PAnTHErS : un outil d'aide pour l'analyse et l'exploration d'algorithmes de chiffrement homomorphe

Cyrielle Feron

UNIVERSITE BRETAGNE LOIRE MATHSTIC

ENSTA Bretagne

THESE DE DOCTORAT DE

L'ECOLE NATIONALE SUPERIEURE
DE TECHNIQUES AVANCEES BRETAGNE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Cyrielle FERON

PuTHEr : un outil d'aide pour l'analyse et l'exploration d'algorithmes de chiffrement homomorphe.

Thèse présentée et soutenue à Brest, le 14 novembre 2018
Unité de recherche : Laboratoire Lab-STICC, UMR, 6285

Rapporteurs avant soutenance :

Lilian BOSSUET	Professeur, Université Jean-Monnet, Laboratoire Hubert Curien, Saint Etienne
Renaud SIRDEY	Directeur de recherche au CEA Saclay

Composition du Jury :

Président :	Régis LEVEUGLE	Professeur, INP, Laboratoire TIMA, Grenoble
Examinateurs :	Lilian BOSSUET	Professeur, Université Jean-Monnet, Laboratoire Hubert Curien, Saint Etienne
	Caroline FONTAINE	Chargée de Recherche au CNRS, ENS Paris-Saclay, LSV
	Renaud SIRDEY	Directeur de recherche au CEA Saclay
Dir. de thèse :	Loïc LAGADEC	Professeur, ENSTA Bretagne, Lab-STICC, Brest
Co-enc. de thèse :	Vianney LAPOTRE	Maître de conférences, Université de Bretagne-Sud, Lab-STICC, Lorient

Remerciements

Je tiens à débuter ces remerciements pour exprimer ma reconnaissance envers mon directeur de thèse Loïc LAGADEC et mon encadrant Vianney LAPOTRE. Comme tout doctorant, il m'aurait été difficile de réaliser ce projet à bien sans leurs aides et leurs conseils avisés. Dès mon arrivée, j'ai apprécié la confiance qu'ils m'ont donné et l'autonomie qu'ils m'ont laissé pour mener mes travaux, au quotidien comme pour les points essentiels de mes travaux. Je trouve que notre trio a très bien fonctionné durant ces 3 ans, ce qui m'a permis d'avoir des conditions idéales de travail. Je remercie spécialement Loïc pour m'avoir fait partager son expertise et son expérience en tant que directeur de thèse. Cela m'a permis de bien comprendre les enjeux d'une thèse et les spécificités du monde de la recherche. J'ai également apprécié notre cohésion d'équipe, ce qui a engendré une bonne dynamique (et une bonne entente au quotidien). De plus, je remercie Vianney pour sa réactivité en réponse à mes mails. Il était plus difficile pour nous d'échanger du fait qu'il soit situé sur Lorient. Toutefois, j'ai apprécié sa disponibilité, la clarté dans ses explications et sa participation active lors de nos réunions.

Je remercie ensuite mes rapporteurs de thèse Lilian BOSSUET et Renaud SIRDEY pour leur lecture attentive de mon manuscrit. Je remercie également les autres membres du jury : Régis LEVEUGLE, président du jury, et Caroline FONTAINE. Je remercie plus particulièrement Caroline que j'ai rencontrée dès le début de ma thèse et qui a répondu avec enthousiasme à mes interrogations dans le domaine de l'homomorphe. Elle a su se montrer disponible et j'ai pu discuter librement avec elle, notamment lors des journées FHE du Lab-STICC.

De façon plus générale, je remercie l'ensemble des collègues de mon équipe. Plus particulièrement, je remercie Ciprian pour toutes les fois où il m'a aidé techniquement, notamment en programmation, et ce, toujours avec le sourire et la bonne humeur. Je remercie également Pascale avec qui j'ai eu de nombreuses discussions tant professionnelles (enseignement, projets) que personnelles. Je remercie les doctorants et post-doctorants avec qui j'ai partagé mon quotidien (par ordre alphabétique) : Bastien, Fatma, Hannah, Mohamad, Nicolas, Théotime et Vincent. J'ai apprécié l'entre-aide commune bien que l'on travaille sur des sujets différents. De plus, je remercie mes stagiaires Quentin et Jean pour avoir travaillé sur des sujets connexes à ma thèse. Je remercie Quentin pour son énorme (dirait-il) enthousiasme durant son stage de fin de master effectué à l'ENSTA Bretagne. J'ai apprécié les discussions (pertinentes) que j'ai partagé avec lui sur divers sujets de cryptographie. Et, je remercie Jean pour son sérieux et sa curiosité dont il a fait preuve tout au long de son stage. Enfin, je remercie Annick et Michèle, toujours disponibles, qui ont su répondre à mes questions liées au côté administratif de la thèse et des missions que j'ai effectuées.

Bien que l'on ne se voyait pas souvent, je remercie Vincent, Donald et Guillaume, trois doctorants en chiffrement homomorphe, qui répondaient toujours présent à mes demandes d'explications supplémentaires. Après leurs explications, les passages complexes d'articles scientifiques me semblaient tout de suite plus clairs à comprendre.

Jusqu'à la fin de ma deuxième année de master, je n'envisageais pas de faire des études au-delà de Bac+5. Mes encadrants de stage de fin d'études, Gaëtan LE GUELVOUIT et Christophe BIDAN, m'ont encouragé à postuler pour une thèse en indiquant que j'avais le profil pour être une bonne doctorante. S'ils n'avaient pas émis cette hypothèse, je n'aurai jamais envisagé de postuler à l'ENSTA Bretagne. Je les remercie donc tous les deux pour m'avoir fait changer d'avis sur ce point.

Bien sûr, je remercie Audrey, ma binôme de master et amie : nous avons partagé (presque tous les jours) nos vies de doctorante. Nous avons beaucoup échangé sur nos thèses respectives et nous nous sommes également entre-aidées. De même, je la remercie grandement pour le magnifique logo de PAnTHErS qu'elle a créé sur son temps personnel.

Je remercie chaleureusement Rémi, mon conjoint, qui connaît maintenant mes travaux de thèse (et PAnTHErS) autant que moi. Sur les 3 ans, il a été présent, a participé aux relectures (d'articles et du manuscrit), a écouté (beaucoup) de répétitions de présentations orales, m'a conseillé pour les tutoriels et m'a aidé lorsque j'étais bloquée dans mon implémentation. Je lui dis un grand MERCI pour avoir participé à tout cela (et m'avoir supporté lorsque ses remarques ne me plaisaient pas).

Je n'oublie pas les relectures de manuscrit faites par ma mère, Geneviève, afin de trouver les dernières fautes d'orthographe. Je l'en remercie vivement. J'en profite également pour remercier le reste de ma famille : mon père et mes deux sœurs. Comme ma mère, ils m'ont soutenu et conseillé tout au long de mes études.

Pour finir, je fais une dédicace spéciale à mes camarades de la Team Badminton et Team Gainage. Entre l'écriture d'un article et l'implémentation d'un programme, j'ai réussi à trouver le temps de venir faire du sport avec eux le midi. Cela a toujours été pour moi un réel plaisir de les retrouver pour me défouler, faire le maître du temps (au gainage) et surtout rire avec eux. Je les remercie également d'être venus assister à ma soutenance bien qu'ils envisageaient de ne rien y comprendre. Je conserve avec nostalgie la coupe remportée lors d'un fameux tournoi international de badminton de l'ENSTA Bretagne.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Chapitre 1

Introduction

1.1 Introduction générale

La sécurisation des données devient un problème majeur dans notre société. Selon une étude du CSA [1] en 2017 [2], 85% des français déclarent être préoccupés par la protection de leurs données personnelles en général et 90% sont préoccupés pour leurs données disponibles en ligne. Les données en ligne ou *numériques* correspondent aux informations renseignées sur des sites internet *p. ex.* service de mails, réseaux sociaux, banque en ligne,... Autrement dit, il s'agit des informations d'identités (nom, prénom, date de naissance), personnelles (adresse mail, adresse postale) et de vie privée (toute information postée *p. ex.* sur les réseaux sociaux).

Le volume des données numériques croît considérablement au cours des années. En effet, "*Tous les deux jours, l'humanité produit autant d'information que ce qu'elle a généré depuis l'aube de la civilisation jusqu'en 2003. Plus de 90% des données disponibles aujourd'hui ont été produites ces 2 dernières années. Et ce volume d'information numérique double tous les deux ans.*" [3]. Afin de stocker toutes leurs données et disposant d'un espace de stockage limité sur leur machine personnelle, les utilisateurs ont alors recours au stockage des données par des tiers *p. ex.* Google Drive, un service de Google permettant de stocker des documents de type Word, PDF, Excel sur le serveur Google. Par ailleurs, de plus en plus d'applications stockent des informations concernant les utilisateurs sur des serveurs distants. En effet, afin d'utiliser les services d'une application Android ou Apple, une inscription demandant à l'utilisateur de renseigner, entre autres, son nom, son prénom, sa date de naissance et son adresse mail est requise. Ces données numériques sont alors conservées sur les serveurs du créateur de l'application. La solution de stockage des données par des tiers est utilisée par près de 1 754 millions d'utilisateurs en 2017 [4]. Cette solution entraîne des questions de sécurisation des données.

1.1.1 Sécurisation des données numériques

La sécurisation des données vise à protéger les informations personnelles contre autrui. Plus précisément, la sécurisation des données numériques vise à préserver la confidentialité, l'authenticité, l'intégrité, la non-répudiation et la disponibilité des données. Autrement dit, l'émetteur d'un message (= donnée) ne doit pas pouvoir nier l'envoi de son message

1. Consumer, Science & Analytics, institut d'études marketing et d'opinion.
2. https://www.csa.eu/fr/survey/les-français-et-la-protection-de-leurs-donnees-personnelles
3. https://www.franceculture.fr/emissions/la-methode-scientifique/big-data-des-chiffres-et-des-chiffres
4. https://www.statista.com/statistics/499558/worldwide-personal-cloud-storage-users/

(non-répudiation), ce message doit toujours être disponible (disponibilité) et de plus, une tierce personne ne doit pas pouvoir :

- avoir accès au message clair/lisible/intelligible (confidentialité),
- se faire passer pour quelqu'un d'autre (authentification),
- substituer le message clair/lisible/intelligible (ou seulement une partie) par un autre message (intégrité).

Toute donnée numérique mise à disposition sur un serveur distant est protégée en respectant ces cinq principes fondamentaux. Ces principes sont garantis d'après les propriétés présentées dans les paragraphes suivants en prenant le cas d'un échange entre un utilisateur et un serveur.

