trouve entre 1000 et 1098.

Bloc trop petit. Le serveur doit envoyer tous les ensembles maximaux de taille supérieure au seuil t. A cause de l'approximation de la longueur (décrite ci-dessus), le serveur peut aussi transmettre certains ensembles qui sont plus petits que t. La taille minimale des ensembles renvoyée est $t - taille\_bloc$. Par exemple, avec une taille de bloc de 50, un seuil de 1000, si le client reçoit 19, il va convertir cela en 999 et accepter l'ensemble puisque la taille réelle est entre 950 et 1048, ce qui peut être inférieur à 1000. Nous avons décidé d'accepter tous les ensembles dont la taille est potentiellement supérieure à t pour privilégier les faux-positifs aux faux-négatifs, i.e. la réponse contient tous les ensembles de taille supérieure au seuil plutôt qu'aucun ensemble de taille inférieure au seuil.

Bloc raté. L'algorithme peut rater un ensemble maximal dans un cas spécifique et peu fréquent. Dans l'algorithme de base, si un préfixe maximal ne peut pas être étendu par la gauche il est maximal. Nous supposons donc la même chose pour l'algorithme par bloc. Cependant puisque nous travaillons par bloc cette supposition n'est plus vraie, il faudrait également vérifier que l'ensemble n'est pas maximal par la droite. Un 0 signifie que tous les *taille_bloc* éléments ne valent pas 1. Par exemple avec $taille\_bloc = 2$ et 2 éléments dans la base de données : e_1=11 11 00 et e_2=01 11 10. L'algorithme réduit la base de données en e'_1=1 1 0 et e'_2=0 1 0, L vaut $2, 1, 0$ et nous mettons le 1 à 0 car on peut étendre par la gauche. Dans ce cas nous avons oublié l'ensemble maximal dans e_2

Les deux premiers cas d'erreur sont maîtrisés et ne portent préjudice que sur la précision du résultat. En effet, l'erreur sur la taille de la séquence trouvée est due au fait que l'algorithme retourne le nombre de blocs commun, mais la séquence est trouvée. L'erreur sur la longueur est d'au plus $taille\_bloc - 1$. L'algorithme peut retourner une séquence plus petite que le seuil car nous avons fait le choix de retourner toutes les séquences d'au moins la taille du seuil. Cependant les blocs renvoyés sont également

de longues séquences communes puisqu'ils sont au moins de taille $seuil - taille\_bloc$. Le dernier cas d'erreur, le bloc raté, est plus problématique puisque cela rend notre algorithme inexact. Il arrive lorsque 2 longues séquences communes avec deux entrées différentes dans la base de données se chevauchent. C'est à dire qu'une commence plus tôt et finit plus tôt que l'autre. Dans tous les jeux de données sur lesquels nous avons testé notre algorithme, ce cas d'erreur n'est jamais survenu, avec un bon ordonnancement des substitutions ce cas semble rare en génomique.

Dans les deux algorithmes le client n'apprend que la taille (approchée dans le second cas) des ensembles maximaux, et le serveur n'apprend rien.

4.3.3 Implémentation, Résultats et Analyse

Dans cette section, nous discutons des performances des algorithmes présentés ci-dessus. Toutes les boucles de l'algorithme sont calculées en parallèle, ce qui permet un gain proportionnel au nombre de coeurs. De plus certaines portes sont également parallélisées, comme le calcul des max, par exemple. Tous les tests ont été effectués sur une machine virtuelle Linux, utilisant 4 Intel Xeon CPU E5-2690 v4 2.60GHz x86_64 avec le jeu d'instruction avx2 et 16GO de RAM.

La première courbe 4.6 montre l'évolution du temps sans parallélisation de l'algorithme exact en fonction de M. La valeur de N est fixée à 10. Conformément à nos attentes, l'algorithme est linéaire par rapport à M. Cet algorithme peut être presque entièrement parallélisé, ainsi en utilisant les 4 coeurs, il est possible de gagner un facteur 4 (3.85 en pratique dans les tests).

La seconde courbe 4.7 montre l'évolution du temps sans parallélisation de l'algorithme exact en fonction de N. La valeur de M est fixée à 10. Conformément à nos attentes, la complexité de l'algorithme augmente en $N * log(N)$. Comme on le voit sur ces courbes, cet algorithme ne passe pas à l'échelle avec des données génomiques de taille raisonnable (environ 70 jours pour une base de données standard avec $M = 100 et N = 100000$).

Cependant, avec cette solution, le résultat renvoyé est précis et exact. Le serveur n'apprend pas la requête grâce au chiffrement homomorphe. Le client n'apprend que la taille des ensembles maximaux. Il n'apprend pas la position de ces derniers grâce à la permutation des L_i. Les résultats de l'algorithme exact illustrent bien la raison qui nous a poussé à développer l'algorithme approché : essayer d'avoir un algorithme qui passe mieux à l'échelle, en particulier par rapport à N.

Analysons maintenant les performances de l'algorithme approché. La troisième courbe 4.8 montre l'évolution du temps sans parallélisation de l'algorithme par bloc en fonction de N. La valeur de M est fixée à 10 et la taille des blocs à 50. On observe un

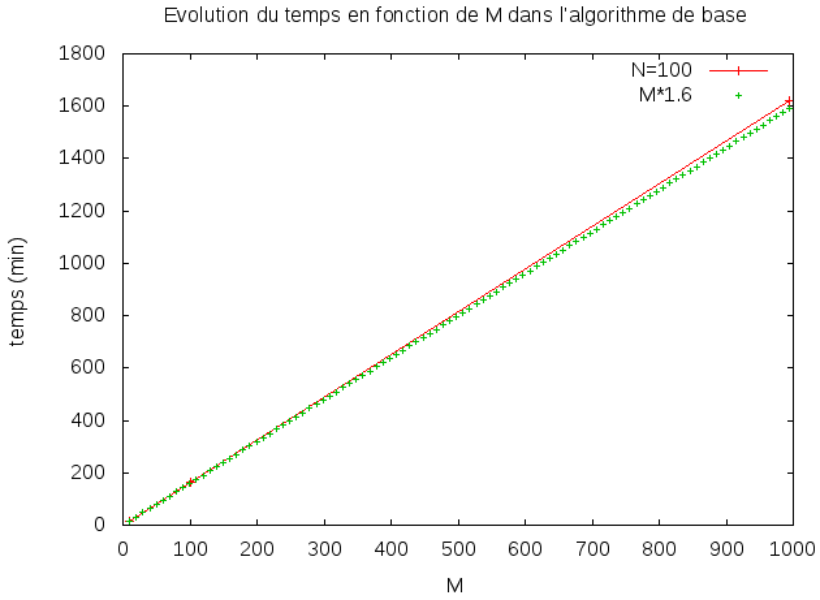

FIGURE 4.6 L'évolution du temps sans parallélisation de l'algorithme exact en fonction de M avec $N = 10$.

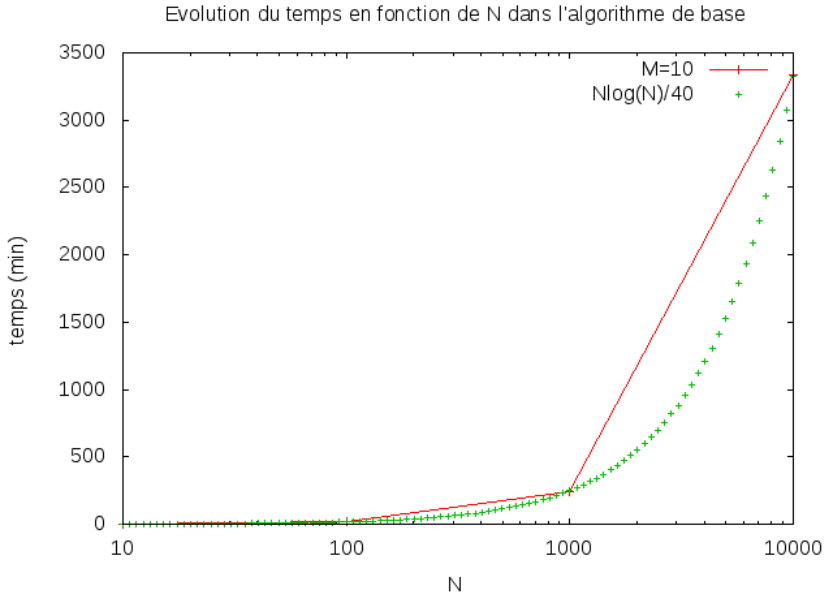

FIGURE 4.7 L'évolution du temps sans parallélisation de l'algorithme exact en fonction de N avec $M = 10$.

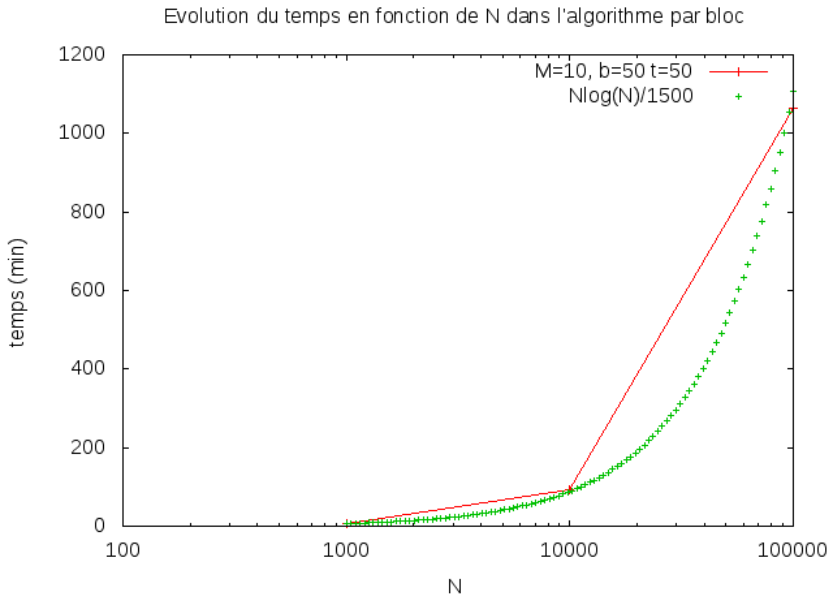

FIGURE 4.8 L'évolution du temps sans parallélisation de l'algorithme par bloc en fonction de N avec $M = 10$ et la taille de bloc 50.

gain de temps significatif, pour $M = 100$ et $N = 100000$, on aurait le résultat en 3.6 jours (contre 70 avec la solution précédente)

La quatrième courbe 4.9 montre l'évolution du temps sans parallélisation de l'algorithme par bloc en fonction de M. La valeur de N est fixée à 1000 et la taille des blocs à 50. Le temps est linéaire en M comme attendu.