La disponibilité est garantie par le serveur qui va devoir mettre en place des mesures techniques pour que l'ensemble des clients puissent accéder simultanément aux données (redondance des serveurs, par exemple).

Plusieurs protocoles de sécurisation existent comme *p. ex.* SSL (Secure Sockets Layer [5]) [FKK11] / TLS (Transport Layer Security [6]) [Die08]. Ces protocoles permettent entre autres l'authentification du serveur. L'utilisation de MAC (*Message Authentication Code* [7], *p. ex.* HMAC [KCB97], permet également de vérifier l'authenticité d'un message.

De plus, un MAC permet également de valider l'intégrité d'un message. L'intégrité est garantie en utilisant notamment des fonctions de hachages [FIP12]. Par exemple, l'utilisateur envoie conjointement un message m ainsi que son hashé $h(m)$ au serveur. Ce dernier peut vérifier l'intégrité du message reçu m' en vérifiant que $h(m') = h(m)$. Si tel est le cas, alors le serveur a la garantie que $m' = m$.

Par ailleurs, si l'utilisateur envoie une signature numérique [FIP13] de son message, alors il ne pourrait plus nier l'envoi de son message : la non-répudiation est assurée.

Enfin, la confidentialité est gérée en utilisant des systèmes de chiffrement pour rendre les messages inintelligibles. Néanmoins, avec les systèmes de chiffrement actuels, la confidentialité n'est plus respectée lorsqu'un utilisateur demande la modification ou l'utilisation d'un message par le serveur car ce dernier a alors besoin de posséder le message en clair.

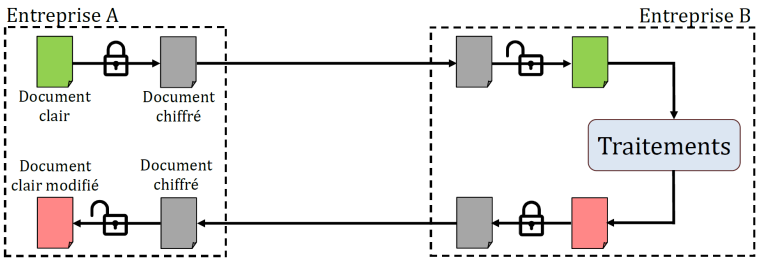


FIGURE 1.1 – Modification d'un document par une entreprise tierce avec les méthodes de chiffrement actuels.

La Figure 1.1 illustre le cas particulier ci-dessus *c-à-d* présente la modification d'un document par un service proposé par une entreprise tierce, appelé B, en utilisant les protocoles cryptographiques actuels. L'entreprise A commence par *chiffrer* ses documents. Les documents dit *chiffrés* sont sécurisés. L'entreprise A envoie ensuite les documents chiffrés à l'entreprise B. Cette dernière pourra les déchiffrer, *c-à-d* les rendre clair et donc non sécurisés, afin d'appliquer les traitements du service proposé. Enfin, l'entreprise B chiffre le résultat obtenu à la fin des traitements pour l'envoyer à l'entreprise A. L'entreprise A peut alors déchiffrer le document reçu pour récupérer le résultat en clair.

5. Couches Sockets Sécurisées, en français. *Socket* est une interface réseau.
6. Sécurité de la couche de transport, en français.
7. Code d'authentification de messages, en français.

En premier lieu, les données doivent être sécurisées durant les transferts entre l'utilisateur (possesseur des données), *p. ex.* l'entreprise A, et le destinataire (serveur distant), *p. ex.* l'entreprise B. Pour cela, un canal sécurisé peut être employé afin d'envoyer les données sur le serveur. Un canal sécurisé assure la confidentialité des données qui sont chiffrées automatiquement avant envoi. Par exemple, les sites bancaires utilisent un HTTPS (HyperText Transfer Protocol Secure [8]). Ce protocole correspond à l'association de HTTP et SSL / TLS [FKK11]. Le protocole HTTP crée une communication entre un client et un serveur par un canal. Ce canal est sécurisé par le protocole SSL / TLS qui assure la confidentialité, l'intégrité et l'authentification des données. Dans le cas où un canal est non sécurisé, un utilisateur doit lui-même chiffrer ses données afin de les protéger lors du transfert sur ce canal.

En second lieu, l'utilisateur (*p. ex.* l'entreprise A) doit faire confiance au tiers (*p. ex.* l'entreprise B) quant à l'accès et l'exploitation des données présentes sur ses serveurs. Deux cas sont alors possibles : soit le tiers est considéré comme digne de confiance, soit il ne l'est pas. Dans le premier cas, le tiers est considéré comme résistant aux attaques extérieures c'est-à-dire son infrastructure est sécurisée et, par conséquent, les données de ses utilisateurs aussi. De plus, si le tiers stocke les données de ses utilisateurs en clair, celles-ci peuvent être manipulées à distance mais ne doivent en aucun cas être diffusées. Dans le second cas, la confiance repose seulement dans le procédé de protection des données qui sont stockées sur le serveur distant. Bien sécurisée, une donnée stockée sur un serveur est inexploitable par un tiers.

1.1.2 Cryptographie

La *cryptographie* joue un rôle important dans la sécurisation des données. La cryptographie regroupe l'ensemble des principes, des méthodes et des techniques qui permettent le chiffrement et le déchiffrement des données. Elle vise à préserver la confidentialité, l'authentification, l'intégrité et la non-répudiation des données. Une donnée *en clair* est une donnée intelligible. Une donnée est dite *chiffrée* (resp. *déchiffrée*) si elle est passée par une fonction de chiffrement (resp. déchiffrement).

Les premières méthodes de chiffrement remontent à l'Antiquité et étaient principalement des substitutions monoalphabétique ou polyalphabétique. Parmi ces méthodes, le chiffrement de César est la plus connue. Cette méthode consiste à décaler chaque lettre de k cases (k étant la clé). Les méthodes de chiffrement créées étaient relativement simples car réalisées manuellement. Depuis, la cryptographie n'a cessé d'évoluer et ce, en même temps que l'évolution de la technologie. En effet, l'automatisation du chiffrement arrive en 1919 avec la création d'une machine à chiffrer électromécanique du nom de *Enigma*. C'est ensuite dans les années 1970, avec l'utilisation des ordinateurs, que la cryptographie fit une grande avancée notamment avec l'arrivée de la cryptographie dite *asymétrique*. Depuis, des nouveaux cryptosystèmes sont créés constamment afin de garantir la sécurisation des données personnelles des utilisateurs qui regroupent entre autres les données bancaires, les messages électroniques et les conversations par téléphone ou en ligne.

Les méthodes de chiffrement sont indispensables à ce jour pour protéger les données numériques dont particulièrement celles stockées sur des serveurs distants. Cependant, les systèmes de chiffrement utilisés actuellement ne permettent pas l'exportation d'un calcul sur l'un de ces serveurs tout en conservant la propriété de confidentialité des données. En effet, actuellement, bien que la donnée est envoyée chiffrée, le serveur distant doit pouvoir posséder la donnée en clair de l'utilisateur afin de pouvoir modifier et/ou faire un calcul. Par exemple, dans la Figure 1.1, une entreprise B possède un algorithme permettant des

8. Protocole de transfert hypertexte sécurisé, en français

calculs statistiques sur des données clients. Cette entreprise veut conserver son algorithme secret face aux autres entreprises. Une entreprise A a des données clients chiffrées et aimerait bénéficier de l'algorithme proposé par l'entreprise B. De nos jours et dans ce cas de figure, l'entreprise B doit posséder les données en clair de l'entreprise A. Malheureusement, les données clients de l'entreprise A sont censées être sécurisées : par conséquent, si l'entreprise A bénéficie du service de l'entreprise B, les données clients perdent leur propriété de confidentialité.

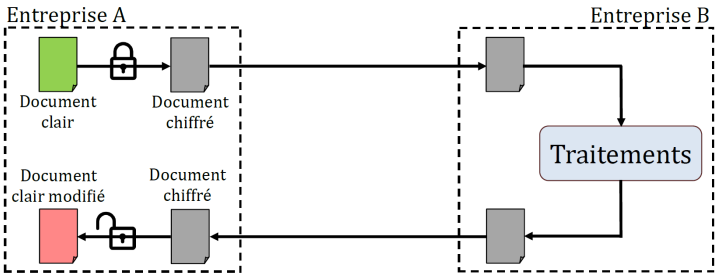


FIGURE 1.2 – Modification d'un document par une entreprise tierce avec utilisation du chiffrement homomorphe.

1.1.3 Chiffrement homomorphe

Un système de chiffrement voit le jour en 2009 : ce chiffrement dit *complètement homomorphe* va permettent de conserver la propriété de confidentialité dans l'exemple détaillé ci-dessus. En d'autres termes, le chiffrement homomorphe permet la manipulation de données chiffrées. De ce fait, des données chiffrées peuvent être stockées et manipulées par un tiers non fiable sur un serveur distant.

La Figure 1.2 reprend la Figure 1.1 dans le cas de l'utilisation de chiffrement homomorphe. La différence notable entre les deux figures est l'absence de documents en clair au sein de l'entreprise B. L'entreprise A peut envoyer les données clients chiffrées et l'entreprise B peut effectuer son calcul statistique directement sur les données chiffrées. Le résultat chiffré est retourné à l'entreprise A qui peut déchiffrer et obtenir le résultat en clair. Dans cette situation, l'entreprise B n'a jamais eu accès aux données en clair, ni aux résultats produits.

Depuis le premier algorithme ou *schéma* de chiffrement complètement homomorphe créé en 2009, le domaine de la cryptographie homomorphe se développe fortement. Cependant, aucun schéma homomorphe n'est utilisé pour des applications demandant des calculs complexes. La cryptographie homomorphe souffre notamment de schémas trop coûteux en temps de calcul et en ressources mémoire. Par exemple, pour le schéma FV [FV12], la génération de clés et le chiffrement d'un bit prend en tout 0.044 seconde. Le résultat chiffré (resp. les clés) a alors une taille de 6 153 octets (resp. 20.4 Ko). En comparaison, un algorithme de cryptographie classique génère la clé de chiffrement et chiffre en moins de 0.001 seconde pour retourner un chiffré (resp. une clé) de taille 32 octets (resp. 16 octets). Ces résultats ont été obtenus en calculant le temps [9] de génération de clé et de chiffrement de AES [DR99] en mode CBC avec OpenSSL 1.0.1t (pour l'algorithme de cryptographie classique) et de FV [FV12] avec Cingulata [Tea17] (pour l'algorithme homomorphe). La taille des chiffrées et clés sont obtenues en regardant le poids du fichier les contenant. La configuration logicielle et matérielle de la machine ayant produit ces résultats est présentée

9. Ce temps est récupéré avec la ligne de commande *time*, en additionnant les temps *user* et *sys c-à-d* la durée pour laquelle le processus à sollicité le processeur de manière effective, en mode noyau et mode utilisateur.

dans la Table 4.1 du chapitre 4. En plus d'une complexité et consommation mémoire importante, la grande quantité de schémas homomorphes créés ne facilite pas leur comparaison entre eux. À chaque création de schéma, seuls quelques jeux de paramètres sont testés afin d'analyser les temps de calculs. Le nombre de jeux de paramètres testés est minime face à la quantité de jeux possibles. En effet, un schéma de chiffrement homomorphe possède en moyenne 3 à 5 paramètres pouvant varier. Par conséquent, une analyse complète des jeux de paramètres n'est pas réalisée. Autrement dit, les quelques jeux de paramètres testés ne rendent pas forcément compte des meilleures configurations du schéma.

1.2 Contributions de ce manuscrit

Les contributions apportées dans ce manuscrit permettent *in fine* une analyse complète des jeux de paramètres de plusieurs schémas de chiffrement homomorphe. Les analyses estiment la complexité algorithmique et de consommation mémoire des schémas. En analysant tous les schémas, ceux-ci peuvent être comparés voire mis en compétition pour déterminer le schéma le plus intéressant pour une application donnée. Les contributions présentées dans ce manuscrit sont les suivantes :

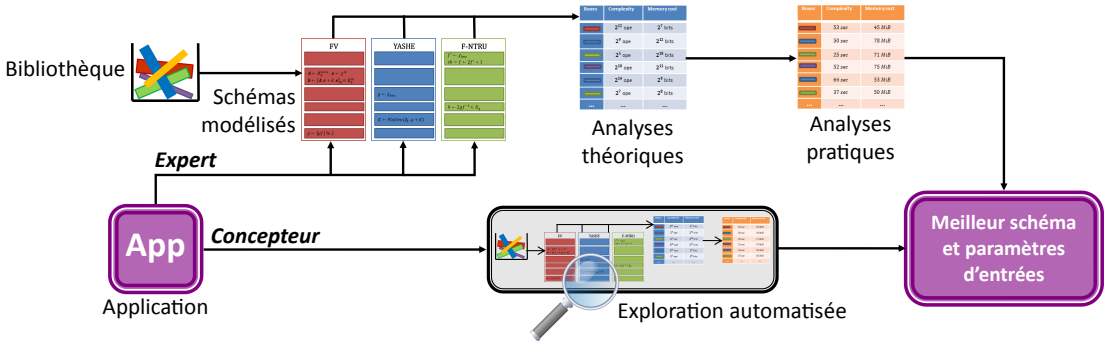


FIGURE 1.3 – Différentes étapes de PAnTHErS.

- Tout d'abord, une étape de modélisation a été créée afin de modéliser tout schéma sous forme d'enchaînement d'algorithmes. La modélisation imaginée est basé sur les schémas homomorphes existants dans le but de créer les algorithmes. Ces algorithmes possèdent peu d'instructions face au schéma tout entier. De plus, les algorithmes peuvent être ré-utilisés dans plusieurs modélisations de schémas. Cette étape préliminaire est indispensable pour analyser, par la suite, des schémas complet.
- Une fois qu'un schéma est modélisé, une ou plusieurs analyses peuvent être réalisées. À partir de la méthode de modélisation, l'analyse des schémas a été conçue afin d'être rapide d'utilisation dès qu'un schéma est modélisé. Une analyse va retourner le nombre d'opérations effectuées par le schéma et/ou le nombre d'entiers stockés pendant l'exécution du schéma. Le nombre d'opérations est la complexité algorithmique et le nombre d'entiers stockés est la consommation mémoire : ce sont les analyses théoriques d'un schéma de chiffrement homomorphe. L'analyse de schéma ne requiert pas l'exécution du schéma en lui-même pour retourner les analyses théoriques du schéma. Les résultats retournés de l'analyse sont sous forme de graphes et montrent la tendance de la complexité et/ou de la consommation mémoire des schémas.

- Le caractère théorique des analyses précédentes donne les tendances d'un schéma mais ne reflète pas son application concrète. Une méthode de calibration a donc été construite pour effectuer une analyse pratique de schémas. Cette calibration part de l'analyse théorique et de quelques exécutions du schéma afin de retourner, pour chaque jeu de paramètre donné, une complexité algorithmique évaluée en secondes et une consommation mémoire exprimée en mébioctets.
- Pour effectuer les analyses, quelles soient théoriques ou pratiques, une connaissance des schémas de chiffrement homomorphes est requise. En effet, il faut être capable de savoir comment faire varier les paramètres d'entrée afin de pouvoir lancer les analyses. Un programme d'exploration a donc été élaboré pour les non-experts en chiffrement homomorphe. Ce programme oriente le choix d'un schéma ainsi que d'un jeu de paramètre d'entrée pour l'employer sur une application donnée.
- Dans l'objectif d'automatiser les méthodes et programme décrits ci-dessus, l'outil appelé PAnTHErS pour *Prototyping and Analysis Tool for Homomorphic Encryption Schemes* [10] a été conçu. PAnTHErS peut être utilisé pour l'analyse d'un schéma mais également pour l'analyse d'une application. La Figure 1.3 montre les différentes étapes de PAnTHErS pour analyser une application du point de vue d'un expert ou d'un concepteur.
  D'un côté, un expert en chiffrement homomorphe choisit des schémas modélisés dans PAnTHErS. L'application peut alors être analysée théoriquement en fonction des différents schémas et paramètres d'entrée choisis. Puis, les analyses théoriques sont converties en données pratiques et sont retournées sous forme de graphes. Ces graphes peuvent être interprétés par l'expert afin de déterminer quel schéma est le plus intéressant en matière de complexité et coût mémoire et avec quel jeu de paramètres d'entrée.
  D'un autre côté, un concepteur d'applications *c-à-d* un non-expert en chiffrement homomorphe va également pouvoir utiliser PAnTHErS grâce à l'algorithme d'exploration des paramètres qui a été intégré. Cet algorithme, à partir d'une application, retourne directement le nom du schéma le plus intéressant avec le jeu de paramètre associé.

1.3 Structure de ce manuscrit

Le manuscrit est organisé de la façon suivante.

Le **chapitre 2** présente l'état de l'art. Tout d'abord, les aspects généraux de la cryptographie sont décrits. Une limitation des algorithmes de cryptographie actuels est mise en avant. Ensuite, le chiffrement homomorphe, solution à cette problématique, est alors présenté. Une première section introduit les évolutions du chiffrement homomorphe. Une seconde section donne les théories mathématiques utiles afin de réaliser des algorithmes homomorphes. Enfin, une dernière section liste les limitations de ce chiffrement prometteur pour le futur.

Le **chapitre 3** présente la méthode utilisée afin de modéliser tout type de schéma de chiffrement homomorphe. Cette méthode permet le stockage d'algorithmes afin qu'ils puissent être réutilisés dans d'autres modélisations. De manière plus générale, cette méthode est applicable sur tout algorithme mathématique. Ce chapitre explique également comment réaliser une analyse théorique à partir des modèles de schémas. Deux types

10. Outil d'analyse et de prototypage pour des schémas de chiffrement homomorphe, en français. La propriété de prototypage n'est pas encore incluse dans PAnTHErS mais fait partie des perspectives de l'outil.

d'analyses sont présentées : la première correspond à l'analyse de la complexité algorithmique et la seconde à l'analyse de la consommation mémoire. Enfin, la dernière section est une discussion autour d'autres analyses théoriques possibles comme par exemple l'analyse de l'expansion de bruit.

Le **chapitre 4** présente une méthode de calibration. Cette méthode correspond à la conversion des analyses théoriques en analyse pratique en se basant sur des exécutions réelles. Ce chapitre explique la p-calibration représentant le cas général et illustre les résultats de son application pour un p allant de 1 à 4.

Le **chapitre 5** présente la méthode d'exploration des paramètres d'entrée, permettant à un concepteur d'utiliser PAnTHErS pour obtenir le schéma ainsi que les jeux de paramètres fournissant des résultats les plus proches possible de l'optimal, favorisant selon ses besoins le temps d'exécution le plus court, le coût mémoire le plus faible ou un compromis entre ces deux facteurs. Ce chapitre expose les résultats de l'exploration appliquée à quatre applications. Enfin, ce chapitre présente l'outil PAnTHErS à travers trois profils d'utilisateurs : concepteur, expert et développeur expert.

Le **chapitre 6** conclut ce manuscrit. Ce chapitre récapitule les contributions apportées afin de répondre à la problématique de ce manuscrit. Enfin, ce chapitre présente les perspectives envisageables pour l'outil PAnTHErS.

Chapitre 2

Etat de l'art

Sommaire

La cryptographie moderne vise à sécuriser les données numériques des utilisateurs sur leurs appareils électroniques, sur internet, sur des serveurs distants mais également pendant leurs transferts (d'un espace de stockage à un autre). Pour cela, la cryptographie garantie les principes fondamentaux suivants : l'authentification, la confidentialité, la non-répudiation et l'intégrité comme énoncé dans le chapitre 1.

Actuellement, les utilisateurs d'internet utilisent des serveurs distants afin de stocker une partie de leurs données numériques personnelles. Ces serveurs distants sont appelés des « nuages » ou *clouds*. De plus, ces mêmes utilisateurs peuvent bénéficier des ressources calculatoires disponibles sur les clouds : dans ce cas, ils utilisent le *cloud computing* ou « l'informatique en nuage » [1]. Afin de protéger la vie privée de l'utilisateur, chaque donnée des utilisateurs est rendue incompréhensible, *c-à-d* chiffrée, pendant son transfert sur le cloud et lors de son stockage dans le cloud. Or, aujourd'hui, si un utilisateur veut exploiter les ressources d'un service cloud computing, chaque donnée envoyée sur le serveur distant doit être compréhensible, *c-à-d* non chiffré, afin d'être manipulé par le service voulu. Le serveur possède donc les données personnelles des utilisateurs en clair : la propriété fondamentale de confidentialité n'est alors plus assurée.

À l'inverse, un système de chiffrement asymétrique, qualifié d'homomorphe, permet d'assurer la confidentialité pour un service de cloud computing. Avec un tel chiffrement, le message n'a pas besoin d'être déchiffré sur le cloud afin d'être modifié ou utilisé. Le chiffrement homomorphe permettrait donc de sécuriser entièrement les données en ligne. Ce chiffrement ne peut cependant pas être utilisé actuellement car possède encore de nombreuses limitations dont notamment un coût trop important en temps de calcul et en ressource mémoire.