La cinquième courbe 4.10 montre la distribution du temps dans l'algorithme par bloc en fonction de la taille du bloc. On constate que la première partie de l'algorithme a un temps indépendant de la taille de bloc. Cette première partie correspond au premier NON XOR et au calcul des blocs. La seconde partie, le calcul de J, des max et des inclusions dépend de la taille du bloc. Une taille de bloc de 50 permet de faire en sorte que la première partie dure la majorité du temps, nous ne pouvons pas gagner de temps sur cette partie en utilisant la stratégie par blocs.

4.3.4 IDASH

Nous avons présenté cette solution lors du challenge IDASH en 2018. En 2018, une des taches du challenge IDASH portait sur la réalisation d'un algorithme de calcul de la taille des ensembles maximaux de manière privée et non-interactive. La soumission de notre

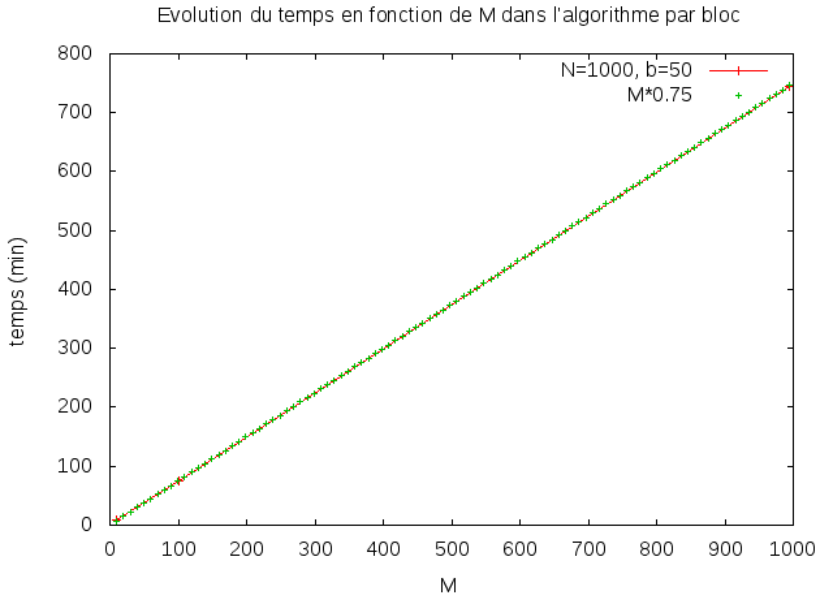

FIGURE 4.9 L'évolution du temps sans parallélisation de l'algorithme par bloc en fonction de M avec $N = 1000$ et la taille des blocs 50.

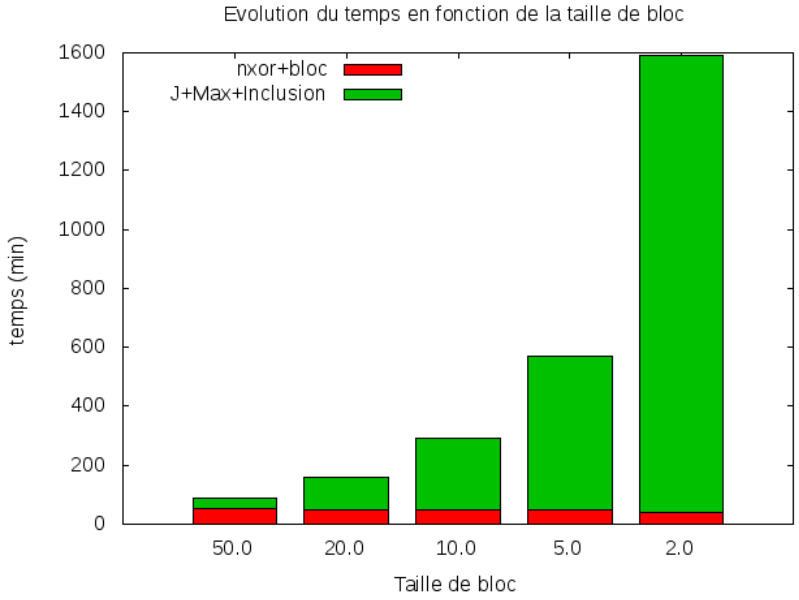

FIGURE 4.10 La distribution du temps dans l'algorithme par bloc en fonction de la taille du bloc.

algorithme nous a permis de finir premier ex-aequo avec une équipe de Microsoft. La solution de Microsoft était plus efficace mais moins précise, à l'opposé de notre solution, plutôt gourmande en calcul mais qui donne une solution exacte.

Le challenge se place exactement dans avec ces condition :

1. Modèle avec un client et un serveur
2. Un seul aller-retour dans les communications
3. Le client ne doit rien apprendre sur le base de données à part la réponse
4. Le serveur ne doit pas apprendre d'informations sur la requête R

Les jeux de tests utilisaient $M = 1000$ et N variant de 100 à 100 000. Avec $M = 1000$ et $N = 100$ notre solution prend 26 heures mais a une précision de 23/26 (23 ensembles renvoyés sur les 26 attendus). Nous nous serions attendus à une précision absolue, de 26/26, mais malheureusement les jeux de tests effectués par les organisateurs n'ont pas été rendu publics, malgré nos multiples requêtes, et il nous est impossible d'analyser plus avant la situation. Nous supposons que la raison de ces imprécisions est la suivante : lorsque 2 ensembles maximaux sont exactement les mêmes nous n'en renvoyons qu'un, i.e. nous avons interprété maximal au sens strict. Nous n'avons pas trouvé de moyen de renvoyer les ensembles maximaux égaux sans réduire considérablement nos performances.

La solution de Microsoft prend, quant à elle, 7 secondes mais a une précision beaucoup plus faible (12/26). Cette précision décroit quand N augmente. Avec $M = 1000$ et $N = 1000$ leur solution prend 50 secondes mais a une précision de 4/50. Les organisateurs n'ont pas testé notre solution avec ce jeu de paramètres (qui aurait pris environ 10 jours).

La solution de Microsoft utilise un schéma basé sur GSW [71]. Contrairement aux schémas comme FV ou BGV (voir section 2.1), lorsque l'on multiplie un chiffré fortement bruité, par un chiffré moins bruité, le bruit dans le chiffré fortement bruité augmente faiblement.

Il est probable qu'une solution hybride entre ces deux solutions aurait donné de bons résultats.

4.4 Conclusions et Perspectives

Nous avons présenté une solution non-interactive permettant de calculer l'ensemble des ensembles maximaux entre une requête et une base de données de manière privée. Avec cette solution, le client n'apprend rien d'autre que la longueur et le nombre d'ensembles maximaux. Le serveur n'apprend rien. Cet algorithme a besoin d'une profondeur multiplicative importante d'où l'utilisation d'un schéma totalement homomorphe. Dans sa

première version, l'algorithme a une précision optimale mais est lent. Nous avons présenté une seconde version moins précise mais plus rapide. Malgré de meilleures performances, le temps de calcul nécessaire à la solution approchée peut paraitre élevé. Voici quelques pistes qui pourraient permettre d'améliorer encore ces performances. Nous avons utilisé TFHE sans batching, le batching permettrait d'utiliser un seul chiffré pour représenter plusieurs données (voir Section 1.2.6). Il est possible de réaliser du batching sur certaines opérations (sur le calcul des max ou les additions sur plusieurs bits par exemple). Cependant, cela n'est pas encore implémenté pour THFE, on estime qu'on pourrait avoir un gain de temps d'un facteur environ 3. TFHE a amélioré le temps d'une réinitialisation du bruit en passant de $690ms$ à $13ms$. On peut supposer que les schéma totalement homomorphes continueront à évoluer ce qui permettrait à notre algorithme d'atteindre un temps raisonnable. Le temps d'une opération dans notre programme correspond au temps d'une réinitialisation.

Pour réduire la base de données, on pourrait éliminer des lignes ou colonnes qui ne seraient pas impliquées dans un ensemble plus grand que le seuil. Trouver ces lignes ou colonnes est facile, le problème est d'utiliser cela tout en conservant la propriété de non divulgation des données. Il est possible de faire cela grâce à une permutation privée, mais la complexité de l'opération est plus importante que celle de notre algorithme.

Nous avons essayé d'utiliser PBWT, cependant l'algorithme nécessite trop d'interactions. Nous n'avons pas trouvé de moyen de l'utiliser dans une version non-interactive et privée.

Nous avons exploré des pistes utilisant des fonctions de hachage. L'idée est de hacher tous les éléments de la taille du seuil t et d'avoir une base de données de ces hachés. En réutilisant les solutions à base de PIR présentées dans le chapitre précédent, pour réaliser N requêtes afin de vérifier si la base de données possède ou non cet ensemble de taille t. En effet, nous avons, dans le chapitre précédent, proposé une solution qui permet de vérifier la présence d'un élément que nous connaissons parfaitement dans une collection, de manière privée. Le principal problème de cette solution est que nous récupérons tous les ensembles plus grands que le seuil même s'ils ne sont pas maximaux. Cela donne trop d'informations au client sur la base de données.

Comparée à PBWT-sec, notre solution permet de calculer l'ensemble des ensembles-maximaux et offre donc une protection de vie privée optimale. PBWT-sec, qui s'appliquerait à l'ensemble de la base de données, sur des ensembles longs (cacher parmi toutes les positions, avec une longueur de requête de plus de 1000) est plus coûteux que la version de base de notre algorithme. Là où notre algorithme prendrait 70 jours, PBWT-sec prendrait environ 220 jours.