Ce chapitre présente tout d'abord l'évolution de la cryptographie et son utilisation au quotidien. Ensuite, ce chapitre adresse en détail le chiffrement homomorphe. Il rappelle les notions mathématiques nécessaires afin de présenter ensuite, concrètement, l'historique et les techniques développés autour du chiffrement homomorphe. Enfin, les limitations des algorithmes de chiffrement homomorphe sont listés.

2.1 Cryptographie

Dans l'Antiquité, la cryptographie visait à assurer la confidentialité d'un message. Pour cela, des méthodes de chiffrements comme des substitutions monoalphabétiques (*p. ex.* chiffrement de César, voir chapitre 1) étaient utilisées. Actuellement, en plus de la confidentialité, la cryptographie moderne vise à préserver l'authentification, la non-répudiation et l'intégrité des messages (voir chapitre 1). Pour cela, il est possible de sécuriser les informations avec soit un système symétrique, soit un système asymétrique ou bien encore une combinaison des deux.

Cette section présente les définitions mathématiques des termes techniques touchant la cryptographie. Puis, les algorithmes symétriques et asymétriques sont expliqués indépendamment. La section explique comment la cryptographie moderne est utilisée au quotidien et présente certaines limitations dont notamment l'incapacité des algorithmes actuels à effectuer des traitements sur des données chiffrées.

2.1.1 Définitions

La *cryptographie* vise à préserver la confidentialité, l'intégrité, la non-répudiation et l'authentification des données (voir chapitre 1). Afin de garantir ces quatre principes fondamentaux, les données sont sécurisées grâce à des systèmes de chiffrement. Une donnée

1. https://www.futura-sciences.com/tech/definitions/informatique-cloud-computing-11573/

intelligible est appelé *donnée en clair* et une donnée en clair passant par une fonction de chiffrement est appelé *donnée chiffrée*. Une donnée est dite *déchiffrée* si la donnée est passée par une fonction de déchiffrement.

Un système de chiffrement ou *cryptosystème* est composé de :

- $\mathcal{P}$, l'ensemble des messages clairs.
- $\mathcal{C}$, l'ensemble des messages chiffrés.
- $\mathcal{K}$, l'ensemble de clés.
- $\mathcal{E}$, l'ensemble des fonctions de chiffrement $E_k : \mathcal{P} \longrightarrow \mathcal{C}$ avec $k \in \mathcal{K}$.
- $\mathcal{D}$, l'ensemble des fonctions de déchiffrement $D_k : \mathcal{C} \longrightarrow \mathcal{P}$ avec $k \in \mathcal{K}$. Pour chaque D_k, il existe $k' \in \mathcal{K}$ tel quel pour tout $p \in \mathcal{P}$, $D_k(E_{k'}(p)) = p$.

À l'opposé de la cryptographie, la *cryptanalyse*, quant à elle, vise à analyser un système de chiffrement en vue de l'attaquer. Grâce à des techniques de cryptanalyse, un attaquant peut retrouver une donnée en clair à partir d'une donnée chiffrée sans posséder la clé de déchiffrement. Un attaquant peut également tenter de trouver directement la clé de déchiffrement en faisant une attaque par force brute *c-à-d* en essayant toutes les clés possibles jusqu'à trouver la bonne. La *cryptologie* est la science qui regroupe la cryptographie et la cryptanalyse.

En cryptographie, deux types de systèmes de chiffrement existent : chiffrement symétrique et chiffrement asymétrique. Un cryptosystème *symétrique* ou *à clé privée* possède une unique clé servant au chiffrement comme au déchiffrement. C'est-à-dire, dans un système de chiffrement symétrique, pour toute clé $k \in \mathcal{K}$ et tout message clair $p \in \mathcal{P}$, $D_k(E_k(p)) = p$. À l'inverse, un cryptosystème *asymétrique* ou *à clé publique* possède deux clés : une clé publique pour le chiffrement et une clé privée pour le déchiffrement.

Ces deux types de chiffrement sont complémentaires dans l'objectif de garantir les quatre principes fondamentaux de la cryptographie. D'autres méthodes, non détaillées dans ce manuscrit, sont également indispensables pour garantir les principes fondamentaux telles que les fonctions de hachage, les signatures numériques ou encore les infrastructures à clé publique (PKI pour *Public Key Infrastructure*). Les fonctions de hachages sont appliquées sur les messages et assurent leur intégrité. Ce sont des algorithmes à sens unique permettant d'obtenir une empreinte du message. Le NIST recommande notamment l'utilisation des fonctions de hachages suivantes : SHA-1, SHA-256, SHA-512 et SHA-3 [FIP12]. Les signatures numériques [FIP13] assurent l'authentification de l'auteur d'un message. Pour cela, les signatures sont effectuées à l'aide d'une fonction de hachage, d'un algorithme de chiffrement asymétrique et de la clé privée de l'auteur. Les infrastructures a clé publique regroupent l'ensemble des procédures permettant la création et la gestion des clés publiques dans un système. Par exemple, le logiciel libre EJBCA [Sol] permet de mettre en place ce type d'infrastructure au sein d'une entreprise ou d'une organisation.

2.1.2 Cryptosystème symétrique

Dans un cryptosystème symétrique, une même clé est employée pour le chiffrement et le déchiffrement d'un message. Cette clé est donc *privée* et ne doit en aucun cas être rendue publique. Par conséquent, lorsque deux entités, nommées Alice et Bob, veulent s'échanger un message, ils doivent auparavant s'échanger la clé de façon sécurisée *p. ex.* lors d'une rencontre en présentiel.

Problème : Alice veut envoyer une lettre à Bob. Elle ne fait pas confiance au facteur qui peut ouvrir la lettre avant de la poster. Comment échanger des lettres sans que le facteur puisse accéder au contenu de la lettre ?

Solution : Une méthode symétrique pour résoudre ce problème est l'achat d'une boîte avec un cadenas. Au préalable, Alice prend une boîte et un cadenas et donne un exemplaire de la clé du cadenas à Bob en toute discrétion lors d'une rencontre. Puis, Alice et Bob peuvent s'échanger des messages : l'un comme l'autre peut mettre un message dans la boîte et refermer celle-ci à l'aide du cadenas. Chacun possédant une clé, ils peuvent tous deux ouvrir le cadenas et découvrir le message que la boîte contient. Le facteur n'a pas la clé et ne peut donc pas obtenir le message contenu dans la boîte.

Dans cet exemple, le cadenas représente la fonction de chiffrement et la clé du cadenas est la clé privée. Le message clair est le message mis dans la boîte et le message chiffré est la boîte contenant le message.

La cryptographie symétrique est le type de cryptographie utilisé depuis l'Antiquité. Les premiers algorithmes de substitutions monoalphabétique ou polyalphabétique consistant en des décalages de lettres sont symétriques. Dans un chiffrement par substitution monoalphabétique, chaque lettre de l'alphabet est chiffrée de la même façon *p. ex.* "A" est toujours chiffré en "D". Le chiffrement d'une lettre peut différer dans un chiffrement par substitution polyalphabétique *p. ex.* "AA" peut être chiffré en "CD".

Commercialisée et vendue à partir de 1923, la machine à chiffrer électromécanique Enigma [Tur40] est également symétrique et consiste en une substitution polyalphabétique. Enigma fut utilisée principalement durant la Seconde Guerre Mondiale et fut cassée par Alan Turing pendant la guerre [DPT].

Les cryptosystèmes symétriques actuels sont soit par bloc, soit par flot. Un chiffrement symétrique par bloc consiste à décomposer le message à chiffrer en bloc de b bits. Chaque bloc est ensuite chiffré en suivant un mode d'opérations *c-à-d* suivant une procédure de traitement des blocs. Le mode d'opérations le plus simple est de chiffrer chaque bloc indépendamment des autres et de concaténer les blocs résultats : cette méthode est appelée ECB pour *Electronic Code Book*. Cependant, d'autres modes sont utilisés en pratique dans le but d'augmenter le niveau de sécurité *p. ex.* CBC pour *Cipher Block Chaining* [EMST78]. Un chiffrement symétrique par flot nécessite d'un générateur de nombre pseudo-aléatoire (PRNG, pour *Pseudo-Random Number Generator*). Un PRNG est utilisé pour générer une suite de nombre pseudo-aléatoire. La fonction *OU exclusif* (ou XOR, pour *eXclusive OR*) est appliquée entre cette suite et les bits du message à chiffrer. Un XOR, noté $\oplus$, est un opérateur logique tel que : $0 \oplus 0 = 0$, $1 \oplus 0 = 1$, $0 \oplus 1 = 1$ et $1 \oplus 1 = 0$.

Le *One Time Pad* (OTP) est un chiffrement symétrique par flot considéré comme cryptographiquement sûr. La clé symétrique consiste en une suite de nombres aléatoires aussi grande que le message à chiffrer. Le chiffrement est la fonction XOR appliquée entre la clé symétrique et le message. Bien que ce cryptosystème soit sûr, l'échange de clé revient au même problème que d'échanger le message en clair, vu que la clé est aussi longue que le message.

Le cryptosystème symétrique le plus utilisée de nos jours est l'AES [DR99] pour *Advanced Encryption Standard* qui est un système de chiffrement par bloc. L'AES a la structure d'un réseau de substitution-permutation. Un bloc de message de clair passant par ce type de réseau va passer dans un enchaînement de transformations, chacune correspondant soit à une substitution, soit à une permutation. Par conséquent, l'AES est une succession de plusieurs rondes, chacune composée de fonctions de substitution et de permutation. Suite à ces transformations, le message obtenu est XORé avec la clé de ronde en fin de ronde.

2.1.3 Cryptosystème asymétrique

Dans un cryptosystème asymétrique, la clé utilisée pour le chiffrement diffère de celle utilisée pour le déchiffrement d'un message. La clé de chiffrement est dite *publique* et est

visible par tous. La clé de déchiffrement est dite *privée* et seul le créateur de la clé doit posséder cette clé. Si Alice génère une paire de clé publique/privée, alors elle seule doit posséder sa clé privée (et la conserver secrète). Dans ce cas de figure, il n'y a pas d'échange préalable de clés à effectuer : la clé publique peut être diffusée directement en ligne. Par exemple, si Bob veut envoyer un message à Alice, il trouve la clé publique d'Alice en ligne, l'utilise pour chiffrer son message et envoie le résultat chiffré à Alice.

Le problème 2.1.2, énoncé dans la section précédente, peut être résolu de façon asymétrique. Bob donne un cadenas ouvert à Alice et conserve précieusement la clé du cadenas. Alice, de son côté, prend une boîte, met son message dedans et referme la boîte avec le cadenas. Elle envoie la boîte par la poste. Le facteur ne peut pas ouvrir la boîte car il ne possède pas la clé du cadenas. Bob, quant à lui, la possède et peut donc ouvrir la boîte et récupérer le message d'Alice.

Dans cet exemple, le cadenas représente la clé publique et la clé du cadenas est la clé privée. Le message clair est le message mis dans la boîte et le message chiffré est la boîte contenant le message. La fonction de chiffrement est symbolisée par la fermeture de la boîte avec le cadenas. La fonction de déchiffrement est l'ouverture du cadenas avec la clé.

La notion de cryptosystème asymétrique apparaît seulement qu'en 1976 par Diffie et Hellman [DH76]. Cependant, aucun cryptosystème n'est fourni dans leur publication. C'est en 1978 que Rivest, Shamir et Adleman [RSA78] ont introduit l'un des premiers cryptosystèmes asymétriques appelé *RSA*, d'après les initiales des noms de ses créateurs.

L'algorithme RSA est défini par les fonctions suivantes :

Génération de clés. Soit p et q deux grands nombres premiers. Le couple (p, q) correspond à la clé privée. Soit $n = p \times q$. L'indice d'Euler de n donne $\varphi(n) = (p-1)(q-1)$. Soit e un nombre premier avec $\varphi(n)$ tel que $e < \varphi(n)$. Le couple (n, e) est la clé publique.

Chiffrement. Soit $m < n$ le message clair (entier) et (n, e) la clé publique. Le chiffrement de m est le suivant : $c = m^e \mod n$.

Déchiffrement. Soit $c < n$ le message chiffré (entier) et (p, q) la clé privée. Le déchiffrement nécessite une étape préliminaire qui consiste à calculer d, l'inverse de e modulo $\varphi(n)$. Pour déchiffrer le message c, il suffit de calculer $m' = c^d \mod n$.

Le résultat m' est bien équivalent au message m de départ. En effet,

$$m' = c^d \mod n = (m^e)^d \mod n = m^{ed} \mod n.$$

Or, $ed = 1 \mod (p-1)(q-1)$ et donc d'après le petit théorème de Fermat,

$$m^{ed} \mod n = m \mod n.$$

De plus, $m < n$ donc $m \mod n = m$.

La sécurité de l'algorithme RSA repose sur la difficulté de factoriser des grands nombres. Autrement dit, à partir de la clé publique (n, e), un attaquant ne doit pas être capable de factoriser n en $p \times q$. Si un attaquant factorise n, alors il peut retrouver facilement la clé privée (p, q) et d. En possédant (n, e), p et q, il est facile de calculer d qui est tel que $de = 1 \mod (p-1)(q-1)$. L'attaquant peut donc déchiffrer tout texte chiffré avec la clé (n, e).

L'algorithme RSA [RSA78] est le cryptosystème asymétrique le plus utilisé de nos jours.

2.1.4 Usage de la cryptographie

De manière générale, les cryptosystèmes asymétriques sont plus lents que les symétriques. En revanche, la gestion des clés pour un cryptosystème asymétrique est plus facile : en effet, chaque correspondant doit posséder sa propre paire clé publique/privée. Dans un cryptosystème symétrique, chaque utilisateur doit posséder une clé privée pour chaque correspondant. De plus, l'envoi préliminaire de la clé privée (partagée entre les deux entités)

doit être impérativement fait à travers un canal de communication sécurisé. Ce problème n'est pas présent pour un cryptosystème asymétrique. En effet, la clé servant au chiffrement est publique.

Afin d'employer au mieux les deux types de cryptosystèmes et de profiter des avantages de chacun, il est préférable de chiffrer les messages avec un cryptosystème symétrique. La clé privée symétrique est ensuite chiffrée avec un cryptosystème asymétrique pour être envoyée au destinataire avec le message chiffré en symétrique.

Cette méthode d'utilisation de la cryptographie permet d'échanger des informations de manière sécurisée *p. ex.* entre un client et sa banque, entre un patient et son médecin, entre les membres d'une famille,... Il en est de même quand un utilisateur se connecte et utilise un site internet ou une application sur son smartphone : les données personnelles de l'utilisateur envoyées sur le serveur du site ou de l'application sont protégées car chiffrées. Les sites ou applications sensibles peuvent être, par exemple, d'ordre militaires, bancaires, conversationnelles, sociales, économiques ou médicales.

Pour toutes ces sites ou applications, les deux entités échangeant des informations se font mutuellement confiance : en effet, chacune va avoir accès aux données en clair. Les tierces personnes, autres que les deux entités concernées, ne pourront en aucun cas avoir accès aux informations déchiffrées. Lorsqu'un utilisateur discute avec un ami à travers, *p. ex.* l'application WhatsApp, les deux entités se connaissent et se font mutuellement confiance. De plus, sur WhatsApp, les données sont dites "protégées avec le chiffrement de bout en bout. Cela signifie que WhatsApp et les tiers ne peuvent pas les lire"[2]. Dans cet exemple, la sécurité repose exclusivement sur la capacité de chiffrement utilisé par WhatsApp. Cependant, la plupart du temps, la sécurité repose sur la confiance portée en l'application.

Prenons l'exemple de l'application Samsung Health[3] installée automatiquement sur tous les smartphones Samsung. Cette application permet entre autre de calculer les calories gagnées/perdues en fonction du sport réalisé et des repas consommés dans la journée. Pour cela, dès l'inscription, l'utilisateur doit rentrer certaines informations personnelles comme son genre, sa date de naissance, sa taille, son poids et son niveau d'activité. Ensuite, chaque jour, l'utilisateur peut renseigner quel sport il a pratiqué, pendant combien de temps ainsi que tous les aliments consommés dans la journée avec leur proportion. Ainsi, l'utilisateur peut voir le nombre de calories perdues (grâce au sport) et gagnées (suite aux repas).

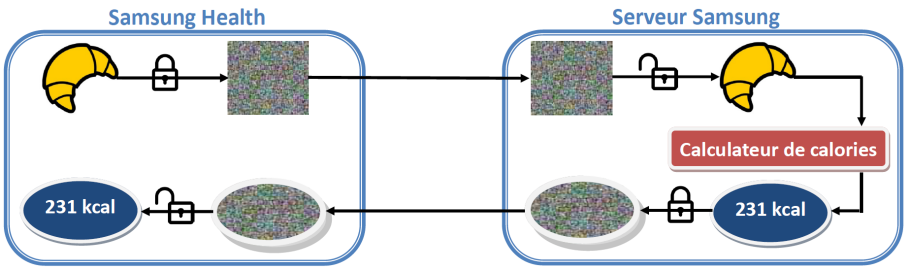


FIGURE 2.1 – Traitements des données lors d'une demande de calculs de calories sur Samsung Health.

La Figure 2.1 montre le cheminement des informations lors de l'utilisation de Samsung Health. Une fois les informations rentrées dans l'application (*p. ex.* un croissant sur la Figure 2.1), les données personnelles de l'utilisateur sont chiffrées et envoyées de manière

2. https://faq.whatsapp.com/en/android/28030015/?lang=fr
3. http://health.apps.samsung.com/

sécurisée au serveur de Samsung. Les données sont protégées mais Samsung doit avoir accès aux données en clair afin de réaliser le calcul de calories gagnées et/ou perdues. Autrement dit, bien que chiffrées durant le transfert jusqu'au serveur Samsung, les données doivent ensuite être déchiffrées sur le serveur afin que Samsung connaisse les activités effectuées et les aliments consommés. Samsung est alors capable de retourner les calories que l'utilisateur a gagnées (*p. ex.* 231 kcal pour un croissant sur la Figure 2.1) et perdues pendant la journée. Le résultat est retourné chiffré.

Dans cet exemple, les informations personnelles concernant la vie privée d'un utilisateur sont protégées pendant les transferts allant de l'utilisateur jusqu'au serveur. Mais ensuite, le serveur a entièrement accès aux données en clair afin de réaliser le service demandé par l'utilisateur. Dans ce cas de figure, l'utilisateur doit faire confiance à Samsung pour que celui-ci :

- protège les informations en clair des utilisateurs *c-à-d* qu'une fois utilisée, aucune information ne doit rester en clair sur les serveurs. De plus, Samsung doit être résistant aux attaques extérieures lors de la manipulation des données en clair.

- ne divulgue aucune information sur des utilisateurs *c-à-d* Samsung ne doit en aucun cas transmettre les informations en clair à d'autres entités.

Pour une meilleure sécurisation des données utilisateurs, dans le cas de Samsung Health, il faudrait que les données restent chiffrées pendant leur manipulation, autrement dit, que les données n'aient pas besoin d'être déchiffrées par Samsung pour permettre le calcul du nombre de calories gagnées et perdues.

Les cryptosystèmes utilisés actuellement, *p. ex.* RSA ou AES, ne permettent pas la manipulation des données chiffrées. Un nouveau type de chiffrement dit *homomorphe* a été inventé afin de pouvoir effectuer des opérations sur les données chiffrées. S'il était utilisé dans Samsung Health, le chiffrement homomorphe permettrait aux données utilisateurs de rester confidentielles. De plus, la sécurité reposerait seulement sur la difficulté à attaquer le cryptosystème homomorphe utilisé.

2.1.5 Cryptographie post-quantique

Comme les technologies évoluent, les cryptosystèmes doivent devenir de plus en plus résistants en matière de sécurité. Les recherches en sécurité tentent déjà de répondre à la question de sécurisation des données face aux futurs calculateurs ou ordinateurs quantiques. Bien que robustes à présent, les algorithmes utilisés, *p. ex.* RSA ou AES, n'ont pas la propriété post-quantique. En d'autres termes, l'apparition des ordinateurs quantiques permettrait de casser les algorithmes cryptographiques utilisés de nos jours.

Les ordinateurs quantiques utilisent des propriétés de la physique quantique permettant ainsi d'accélérer les calculs de problèmes mathématiques difficiles. En l'occurrence, l'algorithme RSA, basé sur la difficulté de factoriser des grands nombres, pourrait être cassé avec un ordinateur quantique en utilisant l'algorithme de Shor [Sho99]. Si l'algorithme RSA est cassé, toute la sécurité actuelle est alors remise en question.

Afin d'anticiper le développement des ordinateurs quantiques, le NIST[4] a lancé un appel aux cryptosystèmes post-quantique. L'objectif est de pouvoir trouver de nouveaux cryptosystèmes robustes aux ordinateurs quantiques qui deviendront les nouveaux standards de la cryptographie.