Malgré tout, les solutions que nous avons présentées dans ce chapitre ne semblent pas assez efficaces pour passer à l'échelle, en particulier au regard de la taille des bases de données génomiques réelles. Cependant, nous pensons que les futures évolutions dans le domaine du chiffrement totalement homomorphe permettront de fortement accélérer la solution qui a une précision parfaite dans sa première version.

Chapitre 5

Détection de fraude privée lors d'un paiement

Beaucoup d'entreprises se sont tournées vers le commerce électronique afin d'améliorer leur productivité et leur visibilité dans le but de vendre leurs produits ou services. Ces systèmes de paiement électronique sont utilisés par des utilisateurs légitimes mais également par certains autres frauduleux. La fraude est un crime dont le but est de s'approprier de l'argent par des moyens non légaux. *Inter-net Crime Complaint Centre* (IC3) est une agence qui regroupe le FBI (*Federal Bureau of Investigation*), le NW3C (*National White Collar Crime Center*) et le BJA (*Bureau of Justice Assistance*) et qui fournit aux plaignants de cybercrimes un mécanisme de déclaration pratique et simple pour alerter les autorités. En 2014, l'IC3 a reçu 269 422 plaintes qui ont entraîné une perte de 800 millions de dollars. Le rapport de 2019 [1] montre l'évolution entre 2015 et 2019 du nombre de plaintes et de pertes entraînées : 467 361 plaintes pour une perte de 3,5 milliards en dollars en 2019. L'importance des pertes dues aux fraudes a entraîné la recherche de techniques pour la combattre [72]. Le principe des mécanismes de prévention de fraude est de protéger les systèmes en empêchant la fraude de se produire. Cependant, ces mécanismes ne sont pas toujours très performants [73] et ne permettent pas seuls de prévenir au mieux la fraude. Les systèmes de détection de fraude permettent d'améliorer la sécurité contre la fraude et notamment de rapporter la fraude à l'administrateur du système. Les fraudes relatives aux cartes bleues peuvent être classées en deux catégories [74] :

— La fraude hors ligne, qui consiste à voler matériellement la carte bleue de la victime pour l'utiliser. Ce type de fraude est peu commun puisque, dans la majeure partie

1. https ://pdf.ic3.gov/2019_IC3Report.pdf

des cas, lorsque la banque a été mise au courant du vol, toutes les transactions provenant de la carte sont bloquées.

— Le fraude en ligne, qui consiste à récupérer les informations liées à la carte bleue de la victime et les utiliser. Ce type de fraude est plus commun. Les informations peuvent être récupérées via une faille lors d'un paiement électronique, une campagne de phishing ou encore une génération aléatoire de carte.

La plupart des techniques pour détecter ces fraudes se basent sur la détection d'anomalies. Le profil du client est dressé en fonction de son comportement. De cette façon, toute transaction qui serait incohérente avec le profil dressé sera considérée comme suspicieuse [75]. Les systèmes de détection de fraude utilisent en général un arbre de décision ou une décision basée sur des règles [76].

Dans ce chapitre, nous présentons un algorithme de détection de fraude privé qui protège les données liées à la détection de la fraude mais également les données du client.

La prise de conscience dans le domaine de la protection de vie privée est très récente, et l'exploration de pistes pour réaliser une détection de fraude de manière privée est presque inexistante. En 2018 Canillas et al. [77] utilisent un arbre de décision qui conserve le caractère privé des données. L'approche est très intéressante mais coûteuse puisqu'elle nécessite environ $1s$ pour réaliser la détection de la fraude ainsi que 2.5Go de bande passante sur le réseau.

5.1 Présentation du problème

5.1.1 Détection privée

Dans la solution présentée, nous supposons que le serveur qui s'occupe de la détection de fraude possède un ensemble de règles concernant les données du client de la forme suivante : si une donnée du client x est supérieure à un seuil t alors le score de fraude du client est incrémenté de α. Si le score de fraude est supérieur à un seuil final T, une alerte est levée par le serveur car le paiement est suspicieux. Nous ne nous intéressons pas à la façon dont ont été générés les seuils t et T ni les pondérations α. Ces données sont généralement obtenues en dressant le profil du client puis par l'utilisation d'algorithmes basés sur les réseaux de neurones [76]. Le nombre de caractéristiques du client est noté N. Ces données peuvent représenter par exemple le montant ou le lieu du paiement, l'historique du client, etc. La détection de la fraude s'effectue lors du paiement, nous supposons donc que les connexions nécessaires sont déjà établies pour permettre le paiement et donc la détection de la fraude. La durée nécessaire pour réaliser un paiement ne dépasse pas quelques secondes, nous souhaitons donc proposer une solution ne dépassant

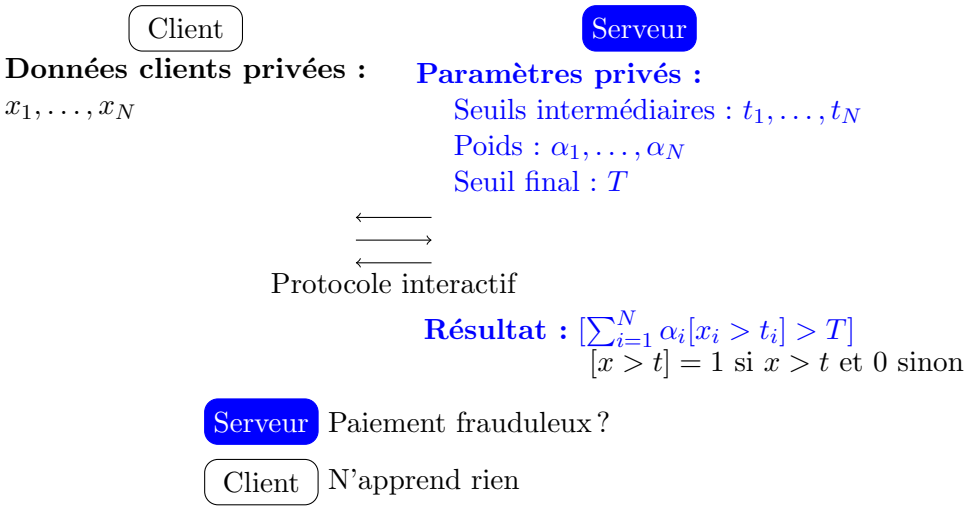

FIGURE 5.1 Description des paramètres et de leur sécurité. Le client possède ses données de paiement. Le serveur possède des seuils intermédiaires qui correspondent à chaque donnée, des poids associés à ces seuils et un seuil final. Il souhaite vérifier sans apprendre les données du client si le paiement est frauduleux, pour cela il doit vérifier pour chaque donnée de paiement si elle est plus grande que le seuil intermédiaire, puis faire la somme pondérée par les poids de ces tests d'inégalité, et enfin vérifier si cette somme est plus grande que le seuil final. Le client ne doit pas apprendre les données du serveur.

pas quelques centaines de millisecondes. Il est préférable, pour respecter cette contrainte temporelle, d'avoir une solution peu interactive.

Le client pourrait être un terminal de paiement ou un service de paiement en ligne. La fraude la plus courante étant celle en ligne, nous baserons les résultats sur les capacités d'un ordinateur. Lors de la détection de fraude privée, le client ne doit apprendre ni les seuils intermédiaires t, ni les pondérations α ni le seuil final T.

Le serveur ne doit apprendre ni les données x du client, ni le résultat final du score de fraude du client, ni les résultats intermédiaires $[x > t]$. $[x > t]$ vaut 1 si x est plus grand que t et 0 sinon.

5.2 Etat de l'art

La solution pour permettre le paiement privé tel que décrit ci-dessus passera par des comparaisons privées. A priori, le moyen le plus rapide [78] pour réaliser les comparaisons privées est l'utilisation des *garbled circuits* [79].

La génération du circuit se fait conventionnellement comme un ensemble de portes logiques XOR et ET à deux entrées en minimisant le nombre de ET puisqu'il existe une

optimisation sur les portes XOR [80].

Prenons l'exemple d'une porte ET à deux entrées pour illustrer le fonctionnement d'un *garbled circuit*. Soit la table de vérité de la porte ET ayant pour entrée a et b et comme sortie c.

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Soit Alice qui réalise le *garbled circuit*. Alice attribue des labels qui sont générés comme des nonce pour chaque élément de la table de vérité. La taille du label représente le paramètre de sécurité. Un label est généré pour chaque valeur binaire. Voici la table de vérité des labels.

a	b	c
X_0^a	X_0^b	X_0^c
X_0^a	X_1^b	X_0^c
X_1^a	X_0^b	X_0^c
X_1^a	X_1^b	X_1^c

Puis Alice chiffre cette table pour obtenir la *garbled table*. La fonction de chiffrement est une fonction symétrique qui utilise 2 clés. Les clés sont les labels correspondants aux entrées, le label correspondant à la sortie est chiffré. Les entrées de la table subissent une permutation aléatoire. Cette table est envoyée à Bob.

Alice envoie à Bob le label correspondant à son entrée X_0^a ou X_1^a. Bob, pour obtenir son label, va réaliser un *1 out of 2 oblivious transfer* (voir Section 1.3.2). Bob peut ensuite déchiffrer une entrée dans la *garbled table* et envoyer le label correspondant à Alice qui peut retrouver la valeur booléenne.

Une inégalité privée avec des entrées sur 8 bits s'exécute en 40ms en utilisant un *garbled circuit*. Cependant dans le protocole de paiement privé, les résultats intermédiaires doivent être privés. Cette contrainte augmente la taille du circuit. En gardant le temps d'exécution dans l'ordre de la centaine de millisecondes, il ne serait possible que de réaliser 4 à 5 inégalités en utilisant un *garbled circuit*.