4. Site de l'appel aux cryptosystèmes post-quantique du NIST : https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals

2.2 Outils mathématiques

Créée il y a des milliers d'années, la cryptographie est aujourd'hui omniprésente pour protéger nos données numériques notamment dans les appareils et les puces électroniques. Pour cela, la cryptographie symétrique est généralement utilisée pour chiffrer les données et la cryptographie asymétrique pour chiffrer la clé symétrique. Bien que la cryptographie soit efficace dans notre société, elle présente quelques limitations quant à la modification des données par des tiers : ces derniers doivent posséder les données clients en clair avant de les modifier. Ceci pose un problème de confidentialité des données.

Le chiffrement homomorphe est une solution répondant à cette problématique. Un algorithme homomorphe permet d'assurer la confidentialité des données *p. ex.* pour un service cloud computing. En d'autres termes, l'un des principaux enjeux de la cryptographie homomorphe est de pouvoir déléguer nos calculs sur des serveurs distants sans perte de confidentialité des données. De plus, les algorithmes homomorphes sont basés sur des problèmes difficiles, non cassables par un ordinateur quantique. En d'autres termes, les algorithmes homomorphes possèdent la propriété post-quantique et résisteront donc aux futurs ordinateurs quantiques.

Avant de présenter ce chiffrement particulier, cette section fait un rappel sur les distributions de probabilités et les réseaux euclidiens avant de présenter en détail des problèmes sur lesquels reposent les algorithmes de chiffrement homomorphe. De plus, cette section présente un chiffrement utilisé pour la création de certains algorithmes homomorphes.

2.2.1 Notations

Dans la suite du manuscrit, les notations suivantes sont utilisées :

• $\mathbb{R}, \mathbb{Z}, \mathbb{N}$ représentent respectivement l'ensemble des réels, des entiers et des entiers naturels.

• Les vecteurs sont représentés par une minuscule en gras *p. ex.* $\boldsymbol{v}$.

• Les matrices sont représentées par une majuscule en gras *p. ex.* $\boldsymbol{M}$.

• Soit $a \in \mathbb{R}$, $|a|$ est la valeur absolue de a.

• La norme de $\boldsymbol{v} = (v_1, v_2, ..., v_n) \in \mathbb{R}^n$ notée $||\boldsymbol{v}||$ fait référence à la norme Euclidienne (ou norme L^2) dont la définition est $||\boldsymbol{v}|| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$.

• L'ensemble $\mathbb{Z}/q\mathbb{Z}$ est noté plus simplement de la façon suivante : $\mathbb{Z}_q$.

• Le produit scalaire de deux vecteurs $\boldsymbol{a}$ et $\boldsymbol{b}$ est noté $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$.

• L'arrondi a est symbolisé par $\lfloor a \rceil$. L'arrondi supérieur (resp. inférieur) de a est $\lceil a \rceil$ (resp. $\lfloor a \rfloor$).

• Le modulo de a par b est noté : $a \mod b$. Afin d'alléger certaines équations, nous utiliserons également la notation $[a]_b$.

2.2.2 Réseaux euclidiens et problèmes NP-complets

Les réseaux euclidiens ont permis la création du tout premier algorithme de chiffrement homomorphe. Depuis, une grande majorité des algorithmes homomorphes créés sont basés sur un problème sur les réseaux. Cette sous-section donne les définitions de réseaux et problème NP-complet avant de présenter les problèmes basés sur les réseaux.

Définition 1 (Réseau) *Soit n un entier positif. Un réseau L de $\mathbb{R}^n$ est un sous-groupe discret de l'espace euclidien $\mathbb{R}^n$. Le réseau L est isomorphe à $\mathbb{Z}^m$ avec $m < n$ en tant que groupe abélien : c'est-à-dire qu'il existe des vecteurs linéairement indépendants $\boldsymbol{b_0}, ..., \boldsymbol{b_{m-1}} \in L$*

tels que $L = \sum_{i=0}^{m-1} \mathbb{Z}\boldsymbol{b_i}$. *Les vecteurs* $\boldsymbol{b_0}, ..., \boldsymbol{b_m - 1}$ *forment une base de* L. *On appelle* m *le rang de* L.

Définition 2 (Problème NP) *Un problème est dans la classe NP (nondeterministic polynomial time, en anglais - "temps polynomial non déterministe", en français) s'il est possible de vérifier en temps polynomial qu'une solution proposée est solution du problème.*

Définition 3 (Problème NP-complet) *Un problème NP-complet est dans la classe NP. De plus, un problème NP-complet est considéré au moins aussi difficile que tous les autres problèmes de la classe NP.*

Les réseaux sont la base de plusieurs problèmes NP-complet dont le plus connu est sûrement le *Shortest Vector Problem* définit ci-dessous.

Définition 4 (*Shortest Vector Problem* (SVP)) *Soit* L *un réseau. Le Shortest Vector Problem ou le problème du plus court vecteur consiste à trouver un plus court vecteur du réseau qui soit différent du vecteur nul. Autrement dit, le SVP cherche à trouver* $\boldsymbol{v} \in L \setminus \{0\}$ *tel que* $\lambda(L) = ||\boldsymbol{v}|| = min\{||\boldsymbol{a}|| \text{ tel que } \boldsymbol{a} \in L \setminus \{0\}\}$.

Défini dans [Ajt98], le problème SVP est prouvé NP-complet par les auteurs. Une version décisionnelle de ce problème existe et est décrit ci-dessous.

Définition 5 ($GapSVP_\gamma$) *Soit* L *un réseau de dimension* n *et soit* $\gamma \geq 1$ *un réel. Le problème* $GapSVP_\gamma$ [5] *détermine si* $\lambda(L) \leq 1$ *ou* $\lambda(L) \geq \gamma$.

L'objectif du problème SVP (Définition 5) est de trouver le vecteur le plus proche du vecteur **0** dans le réseau. Le problème suivant, appelé *Closest Vector Problem*, vise à trouver le vecteur le plus proche d'un vecteur fixé $\boldsymbol{t} \in \mathbb{R}^n$, avec n la dimension du réseau.

Définition 6 (*Closest Vector Problem* (CVP)) *Soit* L *un réseau de dimension* n *et* $\boldsymbol{t} \in \mathbb{R}^n$ *un vecteur. Le Closest Vector Problem ou le problème du vecteur le plus proche consiste à trouver un vecteur du réseau le plus proche de* $\boldsymbol{t}$. *Autrement dit, le CVP cherche à trouver* $\boldsymbol{v} \in L$ *tel que* $\lambda(L) = ||\boldsymbol{t} - \boldsymbol{v}|| = min\{||\boldsymbol{t} - \boldsymbol{a}|| \text{ tel que } \boldsymbol{a} \in L\}$.

Les problèmes énoncés vont permettre la création d'autres problèmes NP-complets sur lesquels va reposer la grande majorité des algorithmes de chiffrement homomorphe.

2.2.3 Distributions de probabilités

Les distributions sont généralement utilisées dans les problèmes sur les réseaux et les algorithmes de chiffrement homomorphe. À titre de rappel, cette sous-section donne les définitions de distributions suivantes : uniforme discrète, gaussienne, gaussienne discrète et bornée.

Définition 7 (Distribution uniforme discrète) *Soit* $\mathbb{R}$ *un ensemble fini. Une distribution sur* $\mathbb{R}$ *est dite uniforme si la probabilité de* $x \in \mathbb{R}$ *est* $\frac{1}{\#R}$ *avec* $\#R$ *représentant le cardinal de* R. *La distribution uniforme sur* $\mathbb{R}$ *est notée* $\mathcal{U}(\mathbb{R})$.

5. *Gap* veut dire "écart".

Définition 8 (Distribution normale ou gaussienne) *Une distribution de déviation σ et de centre c sur $\mathbb{R}$ est dite normale ou gaussienne si la probabilité de $x \in \mathbb{R}$ est $\frac{E_{\sigma,c}(x)}{\sigma\sqrt{2\pi}}$ avec $E_{\sigma,c}(x) = e^{\frac{-(x-c)^2}{2\sigma^2}}$.*
De plus, soit n un entier positif, si $\boldsymbol{x}$ et $\boldsymbol{c}$ sont des vecteurs de $\mathbb{R}^n$, alors la probabilité de $\boldsymbol{x}$ est $\frac{E_{\sigma,\boldsymbol{c}}(\boldsymbol{x})}{\sigma\sqrt{2\pi}}$ avec $E_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = e^{\frac{||\boldsymbol{x}-\boldsymbol{c}||^2}{2\sigma^2}}$.

Définition 9 (Distribution gaussienne discrète) *Une distribution gaussienne de déviation σ et de centre c est dite discrète sur $\mathbb{R}$ si la probabilité de $x \in \mathbb{R}$ est $\frac{E_{\sigma,c}(x)}{S_{\sigma,c}}$ avec $S_{\sigma,c} = \sum_{k\in\mathbb{R}} E_{\sigma,c}(k)$.*

Définition 10 (Distribution gaussienne discrète sur un réseau) *Soit n un entier positif. Une distribution gaussienne de déviation σ et de centre $\boldsymbol{c}$ est discrète sur un réseau $L \subseteq \mathbb{R}^n$ si la probabilité de $\boldsymbol{x} \in L$ est $\frac{E_{\sigma,\boldsymbol{c}}(\boldsymbol{x})}{S_{L,\sigma,\boldsymbol{c}}}$ avec $S_{L,\sigma,\boldsymbol{c}} = \sum_{k\in L} E_{\sigma,\boldsymbol{c}}(k)$.*

Définition 11 (Distribution bornée) *Une distribution sur L est dite bornée par B si la probabilité que $|\boldsymbol{x}| > B$ avec $x \in L$ est négligeable.*

Dans les problèmes sur les réseaux, des échantillons sont prélevés à partir de ces distributions. Les échantillons doivent alors être représentatifs de la distribution de probabilités par laquelle ils ont été produits. Dans [DG14], Dwarakanath et Galbraith ont fait une étude sur les implémentations efficaces d'échantillons. En chiffrement homomorphe, ces échantillons sont utilisés afin de rajouter de l'erreur dans un calcul, comme *p. ex.* dans le problème du *Learning With Errors* présenté dans la sous-section suivante.

2.2.4 Problèmes utilisés pour l'homomorphe

À partir des problèmes présentés dans la sous-section 2.2.2 et des définitions de la sous-section précédente, d'autres problèmes utiles à la cryptographie, notamment homomorphe, ont été créés.

En 2005, le problème de *Learning Parity with Noise* [6] [BKW03] est généralisé par Regev [Reg05]. Ce dernier a introduit le problème de *Learning With Errors* [7]

Définition 12 (Distribution *Learning With Errors* (LWE)) *Soit $n > 0$ et $q \geq 1$ deux entiers. Soit χ une distribution de probabilités. Soit $\boldsymbol{s} \in \mathbb{Z}_q$ un vecteur. Une distribution Learning With Errors, notée D_χ^{LWE}, retourne $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s}\rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ avec $\boldsymbol{a}$ échantillonné uniformément dans $\mathbb{Z}_q^n$ et e échantillonné par χ.*

Définition 13 (Problème du *Learning With Errors* (LWE)) *Soit $n > 0$ et $q \geq 1$ deux entiers. Soit χ une distribution de probabilités. Soit $\boldsymbol{s} \in \mathbb{Z}_q$ un vecteur. Le problème de Learning With Errors consiste à retrouver $\boldsymbol{s}$ à partir de m échantillons de D_χ^{LWE} notés $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s}\rangle + e)$.*
La version décisionnelle de ce problème repose sur la distinction des m échantillons avec m autres échantillons venant de la distribution uniforme sur $\mathbb{Z}_q^n \times \mathbb{Z}_q$.
Ce problème peut être écrit sous forme de matrice. Soit $\boldsymbol{A} \in \mathbb{Z}_q^{m\times n}$ et $\boldsymbol{e} \in \mathbb{Z}_q^m$ la matrice et le vecteur regroupant respectivement les m échantillons $\boldsymbol{a}$ et $\boldsymbol{e}$ d'une distribution LWE. Le problème LWE consiste à retrouver $\boldsymbol{s}$ à partir de $(\boldsymbol{A}, \boldsymbol{b}) = (\boldsymbol{A}, \boldsymbol{A} \times \boldsymbol{s} + \boldsymbol{e})$.

6. Apprentissage de la parité avec du bruit, en français.
7. Apprentissage par erreurs, en français.

Le problème LWE a ensuite été modifié afin d'être défini sur des anneaux de polynômes. Ce problème, appelé Ring-LWE, est introduit dans [LPR10].

Définition 14 (Distribution *Ring Learning With Errors* (R-LWE)) *Soit $n > 0$ et $q \geq 1$ deux entiers. Soit χ une distribution de probabilités. Soit $R = \mathbb{Z}[X]/f(X)$ avec $f(X) = X^d+1$ et d une puissance de 2. Soit $\boldsymbol{s} \in R$ un vecteur. Une distribution Ring Learning With Errors, notée D_{χ}^{R-LWE}, retourne $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e) \in R^n \times R$ avec $\boldsymbol{a}$ échantillonné uniformément dans R^n et e échantillonné par χ.*

Définition 15 (Problème du *Ring Learning With Errors* (R-LWE)) *Soit $n > 0$ et $q \geq 1$ deux entiers. Soit χ une distribution de probabilités. Soit $R = \mathbb{Z}[X]/f(X)$ avec $f(X) = X^d + 1$ et d une puissance de 2. Soit $\boldsymbol{s} \in R$ un vecteur. Le problème de Ring Learning With Errors consiste à retrouver $\boldsymbol{s}$ à partir de m échantillons de D_{χ}^{R-LWE} notés $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e)$.*
La version décisionnelle de ce problème repose sur la distinction des m échantillons avec m autres échantillons venant de la distribution uniforme sur $R^n \times R$.

Tous les schémas ne sont pas basés sur les réseaux. En effet, Dijk et al. [DGHV10] introduisent un schéma de chiffrement homomorphe basé sur les entiers. La sécurité de ce schéma repose sur le problème de l'Approximate Greatest Common Divisor [8] [HG01] défini ci-dessous.

Définition 16 (Problème de l'*Approximate Greatest Common Divisor* (A-GCD)) *Soit p un entier. Soit $q_i, r_i \in \mathbb{Z}\ 2 \times m$ échantillons d'entiers venant respectivement de distributions de probabilités nommées $\mathcal{D}_q$ et $\mathcal{D}_r$. Soit m entiers $x_i = q_i p + r_i$. Le problème de l'Approximate Greatest Common Divisor est de retrouver p à partir des m entiers x_i.*

2.2.5 Chiffrement NTRU

Le chiffrement NTRU [9] est un chiffrement asymétrique basé sur les réseaux, créé par Hoffstein et al. [HPS98] en 1996. Ce chiffrement fut ensuite revisité par Stehlé et al. [SS11] en 2011. Dans [SS11], la sécurité du chiffrement NTRU repose sur la difficulté du problème R-LWE. Brevetée, la bibliothèque *libntru* devient seulement libre de droits en 2017. Cette bibliothèque implémente NTRU en C. Une autre bibliothèque implémente NTRU en Java. Ces deux bibliothèques sont accessibles à partir de [Buk12]. Les performances de *libntru* se sont montrées bien supérieures à celles d'autres cryptosystèmes asymétriques dont notamment RSA [Buk12]. Certains algorithmes homomorphes ont été créés à partir du chiffrement NTRU, décrit ci-dessous.

Soit n une puissance de 2 et $q \neq 2$ un nombre premier. Soit $R = \mathbb{Z}[X]/\langle \Phi(X) \rangle$ avec $\Phi(X) = X^n + 1$ et $R_q = R/qR$. Soit $t \in R_q^{\times}$ tel que $R_t = R/tR$ soit l'ensemble des messages clairs. Soit χ une distribution *p. ex.* gaussienne.

Générations de clés. Soit g un échantillon de χ. Soit f' un échantillon de χ et $f = t \times f' + 1$. Si f n'est pas inversible dans R_q, prendre un autre f'. La génération des clés retourne (privée, publique) = $(sk, pk) = (f, h) = (f, pgf^{-1})$

Chiffrement. Soit $m \in R_t$ le message clair. Soit pk la clé publique. Soit s et e deux échantillons de χ. Le chiffrement retourne $c = pk \times s + pe + m$.

Déchiffrement. Soit $c \in R_q$ le message chiffré. Soit sk la clé privée. Le déchiffrement retourne $m = (sk \times c) \mod t$.

8. L'approximation du plus grand diviseur commun, en français.
9. N$^{\text{th}}$ Degree Truncated Polynomial Ring Units ou $\mathbb{Z}[X]/(X^N - 1)$.

2.3 Chiffrement homomorphe

Indispensable de nos jours, la cryptographie joue un rôle important dans la sécurisation de nos données au quotidien. Toutefois, les algorithmes utilisés en pratique présentent des limites dont l'impossibilité de modifier des données chiffrées. Le chiffrement homomorphe est une solution qui vise à sécuriser la modification des données numériques par des services tiers.

La cryptographie homomorphe est un domaine de recherche actif. En prouvant la faisabilité d'un algorithme de chiffrement homomorphe, Gentry [Gen09] ouvre en 2009 la quête d'un algorithme utilisable en pratique. Des techniques ont été créées afin d'optimiser au mieux les algorithmes. Ces optimisations ne rendent néanmoins pas le chiffrement homomorphe utilisable en pratique : les algorithmes présentent des limitations toujours notables qui restent à améliorer.

Cette section présente l'évolution et la création des algorithmes de chiffrement homomorphe. Bien que les recherches dans ce domaine soient récentes, le chiffrement homomorphe a connu de nombreuses avancées tant sur le plan algorithmique que sur le plan implémentation. Cette section introduit la méthode initiale pour créer un algorithme homomorphe et les différents algorithmes qui ont suivi. Puis, quelques implémentations sont citées dans cette section. Enfin, les limitations du chiffrement homomorphes sont listées.

2.3.1 *Privacy homomorphisms*

En 1978, Rivest et al. [RAD78] ont mis en avant le problème des cryptosystèmes actuels ne pouvant réaliser des opérations sur les données chiffrées. Pour répondre à ce problème, ils ont introduit la notion de *privacy homomorphisms* (ou homomorphismes privés). Ce type d'homomorphismes permettrait de faire des opérations sur les données chiffrées. C'est plus tard que le concept d'homomorphisme privé est devenu le chiffrement homomorphe. Bien qu'énoncé dans leur publication, Rivest et al. [RAD78] ne proposent aucun homomorphisme privé. Les auteurs présentent seulement quelques exemples simples afin de prouver que de tels algorithmes peuvent exister. Les exemples d'algorithmes donnés à l'époque sont néanmoins cryptographiquement faibles.

C'est seulement 31 ans plus tard, en 2009, que Gentry [Gen09] publie et propose le tout premier algorithme de chiffrement complètement homomorphe (*Fully Homomorphic Encryption* ou FHE, en anglais) qui est basé sur les réseaux euclidiens.

2.3.2 Cryptosystèmes partiellement homomorphes

Bien que le premier algorithme de chiffrement *complètement* homomorphe ne soit apparu qu'en 2009, d'autres cryptosystèmes créés avant cela sont qualifiés de *partiellement* homomorphes.

Par exemple, le cryptosystème RSA créé par Rivest, Adleman et Shamir [RSA78] s'avère être homomorphe multiplicativement. Cela veut dire que RSA, détaillé dans la section 2.1, vérifie la propriété suivante :

$$Encrypt(m_1) \times Encrypt(m_2) = Encrypt(m_1 \times m_2), \tag{2.1}$$

avec *Encrypt* la fonction de chiffrement RSA et m_1 et m_2 deux messages clairs. En effet,

$$\begin{aligned} Encrypt(m_1) \times Encrypt(m_2) &= (m_1^e \mod n) \times (m_2^e \mod n) \\ &= m_1^e \times m_2^e \mod n \\ &= (m_1 m_2)^e \mod n \\ &= Encrypt(m_1 \times m_2). \end{aligned} \tag{2.2}$$

Autrement dit, RSA vérifie :

$$Decrypt(Encrypt(m_1) \times Encrypt(m_2)) = m_1 \times m_2, \tag{2.3}$$

avec ***Encrypt*** la fonction de chiffrement RSA, ***Decrypt*** la fonction de déchiffrement RSA et m_1 et m_2 deux messages clairs. Avec ce cryptosystème, il est donc possible de multiplier des chiffrés entre eux en étant capable de pouvoir déchiffrer. Le résultat déchiffré est égal à la multiplication des clairs entre eux. Ce résultat est valide tant que $m_1 \times m_2 < n$.

De plus, certains algorithmes, comme le cryptosystème de Paillier [Pai99], vont être homomorphe additivement. Cela veut dire qu'ils vérifient la propriété suivante :

$$Encrypt(m_1) + Encrypt(m_2) = Encrypt(m_1 + m_2), \tag{2.4}$$

avec ***Encrypt*** une fonction de chiffrement et m_1 et m_2 deux messages clairs.