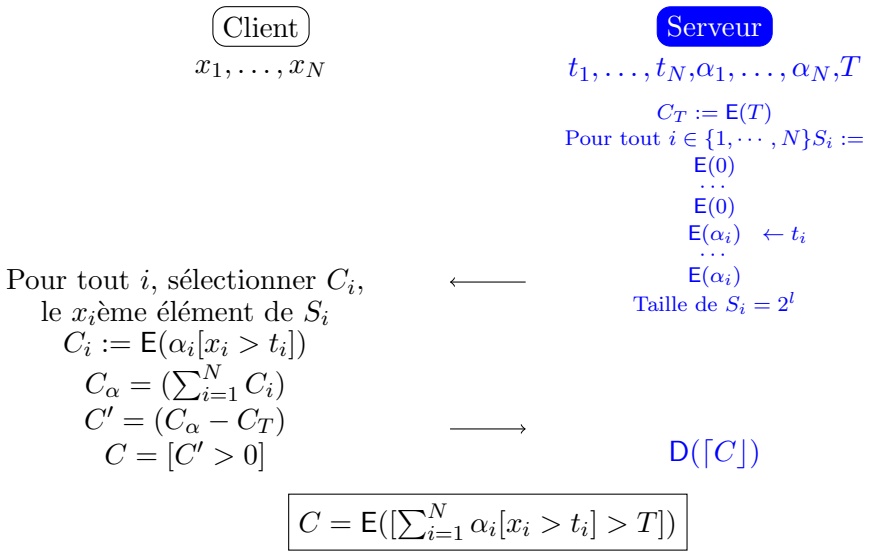

FIGURE 5.2 Protocole permettant de réaliser la détection de fraude de manière privée. Le serveur envoie, pour chaque donnée x du client de taille l, 2^l chiffrés homomorphes. Ces chiffrés sont des chiffrés de 0 pour les t premiers puis des chiffrés de α pour les suivants. Le client sélectionne le $x^{ème}$ chiffré qui correspond au test $[x > t]$ multiplié par α. Il somme l'ensemble des résultats obtenus, soustrait le seuil final envoyé par le serveur sous forme de chiffré homomorphe et teste si le résultat est positif ou négatif. Cette étape sera décrite dans l'algorithme 13. Le client envoie le résultat du test final qui est déchiffré par le serveur pour savoir s'il y a fraude.

5.3 Notre solution

5.3.1 KODA

Notre solution est décrite dans la figure 5.2. Soit l la taille d'une donnée x_i possédée par le client. Le serveur va utiliser une fonction de chiffrement homomorphe pour obtenir 2^l chiffrés qui représentent toutes les valeurs possibles de x_i. Le serveur génère t_i, le seuil intermédiaire, chiffrés de 0 puis $2^l - t_i$ chiffrés de α_i, le poids associé. Cet ensemble de chiffrés est noté S_i, les premiers chiffrés de cet ensemble sont ceux de 0. Cette opération est répétée pour les N données du client. Le serveur chiffre également, avec une fonction de chiffrement homomorphe T, le seuil final. Ces chiffrés sont envoyés au client. Pour chaque ensemble le client va sélectionner le $x_i^{ème}$ chiffré noté C_i. Le chiffré représente $\mathsf{E}(\alpha_i[x_i > t_i])$ où $[x_i > t_i]$ vaut 1 si x_i est plus grand que t_i et 0 sinon. En effet, le serveur a généré t_i chiffrés de 0, ainsi en sélectionnant le $x_i^{ème}$ chiffré, le client récupère le résultat de l'inégalité puisque si x_i est inférieur à t_i il sélectionne un chiffré de 0 et sinon un chiffré de α_i.

Le client somme tous les C_i, $C_\alpha = (\sum_{i=1}^{N} C_i)$ représente le score de fraude du client. La dernière étape de l'algorithme consiste à calculer $[C_\alpha > C_T]$ puis à renvoyer ce résultat au serveur. Si le score de fraude du client est supérieur au seuil final alors le serveur lèvera une fraude. Cette dernière étape est décrite dans l'algorithme 13.

5.3.2 Notre implémentation

L'implémentation de la solution a été réalisée sur SEALv2.3 (voir Section 2.1.2). Les paramètres ont été choisis par défaut en attribuant le degré du polynôme $n = 2048$. La taille d'un coefficient dans le domaine des chiffrés est de 54, dans le domaine des clairs de 16. 16 bits sont suffisants pour encoder les α et la somme des α. Un chiffré est donc de taille $54 \times 2048 = 108Kb$. Afin de réduire le coût réseau il est possible d'utiliser le *full batching* (voir Section 1.2.6). Ainsi chaque coefficient du chiffré sera utilisé pour chiffrer les éléments des ensembles S_i. Il est donc possible d'envoyer $\lceil 2^l/n \rceil$ chiffrés par ensemble et de réduire les coûts liés au réseau d'un facteur n environ. Le client plutôt que de sélectionner le $x_i^{ème}$ chiffré, sélectionnera le $x_i^{ème}$ coefficient. Pour cela il sélectionne le chiffré $\lfloor x_i/n \rfloor$ puis réalise une rotation du reste de la division euclidienne de x par $n : r$. Cette rotation permet de positionner le $x_i^{ème}$ coefficient sur le coefficient constant pour permettre l'addition $C_\alpha = (\sum_{i=1}^{N} C_i)$. La solution est détaillée dans l'algorithme 12.

Réalisation de $C = [C' > 0]$ (voir Figure 5.2) : Pour tester l'inégalité $[C' > 0]$ (C' est la somme pondérée par les α des tests d'inégalités intermédiaires à laquelle nous

Algorithme 12 : Utilisation du *full batching* dans l'envoi des chiffrés par le serveur

Input : t un seuil intermédiaire, α un poids associé, x la donnée du client, l la taille de x, n le degré d'un polynôme RLWE
Output : $\mathsf{E}(\alpha[x > t])$

begin
  //Côté serveur
  for $i \in 0..2^l - 1$ **do**
    if $i > t$ **then**
      $p_i = \alpha$
    else
      $p_i = 0$
  for $j \in 0..\lfloor 2^l/n \rfloor$ **do**
    $P_j = p_{j*n}X^0 + .. + p_{j*n+n-1}X^{n-1}$
    //Le serveur envoie $\mathsf{E}(P_j)$
  //Côté client
  Sélection du chiffré $\mathsf{E}(P_{\lfloor x/n \rfloor})$
  Soit r le reste de la division euclidienne de x par n
  $C_i := -X^{n-r} * \mathsf{E}(P_{\lfloor x/n \rfloor})$
  return $C_i := \mathsf{E}(\alpha[x > t])$;

avons enlevé le seuil final T) nous avons défini un nombre maximal noté T_{max} qui est la somme des α. Il n'est pas possible que le résultat contenu dans $C' + C_T$ dépasse T_{max}. L'algorithme pour tester l'inégalité est formalisé dans l'algorithme 13.

Pour réaliser l'algorithme 13, le client doit réaliser la boucle For $Ti \in T + 1..T_{max}$. Il est donc nécessaire pour le client de connaître la valeur de $T_{max} - T$. Nous supposons que l'information révélée au client par l'apprentissage de cette valeur est sans importance. Puisque le client ne connaît pas la pondération des seuils intermédiaires il n'est pas possible pour lui de calculer T_{max}.

5.3.3 Les améliorations possibles

La solution décrite ci-dessus présente plusieurs axes d'amélioration possibles. Il est possible de choisir les pondérations α comme une fonction de distribution F telle que $x \in [1; 2^l]$ et $F(x) \in \mathbb{N}$ avec $F(x) = \alpha_x$. Dans ce cas les seuils intermédiaires n'ont plus d'intérêt. Lors de la création des S_i dans la figure 5.2, chaque élément de l'ensemble sera associé à une pondération $F(x)$. Ainsi, lors de la sélection du $x^{ème}$ élément de l'ensemble, le client sélectionnera un chiffré de $F(x)$ qui est un chiffré de la pondération α_x. Cette amélioration permet potentiellement de réduire la taille de N. En effet, lors d'un paiement, si le montant dépasse un seuil important, le paiement peut sembler frauduleux. Un paiement inférieur à un certain seuil peut aussi sembler frauduleux. Il

Algorithme 13 : Test d'inégalité final

Input : C_α chiffré RLWE tel que décrit dans 5.2, $C_{T+1}..C_{T_{max}}$ chiffrés RLWE de $T+1..T_{max}$, p le module dans le domaine des clairs pour RLWE, n le degré du polynôme RLWE

Output : Un ensemble de chiffrés RLWE qui permet de savoir le signe de C' pour quelqu'un ayant la clé privée.

```
1  begin
2      //Côté Client
3      for Ti ∈ T + 1..T_max do
4          //on répète pour toutes les valeurs possibles supérieures à T
5          Z_Ti = C_α − C_Ti //D(Z_Ti) vaut 0 si D(C_α) = D(C_Ti) et donc D(C_α) > D(C_T)
6          r ←$ 1..p
7          R_j ←$ 1..p pour j ∈ {1..n − 1}
8          P = R_{n−1} X^{n−1} + .. + R_1 X
9          Z_Ti = Z_Ti + E(P) //On ajoute un aléa sauf pour le coefficient constant
10         Z_Ti = Z_Ti × r //On multiplie les coefficients par un aléa
11     //On réalise une permutation aléatoire dans l'ordre des Z_Ti avant de les renvoyer au serveur.
12     //Côté Serveur
13     for Ti ∈ T + 1..T_max do
14         R = D(Z_Ti)
15         if R = 0 then
16             Lever une alerte de fraude
17     return [C' > 0];
```

faudrait en utilisant des seuils intermédiaires traiter deux fois l'information du montant de la transaction alors qu'en utilisant une fonction de distribution, traiter l'information une seule fois suffit. De plus cette amélioration permet également un résultat plus juste. En effet si une information de paiement est juste au dessus d'un seuil intermédiaire t_i, le paiement est potentiellement moins frauduleux que si l'information dépassait largement ce seuil. Cependant, les techniques de détection de fraude basées sur des seuils sont plus faciles à maîtriser. La solution KODA est également compatible avec des techniques de détection plus poussées.

Il est possible de considérablement réduire les coûts réseaux en préchargeant les ensembles S_i en mémoire chez le client. En effet, plutôt que d'envoyer les S_i pour chaque transaction, il est possible de les précharger chez le client. L'inconvénient principal de cette solution est que l'ensemble S_i est susceptible d'être différent pour chaque acheteur. Ainsi si le client est par exemple un terminal de paiement, il est impossible de précharger le profil de tous les acheteurs dans le terminal. Cette solution peut être retenue dans certains cas d'utilisation.

La solution offre la possibilité de faire des ET sur les informations du client. Considérons 2 données du client, x_1 et x_2. L'idée est de vérifier si ces 2 données sont supérieures respectivement aux seuils t_1 et t_2. Si les deux données sont supérieures aux seuils nous

Configuration	**Classique**	**Multiplication**	**Sécurisée**	**Grands clairs**	**Grandes données**
n	2048	4096	4096	4096	2048
$log(p)$	16	16	16	32	16
$log(q)$	54	109	58	109	54
$l = log(N)$	16	16	16	16	32

TABLE 5.1 Différents paramétrages possibles. Le paramétrage classique est celui permettant de réaliser l'algorithme décrit offrant 128 bits de sécurité. Le second paramétrage permet de réaliser une multiplication pour faire un ET. Le troisième paramétrage offre 256 bits de sécurité. Le quatrième paramétrage permet d'utiliser des pondérations sur 32 bits plutôt que 16. La cinquième configuration utilise des données client sur 32 bits.

souhaitons obtenir une pondération α et sinon une pondération de 0. Pour réaliser ce ET, S_1 est créé de la même façon que dans la solution KODA, en attribuant une pondération α à partir de l'élément t_1. S_2 sera créé en attribuant une pondération 1 à partir de l'élément t_2. Après le choix des $x_1^{ème}$ et $x_2^{ème}$ éléments par le client, il réalise la multiplication homomorphe des 2 éléments sélectionnés. Il obtient ainsi un chiffré de α si $x_1 > t_1$ et $x_2 > t_2$ et un chiffré de 0 dans un cas contraire.

L'inconvénient principal de l'approche est la réalisation d'une multiplication homomorphe qui est coûteuse en terme de coût de calcul mais qui accroît également fortement le bruit. Il est donc nécessaire de réviser le choix du degré n et de la taille des coefficients p et q dans SEAL.

5.3.4 Évaluation de la solution

La configuration de base utilisée pour réaliser KODA utilise SEAL v2.3 avec n le degré du polynôme RLWE qui vaut 2048, la taille des coefficients dans le domaine des clairs est sur 16 bits, ce qui permet de représenter les pondérations α sur $16bits$. Afin de permettre 128 bits de sécurité, la taille des chiffrés est sur 54 bits. Ce qui permet, pour N en dessous de 2^{10}, de déchiffrer correctement. En effet le bruit issu des additions dans le calcul C_α décrit dans la figure 5.2 peut empêcher de déchiffrer correctement si N est trop grand. La taille des données du client est considérée sur 16 bits. D'autres configurations sont décrites dans la table 5.1, une permettant de réaliser une multiplication homomorphe afin de réaliser un ET tel que décrit dans les améliorations possibles, une autre offrant 256 bits de sécurité, la quatrième permettant de travailler avec des pondérations α sur 32 bits et la dernière considère les données du client sur 32 bits.

Nous allons présenter les résultats en trois étapes différentes. La première étape présentera les coûts réseaux nécessaire à l'envoi des données du serveur pour le client (notamment les ensembles S_i). La seconde étape présentera le coût calculatoire sans tenir

Configuration	Classique	Multiplication	Sécurisée	Grands clairs	Grandes données
$taille(Ko) N = 10$	445	926	493	926	$28.(10^6)$
$taille(Ko) N = 20$	877	1798	957	1798	$57.(10^6)$
$taille(Ko) N = 50$	2173	4414	2349	4414	$142.(10^6)$

TABLE 5.2 Coûts en Ko de l'envoi des données du serveur au client pour les 4 configurations différentes.

compte de l'inégalité finale $C = [C' > 0]$. La dernière étape présentera le coût calculatoire et réseau de l'inégalité finale.

Pour les expérimentations, nous avons implémenté le serveur et le client en C++ sur une machine virtuelle Ubuntu 64-bit avec 4Go de RAM et 200Go d'espace disque. La machine hôte a un processeur Intel Core i5 2.4GHz. Les résultats présentés sont une moyenne sur 5 tests.

La table 5.2 présente le coût en Ko des différentes configurations. Les coûts réseaux sont presque linéaires en le nombre de données du client. En effet, le seuil final est envoyé en un chiffré C_T. Les autres chiffrés envoyés dépendent de la taille l et du nombre de données N du client. La taille des chiffrés dépend de n le degré du polynôme et de q la taille d'un coefficient. Puisqu'on utilise le *full batching*, n influe peu sur la taille des données. Nous constatons donc qu'augmenter la sécurité influe peu sur le coût réseau. Utiliser des données client sur plus de 16 bits semble peu réalisable. En effet les coûts réseaux pour des données sur 16 bits sont de l'ordre de quelques Mo si on utilise $N = 50$. Il est alors nécessaire d'avoir un réseau qui a un débit de plusieurs dizaines de Mo/s pour rester dans l'ordre de la centaine de ms. Utiliser des données sur 32 bits coûte plusieurs dizaines de Go d'échanges réseaux, nous verrons en conclusion s'il existe des solutions pour réduire ce coût réseau. Le coût réseau de la configuration multiplication et grands clairs est le même.

Nous supposons que toutes les données envoyées par le serveur pour le client ont été pré-calculées. Ainsi le coût de chiffrement n'intervient pas. Pour la configuration classique ce coût serait d'environ $35ms$ multiplié par N. Il est nécessaire que le serveur possède une mémoire suffisante pour stoker ces chiffrés relatifs à chaque profil de client.

Nous constatons dans la table 5.3 que le coût de cette étape est négligeable si on ne réalise pas de multiplication homomorphe. Cependant, réaliser plusieurs multiplications homomorphes et donc réaliser des ET sur les données du client peut se révéler être un critère limitant. Nous voulons limiter la détection de fraude à quelques centaines de millisecondes. Il est donc possible de réaliser quelques multiplications (de l'ordre d'une dizaine). La taille des données n'influe pas sur les coûts calculatoires. En effet, la taille

Configuration	Classique	Multiplication	Sécurisée	Grands clairs	Grandes données
$temps(ms) N = 10$	0.04	80	0.08	0.15	0.04
$temps(ms) N = 20$	0.08	159	0.16	0.31	0.08
$temps(ms) N = 50$	0.21	402	0.4	0.78	0.21

TABLE 5.3 Coûts en millisecondes pour la seconde étape qui correspond au coût calculatoire sans tenir compte de l'inégalité finale $C = [C' > 0]$. Pour toutes les configurations sauf celle permettant la multiplication, ce coût correspond seulement aux coûts d'additions. Le coût pour la multiplication est celui de N ET.

Configuration	Classique	Multiplication	Sécurisée	Grands clairs	Grandes données
$temps(ms) T_{max} - T = 100$	15	56	28	56	15
$temps(ms) T_{max} - T = 1000$	152	570	283	568	153
$taille(Ko) T_{max} - T = 100$	1350	5450	2900	5450	1350
$taille(Ko) T_{max} - T = 1000$	13500	54500	29000	54500	13500

TABLE 5.4 Coût calculatoire et réseau pour réaliser l'algorithme 13.

des données influe uniquement sur la taille de l'ensemble S_i dont un seul élément est sélectionné.

Nous supposons que la génération de l'aléatoire et son chiffrement sont pré-calculés. En effet, entre deux paiements le client a le temps de réaliser ces opérations.

Pour obtenir les chiffrés de $T + 1..T_{max}$, le client peut chiffrer $1..T_{max} - T$ puis ajouter ces chiffrés au chiffré de C_T envoyé par le serveur. L'opération de chiffrement est pré-calculée.

On constate que les coûts calculatoires décrits dans la table 5.3 sont négligeables par rapport à ces coûts en dehors de la configuration effectuant une multiplication. L'algorithme 13 utilise une multiplication d'un chiffré par un scalaire, cette opération est plus coûteuse que les additions nécessaires à la seconde étape. La valeur de $T_{max} - T$, qui est le paramètre qui détermine combien de multiplications par un scalaire sont nécessaires, est limitant pour la solution. Une valeur de 100 pour ce paramètre permet de réaliser l'algorithme dans un temps raisonnable, cependant, si cette valeur atteint 1000 le coût calculatoire devient limitant. Nous constatons le même résultat sur le coût réseau. En effet $T_{max} - T$ chiffrés sont envoyés par le client. La somme des pondérations qui détermine T_{max} est donc une donnée limitante pour l'algorithme, ce qui peut contraindre les algorithmes d'apprentissage qui génèrent les α et potentiellement rendre la détection de fraude moins précise.

Nous supposons que cette détection de fraude a lieu dans une connexion déjà établie. Le temps d'aller-retour réseau nécessaire pour réaliser l'algorithme ne rentre pas en compte dans les résultats puisque le client et le serveur sont localisés sur la même

Configuration	Classique	Multiplication	Sécurisée	Grands clairs	Grandes données
$temps(ms)10Mo/s$	238	861	414	781	$57.(10^5)$
$temps(ms)100Mo/s$	37	208	67	129	$57.(10^4)$

TABLE 5.5 Temps nécessaire pour réaliser l'algorithme total avec $N = 20$ et $T_{max} - T = 100$, le débit réseau simulé est de 10Mo/s et 100Mo/s. Pour la configuration multiplication nous supposons 10 multiplications, et 10 traitements classiques.

machine pour les simulations. L'utilisation de grandes données n'est pas réalisable même pour un réseau ayant un débit important. L'utilisation de pondérations sur 32 bits est réalisable, cependant la limitation de la valeur de T_{max} rend cette configuration inutile. Il faudrait dans ce cas trouver une solution différente pour réaliser l'inégalité finale. Doubler la sécurité coûte environ un facteur 2. Pour réaliser plusieurs multiplications un débit d'environ $100Mo/s$ est nécessaire pour rester dans un temps raisonnable. Pour la configuration classique un débit de $10Mo/s$ est suffisant.

La propriété d'indistingabilité des chiffrés des ensembles S_i permet au serveur de ne pas révéler la valeur du seuil t_i ni la valeur de la pondération α_i. Afin de réaliser l'inégalité finale, le client doit connaître $T_{max} - T$. Cependant le client n'a aucune information sur T_{max} et ne peut donc pas retrouver T. De plus le client n'a aucune information sur T il ne peut donc pas apprendre d'information sur T_{max} qui est la somme des pondérations.

On suppose que le client ne peut pas tricher avec ses informations de paiement et sélectionne le chiffré correspondant à sa donnée. Il ne peut donc pas sélectionner des chiffrés choisis pour essayer de savoir quand il y a fraude ou non. De plus le client n'apprend pas si une alerte de fraude a été levée par le serveur.

Le serveur n'apprend pas les données du client grâce à la multiplication par un aléatoire décrite dans l'algorithme 13. On suppose que le serveur génère le bruit de la même façon pour tous les chiffrés qu'il envoie au client, ainsi il n'est pas possible de retrouver l'aléa utilisé par le client.

5.4 Conclusion et perspectives

5.4.1 Réduire le coût réseau

Le coût majoritaire pour la solution classique, sans l'utilisation de multiplication, est le coût réseau. Utiliser des données sur plus de 16 bits peut s'avérer utile, ce qui n'est pas réalisable en utilisant la solution classique. Nous avons essayé de trouver une solution alternative permettant de réduire ce coût. La figure 5.3 décrit l'algorithme. Dans cette solution, le serveur transmet au client un chiffré de chaque seuil intermédiaire. Pour

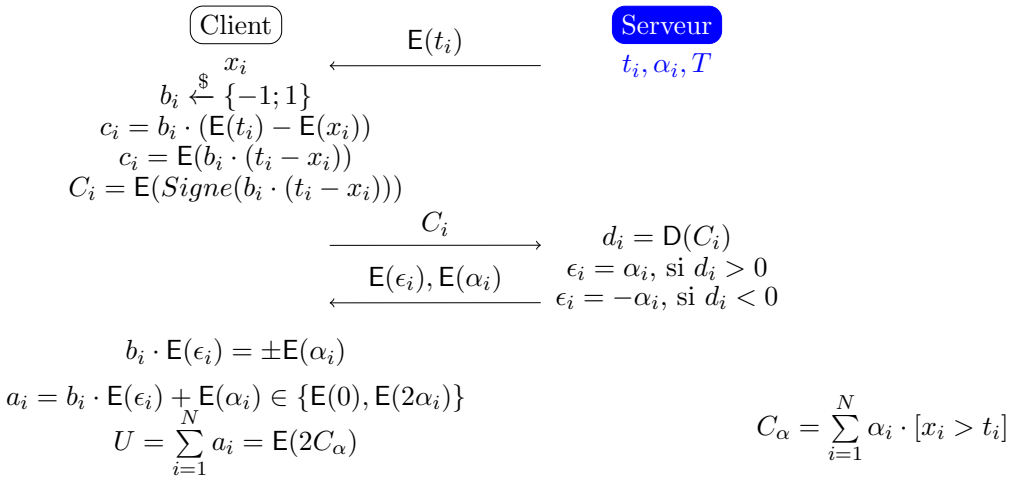

FIGURE 5.3 Protocole permettant de réaliser la détection de fraude de manière privée avec un coût réseau moindre. Le serveur envoie les seuils intermédiaires de manière chiffrée puis le client renvoie le signe de la soustraction entre le seuil et sa donnée, multipliée par aléatoirement 1 ou -1. Le serveur déchiffre ce résultat et envoie un chiffré du poids et un chiffré dépendant du signe. Avec ces chiffrés le client est capable d'obtenir un chiffré de 0 si sa donnée est inférieure au seuil ou un chiffré de 2 fois le poids sinon. À partir de ce moment le client se retrouve à une étape de l'algorithme KODA et peut terminer de la même manière qu'à la figure 5.2

chaque seuil le client va faire la soustraction entre ce seuil et sa donnée correspondante. Afin de cacher le résultat au serveur, l'idée est de multiplier le résultat de la soustraction par un aléatoire b_i tiré de manière uniformément aléatoire dans $\{-1; 1\}$ puis de cacher l'information de $t_i - x_i$ tout en gardant l'information sur le signe de $b_i(t_i - x_i)$. Nous n'avons pas trouvé de solution convenable pour réaliser cette étape. Multiplier le chiffré par un aléatoire très grand permettrait de cacher l'information sur $t_i - x_i$ mais l'information sur le signe serait perdue puisque nous travaillons dans le domaines des chiffrés modulo q. Multiplier par un aléatoire plus petit pour ne pas dépasser le module q permettrait de garder l'information sur le signe mais révélerait des informations sur $t_i - x_i$. Il est possible d'isoler le bit de signe en travaillant modulo 2 dans le domaine des clairs et en réalisant le circuit logique correspondant à l'algorithme. Cependant cette solution se révèle trop coûteuse. Nous n'avons pas trouvé de moyen peu coûteux pour isoler le bit de signe. La suite de l'algorithme consiste à envoyer l'information du signe au serveur qui envoie un chiffré de plus ou moins α_i au client. Le client multiplie ce chiffré par b_i ainsi si t_i est supérieur à x_i il obtient un chiffré de $-\alpha_i$ et un chiffré de α_i sinon. En ajoutant, à ce chiffré, un chiffré de α_i envoyé par le serveur il obtient un chiffré de 0 si la donnée est inférieure au seuil et un chiffré de deux fois la pondération sinon. Il est possible de diviser ce chiffré par 2 pour se retrouver dans une configuration équivalente à l'algorithme décrit dans la figure 5.2. La fin des deux algorithmes est la même.

5.4.2 Résulats

La solution présentée permet de réaliser une détection de fraude de manière privée. Les données du client ne sont pas révélées au serveur et les données utilisées par l'algorithme de détection ne sont pas révélées au client. Le coût principal de la solution classique est le coût réseau. Avec un réseau possédant un débit de 10Mo/s il est possible de réaliser la détection de fraude en se basant sur 20 données du client en environ 200 ms. L'algorithme présenté possède de nombreux avantages :

— Possibilité d'utiliser une fonction de distribution plutôt que des seuils, de manière gratuite, afin d'utiliser des algorithmes de détection de fraude plus précis.
— Il est possible d'utiliser des multiplications afin de réaliser des ET entre certaines données du client. La multiplication a un coût non négligeable. Cependant, avec un réseau de 100Mo/s, on peut réaliser la détection de fraude basée sur 10 données et 10 ET en environ 200 ms.
— Doubler la sécurité pour un facteur 2 dans les coûts.

Certains points sont contraignants. Les pondérations utilisées doivent être des entiers petits pour que l'algorithme 13 soit efficace. La taille des données du client doit être sur

au plus de 20 bits, au delà l'algorithme prendrait trop de temps.

Conclusion et Perspectives

La cryptographie est devenue un moyen indispensable de protéger les informations et d'avoir confiance dans les échanges dans le domaine du numérique. Cependant les techniques classiques ne permettent pas de protéger les données d'un client vis à vis d'un serveur lors d'un échange pair à pair. L'apparition du chiffrement homomorphe a permis la conception de nombreux algorithmes permettant d'améliorer la protection de la vie privée d'un client. Mais, le chiffrement homomorphe a longtemps été réputé comme peu efficace. En effet, certains algorithmes nécessitant une profondeur multiplicative importante sont très coûteux à réaliser dans le domaine chiffré. Pourtant, ce n'est pas le cas de tous les algorithmes, pour nombre d'entre eux il est possible d'utiliser le chiffrement homomorphe de façon relativement efficace.

Nous avons étudié les coûts de différentes librairies permettant de faire du chiffrement homomorphe pour différentes tailles du module dans l'espace des clairs. Notamment SEAL, FV-NFLlib et HElib. Le développement de SEAL et HElib est en continuelle évolution et ces librairies sont très prometteuses. SEAL est la plus accessible grâce à sa facilité d'utilisation et c'est la plus efficace lorsque les paramètres ne sont pas trop grands. HElib est la librairie permettant de faire le plus d'opérations différentes sur les chiffrés et la plus efficace asymptotiquement. Il serait intéressant d'intégrer les optimisations de certaines librairies à d'autres. L'utilisation de NFLlib dans les librairies SEAL et HElib pourrait renforcer les performances.

Les données génomiques sont liées aux individus et soulèvent des problèmes de vie privée qui sont difficiles à contourner. La protection de ces données n'est pas une tâche aisée, elle est cependant capitale. Nous avons présenté une solution à un des potentiels problèmes du stockage externalisé de données génomiques.

Notre solution permet, en outre du stockage confidentiel, d'interroger le serveur sur la présence ou l'absence d'une mutation dans le génome stocké. Nous assurons la confidentialité des données stockées, mais également celle de la requête, vis-à-vis du serveur. La solution présentée est très efficace d'un point de vue mémoire et temps

calculatoire. L'amélioration la plus importante pour la solution serait de trouver un moyen permettant d'interroger le serveur sur la présence de plusieurs mutations autrement qu'en effectuant une requête par mutation.

Dans le domaine de la bio-informatique, le calcul d'un ensemble maximal est souvent utilisé pour connaître la corrélation entre un génome individuel et les génomes d'une base de données. Nous avons présenté une solution non-interactive fondée sur le chiffrement homomorphe permettant de calculer l'ensemble des ensembles maximaux entre une requête et une base de données de manière privée.

L'utilisation de TFHE a été nécessaire pour réaliser cette solution. Même si l'algorithme semble non utilisable actuellement à cause des temps de calcul trop importants, l'évolution dans le domaine du FHE est rapide et nous espérons que celle-ci permette de rendre l'algorithme plus performant.

La dernière contribution du manuscrit est une solution permettant de réaliser une détection de fraude de manière privée. Ainsi les données du client ne sont pas révélées au serveur qui s'occupe de la détection. Nous avons montré qu'il était possible de réaliser la détection sur un nombre de données suffisant en moins de 200ms en se basant sur un protocole proche de l'OT. Ce temps semble suffisamment petit pour être intégré à un paiement. L'inconvénient majeur de l'algorithme est que la taille des données ne peut dépasser quelques bits, au delà l'algorithme devient trop coûteux en bande passante. Trouver des astuces permettant de travailler sur des données plus grandes permettrait d'avoir un algorithme utilisable.

À travers ces travaux nous avons contribué au développement des applications du chiffrement homomorphe et montré qu'il était possible de réaliser des protocoles utiles s'exécutant en un temps acceptable.

Publications

— J. S. Sousa, C. Lefebvre, Z. Huang, J. L. Raisaro, C. Aguilar Melchor, M. Killijian, and J. Hubaux, "Efficient and secure outsourcing of genomic data storage," IACR Cryptol. ePrint Arch., vol. 2017, p. 228, 2017

— C. Aguilar Melchor, M. Killijian, C. Lefebvre, and T. Ricosset, "A comparison of the homomorphic encryption libraries helib, SEAL and fv-nfllib," in Innovative Security Solutions for Information Technology and Communications - 11th International Conference, SecITC 2018, Bucharest, Romania, November 8-9, 2018, Revised Selected Papers (J. Lanet and C. Toma, eds.), vol. 11359 of Lecture Notes in Computer Science, pp. 425–442, Springer, 2018

— C. Aguilar Melchor, M. Killijian, and P. I. andCédric Lefebvre, "idash privacy and security workshop 2018." `http://www.humangenomeprivacy.org/2018/`

—

Bibliographie

[1] D. Müller, Les codes secrets décryptés. City éditions, 2007.

[2] D. E. Standard et al., "Data encryption standard," Federal Information Processing Standards Publication, p. 112, 1999.

[3] J. Daemen and V. Rijmen, "Reijndael : The advanced encryption standard.," Dr. Dobb's Journal : Software Tools for the Professional Programmer, vol. 26, no. 3, pp. 137–139, 2001.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, 1978.

[5] S. Goldwasser and S. Micali, "Probabilistic encryption," Journal of computer and system sciences, vol. 28, no. 2, pp. 270–299, 1984.

[6] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," Foundations of Secure Computation, Academia Press, pp. 169–179, 1978.

[7] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in Theory of cryptography conference, pp. 325–341, Springer, 2005.

[8] C. Aguilar Melchor, P. Gaborit, and J. Herranz, "Additively homomorphic encryption with d-operand multiplications," in Proceedings of the 30th Annual Conference on Advances in Cryptology, CRYPTO'10, (Berlin, Heidelberg), p. 138–154, Springer-Verlag, 2010.

[9] C. Gentry, "Fully homomorphic encryption using ideal lattices," vol. 9, pp. 169–178, 01 2009.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping." Cryptology ePrint Archive, Report 2011/277, 2011. `https://eprint.iacr.org/2011/277`.

[11] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp." Cryptology ePrint Archive, Report 2012/078, 2012. `https://eprint.iacr.org/2012/078`.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in Proceedings of the 3rd Innovations

in Theoretical Computer Science Conference, ITCS '12, (New York, NY, USA), p. 309–325, Association for Computing Machinery, 2012.

[13] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," in 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, pp. 97–106, 2011.

[14] C. Aguilar Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey, "Recent advances in homomorphic encryption : A possible future for signal processing in the encrypted domain," IEEE Signal Processing Magazine, vol. 30, no. 2, pp. 108–117, 2013.

[15] Ö. Kocabaş and T. Soyata, "Medical data analytics in the cloud using homomorphic encryption," in E-Health and Telemedicine : Concepts, Methodologies, Tools, and Applications, pp. 751–768, IGI Global, 2016.

[16] M. Kim and K. Lauter, "Private genome analysis through homomorphic encryption," in BMC medical informatics and decision making, vol. 15, pp. 1–12, BioMed Central, 2015.

[17] L. Xiao, O. Bastani, and I.-L. Yen, "An efficient homomorphic encryption protocol for multi-user systems.," IACR Cryptol. EPrint Arch., vol. 2012, p. 193, 2012.

[18] S. S. Shinde, S. Shukla, and D. Chitre, "Secure e-voting using homomorphic technology," International Journal of Emerging Technology and Advanced Engineering, vol. 3, no. 8, pp. 203–206, 2013.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe : Fast fully homomorphic encryption over the torus," Journal of Cryptology, vol. 33, 04 2019.

[20] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[21] P. Bachmann, Die analytische zahlentheorie, vol. 2. Teubner, 1894.

[22] O. 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, (New York, NY, USA), p. 84–93, Association for Computing Machinery, 2005.

[23] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in Advances in Cryptology – EUROCRYPT 2010 (H. Gilbert, ed.), (Berlin, Heidelberg), pp. 1–23, Springer Berlin Heidelberg, 2010.

[24] C. F. Gauss, Disquisitiones arithmeticae, vol. 157. Yale University Press, 1966.

[25] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," J. ACM, vol. 45, p. 965–981, Nov. 1998.

[26] B. Chor and N. Gilboa, "Computationally private information retrieval," in Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pp. 304–313, 1997.

[27] C. Aguilar Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, "Xpir : Private information retrieval for everyone," Proceedings on Privacy Enhancing Technologies, vol. 2016, no. 2, pp. 155–174, 2016.

[28] M. O. Rabin, "How to exchange secrets with oblivious transfer," IACR Cryptology ePrint Archive, vol. 2005, p. 187, 2005.

[29] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," Communications of the ACM, vol. 28, no. 6, pp. 637–647, 1985.

[30] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, (New York, NY, USA), p. 818–829, Association for Computing Machinery, 2016.

[31] S. Halevi and V. Shoup, "Design and implementation of a homomorphic-encryption library," tech. rep., MIT, 2014.

[32] S. Halevi and V. Shoup, "Algorithms in helib." Cryptology ePrint Archive, Report 2014/106, 2014. https://eprint.iacr.org/2014/106.

[33] H. Chen, K. Han, Z. Huang, and A. Jalali, "Simple encrypted arithmetic library - seal (v2.3)," tech. rep., Microsoft, December 2017.

[34] CryptoExperts, "Fv-nfllib." https://github.com/CryptoExperts/FV-NFLlib.

[35] T. Ricosset, "Helib-multiprecision," 2017.

[36] R. P. Kim Laine, Hao Chen, "Simple encrypted arithmetic library - seal (v2.1)," tech. rep., Microsoft, September 2016.

[37] J.-C. Bajard, J. Eynard, A. Hasan, and V. Zucca, "A full rns variant of fv like somewhat homomorphic encryption schemes." Cryptology ePrint Archive, Report 2016/510, 2016. https://eprint.iacr.org/2016/510.

[38] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors." Cryptology ePrint Archive, Report 2015/046, 2015. https://eprint.iacr.org/2015/046.

[39] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

[40] T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption schemes fv and yashe." Cryptology ePrint Archive, Report 2014/062, 2014. https://eprint.iacr.org/2014/062.

[41] "Sequencing-human-genome-cost." https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost.

[42] M. M. Clark, A. Hildreth, S. Batalov, Y. Ding, S. Chowdhury, K. Watkins, K. Ellsworth, B. Camp, C. I. Kint, C. Yacoubian, et al., "Diagnosis of genetic diseases in seriously ill children by rapid whole-genome sequencing and automated phenotyping and interpretation," Science translational medicine, vol. 11, no. 489, p. eaat6177, 2019.

[43] D. G. R. Evans, J. Barwell, D. M. Eccles, A. Collins, L. Izatt, C. Jacobs, A. Donaldson, A. F. Brady, A. Cuthbert, R. Harrison, et al., "The angelina jolie effect : how high celebrity profile can have a major impact on provision of cancer related services," Breast Cancer Research, vol. 16, no. 5, p. 442, 2014.

[44] "Pure storage case." `https://media.erepublic.com/document/Pure_Storage_Case_UC_Berkeley.pdf`.

[45] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for snps with cloud computing," Genome biology, vol. 10, no. 11, p. R134, 2009.

[46] M. Humbert, E. Ayday, J.-P. Hubaux, and A. Telenti, "Addressing the concerns of the lacks family : quantification of kin genomic privacy," in Proceedings of the 2013 ACM SIGSAC conference on Computer communications security, pp. 1141–1152, ACM, 2013.

[47] Y. Erlich, T. Shor, I. Pe'er, and S. Carmi, "Identity inference of genomic data using long-range familial searches," Science, vol. 362, no. 6415, pp. 690–694, 2018.

[48] C. Phillips, "The golden state killer investigation and the nascent field of forensic genealogy," Forensic Science International : Genetics, vol. 36, pp. 186–188, 2018.

[49] D. H. Kaye and M. E. Smith, "Dna identification databases : legality, legitimacy, and the case for population-wide coverage," Wis. L. Rev., p. 413, 2003.

[50] D. Botstein and N. Risch, "Discovering genotypes underlying human phenotypes : past successes for mendelian disease, future approaches for complex disease," Nature genetics, vol. 33, no. 3s, p. 228, 2003.

[51] H. Im, E. Gamazon, D. Nicolae, and N. Cox, "On sharing quantitative trait gwas results in an era of multiple-omics data and the limits of genomic privacy," AmericanJournal of Human Genetics, vol. 90, no. 4, p. 591, 2012.

[52] M. Humbert, K. Huguenin, J. Hugonot, E. Ayday, and J.-P. Hubaux, "De-anonymizing genomic databases using phenotypic traits," Proceedings on Privacy Enhancing Technologies, vol. 2015, no. 2, pp. 99–114, 2015.

[53] G. Naik, "Family secrets : An adopted man's 26-year quest for his father," The Wall Street Journal, 2009.

[54] J. S. Goldman, S. E. Hahn, J. W. Catania, S. Larusse-Eckert, M. B. Butson, M. Rumbaugh, M. N. Strecker, J. S. Roberts, W. Burke, R. Mayeux, et al., "Genetic counseling and testing for alzheimer disease : joint practice guidelines of the american college of medical genetics and the national society of genetic counselors," Genetics in Medicine, vol. 13, no. 6, p. 597, 2011.

[55] G. Barbujani and V. Colonna, "Human genome diversity : frequently asked questions," Trends in Genetics, vol. 26, no. 7, pp. 285–295, 2010.

[56] P. J. McLaren, J. L. Raisaro, M. Aouri, M. Rotger, E. Ayday, I. Bartha, M. B. Delgado, Y. Vallet, H. F. Günthard, M. Cavassini, et al., "Privacy-preserving genomic testing in the clinic : a model using hiv treatment," Genetics in medicine, vol. 18, no. 8, p. 814, 2016.

[57] K. Lauter, A. López-Alt, and M. Naehrig, "Private computation on encrypted genomic data," in International Conference on Cryptology and Information Security in Latin America, pp. 3–27, Springer, 2014.

[58] S. Wang, Y. Zhang, W. Dai, K. Lauter, M. Kim, Y. Tang, H. Xiong, and X. Jiang, "Healer : homomorphic computation of exact logistic regression for secure rare disease variants analysis in gwas," Bioinformatics, vol. 32, no. 2, pp. 211–218, 2015.

[59] N. Karvelas, A. Peter, S. Katzenbeisser, E. Tews, and K. Hamacher, "Privacy-preserving whole genome sequence processing through proxy-aided oram," in Proceedings of the 13th Workshop on Privacy in the Electronic Society, pp. 1–10, ACM, 2014.

[60] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram : An extremely simple oblivious ram protocol," in Proceedings of the 2013 ACM SIGSAC Conference on Computer, Communications Security, CCS '13, (New York, NY, USA), p. 299–310, Association for Computing Machinery, 2013.

[61] . G. P. Consortium et al., "A global reference for human genetic variation," Nature, vol. 526, no. 7571, p. 68, 2015.

[62] C. Aguilar Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, and T. Lepoint, "Nfllib : Ntt-based fast lattice library," in Topics in Cryptology - CT-RSA 2016 (K. Sako, ed.), (Cham), pp. 341–356, Springer International Publishing, 2016.

[63] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors." Cryptology ePrint Archive, Report 2015/046, 2015. `https://eprint.iacr.org/2015/046`.

[64] M. G. Ross, C. Russ, M. Costello, A. Hollinger, N. J. Lennon, R. Hegarty, C. Nusbaum, and D. B. Jaffe, "Characterizing and measuring bias in sequence data," Genome biology, vol. 14, no. 5, p. R51, 2013.

[65] "idash privacy and security workshop 2016." `http://www.humangenomeprivacy.org/2016/`.

[66] G. S. Cetin, H. Chen, K. Laine, K. Lauter, P. Rindal, and Y. Xia, "Private queries on encrypted genomic data." Cryptology ePrint Archive, Report 2017/207, 2017. `https://eprint.iacr.org/2017/207`.

[67] J. H. Cheon, M. Kim, and Y. S. Song, "Secure searching of biomarkers using hybrid homomorphic encryption scheme," IACR Cryptol. ePrint Arch., vol. 2017, p. 294, 2017.

[68] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol. 13, p. 422–426, July 1970.

[69] R. Durbin, "Efficient haplotype matching and storage using the positional burrows-wheeler transform (pbwt).," Bioinformatics, vol. 30, no. 9, pp. 1266–1272, 2014.

[70] K. Shimizu, K. Nuida, and G. Ratsch, "Efficient privacy-preserving string search and an application in genomics," Bioinformatics, vol. 32, pp. 1652–1661, 03 2016.

[71] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors : Conceptually-simpler, asymptotically-faster, attribute-based." Cryptology ePrint Archive, Report 2013/340, 2013. `https://eprint.iacr.org/2013/340`.

[72] Y. Kou, C.-T. Lu, S. Sirwongwattana, and Y.-P. Huang, "Survey of fraud detection techniques," in IEEE International Conference on Networking, Sensing and Control, 2004, vol. 2, pp. 749–754, IEEE, 2004.

[73] J. Lopes, O. Belo, and C. Vieira, "Applying user signatures on fraud detection in telecommunications networks," in Industrial Conference on Data Mining, pp. 286–299, Springer, 2011.

[74] N. Laleh and M. A. Azgomi, "A taxonomy of frauds and fraud detection techniques," in International Conference on Information Systems, Technology and Management, pp. 256–267, Springer, 2009.

[75] D. Malekian and M. R. Hashemi, "An adaptive profile based fraud detection framework for handling concept drift," in 2013 10th International ISC Conference on Information Security and Cryptology (ISCISC), pp. 1–6, IEEE, 2013.

[76] A. Abdallah, M. A. Maarof, and A. Zainal, "Fraud detection system : A survey," Journal of Network and Computer Applications, vol. 68, pp. 90–113, 2016.

[77] R. Canillas, R. Talbi, S. Bouchenak, O. Hasan, L. Brunie, and L. Sarrat, "Exploratory study of privacy preserving fraud detection," in Proceedings of the 19th International Middleware Conference Industry, pp. 25–31, 2018.

[78] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits.," in USENIX Security Symposium, vol. 201, pp. 331–335, 2011.

[79] A. C.-C. Yao, "How to generate and exchange secrets," in Foundations of Computer Science, 1986., 27th Annual Symposium on, pp. 162–167, IEEE, 1986.

[80] V. Kolesnikov and T. Schneider, "Improved garbled circuit : Free xor gates and applications," in International Colloquium on Automata, Languages, and Programming, pp. 486–498, Springer, 2008.

[81] J. S. Sousa, C. Lefebvre, Z. Huang, J. L. Raisaro, C. Aguilar Melchor, M. Killijian, and J. Hubaux, "Efficient and secure outsourcing of genomic data storage," IACR Cryptol. ePrint Arch., vol. 2017, p. 228, 2017.

[82] C. Aguilar Melchor, M. Killijian, C. Lefebvre, and T. Ricosset, "A comparison of the homomorphic encryption libraries helib, SEAL and fv-nfllib," in Innovative Security Solutions for Information Technology and Communications - 11th International Conference, SecITC 2018, Bucharest, Romania, November 8-9, 2018, Revised Selected Papers (J. Lanet and C. Toma, eds.), vol. 11359 of Lecture Notes in Computer Science, pp. 425–442, Springer, 2018.

[83] C. Aguilar Melchor, M. Killijian, and P. I. andCédric Lefebvre, "idash privacy and security workshop 2018." `http://www.humangenomeprivacy.org/2018/`.

Résumé

Le chiffrement homomorphe est une primitive cryptographique permettant de réaliser des opérations arithmétiques sur des données chiffrées. Grâce à celui-ci il est possible de confier des calculs à un agent externe, sans que les données traitées ou les résultats obtenus ne soient accessibles à cet agent. Ainsi il est possible de réaliser des protocoles améliorant la protection de la vie privée dans de nombreux domaines. La découverte de schémas de chiffrement homomorphe relativement efficaces, en général fondés sur les réseaux euclidiens, permet d'envisager des applications intéressantes à un coût calculatoire acceptable.

Cette thèse s'attelle dans un premier temps à étudier les principales implémentations rendues publiques de ces schémas, notamment en comparant leur performance. L'étude des librairies FV-NFLlib, SEAL et HElib permet de choisir celle donnant des résultats optimaux en fonction de la situation.

Un domaine d'application primordial de la protection de la vie privée est celui des données génomiques. Il est crucial de protéger ces données puisqu'elles sont liées aux individus et soulèvent des problèmes de vie privée qui sont difficiles à contourner. Nous proposons un protocole permettant d'externaliser les données génomiques chiffrées vers un serveur tout en ayant la possibilité de tester la présence d'un élément (une mutation) et ce de manière privée. Le protocole est basé sur un retrait d'information privée et est efficace d'un point de vue consommation mémoire et performance temporelle. Les techniques utilisées permettent une protection de la vie privée optimale.

D'autres opérations sont importantes sur les données génomiques, notamment l'étude de la corrélation entre certaines données. Nous montrons la possibilité de réaliser une solution permettant de calculer un ensemble maximal de manière privée entre un élément et une base de données. La profondeur multiplicative du protocole est trop importante pour utiliser les schémas classiques, c'est pourquoi nous avons exploité TFHE.

Pour finir, nous avons mis en place un algorithme permettant de détecter une fraude lors d'un paiement en ligne en utilisant le calcul homomorphe. Cet algorithme permet de prédire si un paiement est potentiellement frauduleux sans apprendre les informations de ce paiement.

Abstract

Homomorphic encryption is a cryptographic primitive that allows arithmetic operations to be performed on encrypted data. It allows to entrust computations to an external agent, without the processed or the resulting data being accessible to this agent. This allows the creation of privacy-enhancing protocols in various domains. The discovery of reasonably efficient homomorphic encryption schemes, generally based on Euclidean networks, leads to interesting applications at an acceptable computational cost.

This thesis first studies the main publicly available implementations of these schemes, notably by comparing their performance. The study of the FV-NFLlib, SEAL and HElib libraries enables to select the optimal library according to the context.

A key area of application for privacy is genomic data. It is crucial to protect this data since it is linked to individuals and raises significant privacy issues. We propose a protocol to externalize encrypted genomic data to a server while being able to test for the presence of an element (a mutation) in a private manner. The protocol is based on private information retrieval and is efficient from both a memory consumption and time performance point of view. The techniques used allow an optimal privacy protection.

Other operations are important on genomic data, such as the study of the correlation between some data. We show the possibility of realizing a solution to compute a maximal set in a private way between an element and a database. The multiplicative depth of the protocol is too important to use classical schemes, that is why we used TFHE.

Finally, we have implemented an algorithm to detect fraud during an online transaction using homomorphic computation. This algorithm allows to predict if a transaction is potentially fraudulent without learning the information of this transaction.