Le principe d'un schéma *complètement* homomorphe est d'être homomorphe additivement et multiplicativement. En d'autres termes, un schéma de chiffrement dit complètement homomorphe vérifie les deux équations suivantes :

$$Encrypt(m_1) \times Encrypt(m_2) = Encrypt(m_1 \times m_2) \tag{2.5}$$

$$Encrypt(m_1) + Encrypt(m_2) = Encrypt(m_1 + m_2). \tag{2.6}$$

2.3.3 Schéma de chiffrement homomorphe

Un schéma de chiffrement homomorphe possède cinq algorithmes :

- L'algorithme de génération de clés ou ***KeyGen*** : soit un paramètre de sécurité λ, ***KeyGen*** retourne une pair de clés (clé publique pk, clé privée sk).
- L'algorithme de chiffrement ou ***Encrypt*** : soit un message clair m et une clé publique pk, ***Encrypt*** retourne le message chiffré $c = Encrypt(m, pk)$.
- L'algorithme de déchiffrement ou ***Decrypt*** : soit un message chiffré c et une clé privée sk, ***Decrypt*** retourne le message clair $m = Decrypt(c, sk)$.
- L'algorithme d'addition homomorphe ou ***Add*** : soit c_1 et c_2 les chiffrés des messages m_1 et m_2 avec la clé pk, ***Add*** retourne le message chiffré $c = Add(c_1, c_2)$ tel que :

$$Decrypt(c, sk) = Decrypt(Add(c_1, c_2), sk) = m_1 + m_2.$$

- L'algorithme de multiplication homomorphe ou ***Mult*** : soit c_1 et c_2 les chiffrés des messages m_1 et m_2 avec la clé pk, ***Mult*** retourne le message chiffré $c = Mult(c_1, c_2)$ tel que :

$$Decrypt(c, sk) = Decrypt(Mult(c_1, c_2), sk) = m_1 \times m_2.$$

Pour toute application A représentant un nombre fini d_{add} d'additions et un nombre fini d_{mult} de multiplications, un schéma de chiffrement homomorphe doit vérifier la propriété suivante :

$$Decrypt(A(Encrypt(m_1, ..., m_n))) = Decrypt(Encrypt(A(m_1, ..., m_n))) \tag{2.7}$$

avec $(m_1, ..., m_n)$ des messages clairs. En supposant $(c_1, ..., c_n)$ les chiffrés respectifs de $(m_1, ..., m_n)$, l'équation 2.7 est équivalente à l'équation suivante :

$$Decrypt(A(c_1, ..., c_n)) = A(m_1, ...m_n). \tag{2.8}$$

Les schémas de chiffrement homomorphe retournent tous des chiffrés possédant un *bruit* (*noise*, en anglais) ou une *erreur*. Celui-ci est dû aux problèmes mathématiques (décrits dans la section 2.2) sur lesquels les schémas reposent. Au fur et à mesure des opérations effectuées après l'étape de chiffrement, *c-à-d* ***Add*** et ***Mult***, le bruit augmente (peu

après *Add* et beaucoup après *Mult*). Si le bruit est trop grand, l'équation 2.7 ne sera plus vérifiée et donc le déchiffrement du chiffré final sera erroné.

Afin de pouvoir effectuer plusieurs multiplications, certains algorithmes de générations de clés de schémas retournent également une clé, notée evk, dite d'*évaluation*. Cette clé est principalement utilisée juste après l'appel de *Mult* dans un algorithme de gestion de bruit ou de relinéarisation. Cet algorithme varie en fonction des schémas. Quelques exemples seront donnés dans les sous-sections suivantes.

Un schéma est dit *fully* ou *complètement* homomorphe si le schéma vérifie l'équation 2.7 pour d_{add} et d_{mult} quelconques. Au contraire, un schéma est dit *somewhat* ou *presque* homomorphe si le schéma vérifie l'équation 2.7 pour d_{add} quelconque et $d_{mult} \leq B$ avec B le nombre maximal de multiplications que le schéma peut effectuer. Le nombre B est appelé la *profondeur multiplicative* du schéma.

2.3.4 Premier schéma *fully* homomorphe

Avant la création d'un schéma complètement homomorphe, Boneh et al. [BGN05] ont créé un schéma permettant de faire un nombre d'additions illimité et une seule multiplication, puis, Aguilar et al. [AMGH10] ont introduit le concept de schéma *somewhat* homomorphe. En 2009, à partir d'un schéma *somewhat* homomorphe, Gentry [Gen09] créa le premier schéma complètement homomorphe. Pour cela, il a introduit une phase de *bootstrapping* (*"amorçage"*, en français) appliquée après chaque multiplication *Mult*. La méthode de bootstrapping réduit le bruit des chiffrés retournés par *Mult* permettant ainsi de pouvoir en effectuer un nombre illimité. Le principe général du bootstrapping est le suivant : soit $c = Mult(Enc(m_1, pk), Enc(m_2, pk))$ un texte chiffré après *Mult* avec m_1 et m_2 des messages clairs, le bootstrapping consiste à déchiffrer c homomorphiquement avec la clé $Encrypt(sk, pk)$ ***c-à-d*** $Decrypt^H(c, Encrypt(sk, pk)) = Encrypt(m_1 \times m_2, sk) = c'$. Le déchiffrement homomorphe applique le déchiffrement, exprimé seulement avec les opérations d'additions et de multiplications, en remplaçant chaque addition et multiplication par l'addition homomorphe et la multiplication homomorphe *c-à-d Add* et *Mult*. Le résultat du bootstrapping c' est un nouveau chiffré de $m_1 \times m_2$ ayant un bruit moins important que c. Le bootstrapping permet d'obtenir un nouveau chiffré dit *frais c-à-d* comme s'il était obtenu juste après chiffrement.

Afin de pouvoir appliquer le bootstrapping sur le schéma *somewhat* homomorphe, Gentry propose une phase de *squashing* ("écrasement", en français) pour l'algorithme de déchiffrement. Cette phase, non détaillée dans ce manuscrit, a pour objectif d'optimiser le déchiffrement afin de réduire sa profondeur multiplicative. En effet, pour pouvoir effectuer un bootstrapping, le schéma *somewhat* homomorphe doit pouvoir effectuer $n + 1$ *Mult* avec n le nombre de multiplications présents dans le déchiffrement et $+1$ la multiplication correspondant à celle effectuée juste avant le bootstrapping.

Le bootstrapping est une opération coûteuse qui demande le respect de l'hypothèse suivante : la sécurité circulaire. Le déchiffrement homomorphe prend pour clé $Encrypt(sk, pk)$. Autrement dit, le schéma requiert la clé privée chiffrée avec la clé publique. La sécurité circulaire assure qu'il est sûr d'effectuer cette opération et de rendre le résultat public.

La publication du schéma complètement homomorphe de Gentry fut une avancée considérable dans le domaine. Depuis, de nombreux schémas ont été créés en reposant sur différents problèmes. Bien qu'intéressant pour obtenir un schéma complètement homomorphe, le bootstrapping a été remplacé par d'autres techniques afin de gérer le bruit des chiffrés plus efficacement.

2.3.5 Frénésie des schémas et implémentations open-sources

Schémas basés sur A-GCD

Après le schéma basé sur les réseaux idéaux de Gentry [Gen09], Dijk et al. [DGHV10] proposent un schéma reposant sur le problème A-GCD, présenté dans la sous-section 2.2.4. Il s'agit du premier schéma sur les entiers. Ce schéma fut, par la suite, amélioré dans [CMNT11], [CNT12], [CCK+13] et enfin [CLT14]. Notamment, la version [CCK+13] utilise une technique appelée *batching* sur le schéma [DGHV10]. Le *batching* ("groupement", en français) consiste à regrouper plusieurs messages clairs dans un seul message chiffré. Cette méthode réduit le nombre de chiffrements à effectuer et, par conséquent, le nombre de chiffrés à transmettre.

Améliorations du schéma de Gentry

Smart et Vercauteren [SV10], quant à eux, sont partis de la méthode de Gentry afin de créer un schéma de chiffrement homomorphe possédant des clés et des messages chiffrés moins grands. Cependant, en implémentant et en effectuant des tests sur l'implémentation de leur schéma, les auteurs ont constaté que sa profondeur multiplicative n'est pas assez grande pour obtenir un schéma complètement homomorphe avec des tailles de clés raisonnables. L'implémentation de [GH11], variante du schéma [SV10], retourne des clés publiques allant de 70 Mo avec des paramètres d'entrée de petites tailles et jusqu'à 2.3 Go avec des paramètres d'entrée de grandes tailles.

Schémas basés sur LWE et R-LWE

Techniques de relinéarisation et *dimension-modulus reduction*

Les schémas présents dans la littérature restent majoritairement basés sur LWE et sur R-LWE. Brakerski et Vaikuntanathan [BV11b] proposent un schéma complètement homomorphe basé sur R-LWE. Les auteurs proposent deux versions d'un schéma *somewhat* homomorphe : une version symétrique et une autre asymétrique. Ces schémas utilisent la méthode du bootstrapping afin d'être complètement homomorphes. Les mêmes auteurs proposent également un autre schéma basé sur LWE [BV11a] en introduisant deux nouvelles techniques : la *relinéarisation* et le *dimension-modulus reduction* [10]. La relinéarisation vise à rendre au chiffré une forme correcte après une multiplication (un exemple est détaillé dans la sous-section 2.3.6). La technique du *dimension-modulus reduction* permet d'éviter la phase de *squashing* de Gentry (voir sous-section 2.3.4). Le but de cette technique est de convertir un chiffré $c = Encrypt(m, pk)$ en un chiffré c' du même message m ayant des paramètres, notamment un module, plus petit que le précédent chiffré c. Une *clé d'évaluation* a besoin d'être générée au départ afin d'être utilisée pour le *dimension-modulus reduction*. Cette méthode est ré-utilisée dans [BV14].

Schémas *scale invariant*

Pour son schéma basé sur LWE, Brakerski [Bra12] a introduit la notion de *scale invariance* [11]. Un schéma homomorphe *scale invariant* a une expansion du bruit plus faible qu'un schéma non *scale invariant*. Le bruit des schémas publiés avant l'article [Bra12] augmente de façon quadratique contrairement au bruit du schéma [Bra12] qui augmente linéairement. Les auteurs de [CLT14] et [BLLN13] ont également proposé des schémas *scale invariant* ; [CLT14] est basé sur les entiers tandis que [BLLN13] est basé sur NTRU.

10. Réduction de la dimension du module, en français.
11. Invariance d'échelle, en français.

Technique du *modulus swicthing*

Les auteurs de [CLT14] utilisent également la technique du *modulus switching* introduite par [BGV12]. Cette méthode est similaire au *dimension-modulus reduction* mais ne requiert pas la création d'une clé d'évaluation. La technique du *modulus switching* est également utilisée sur les schémas [CNT12], basé sur les entiers, et [CKKS17], basé sur R-LWE.

Technique de l'*approximate eigenvector*

Les auteurs de [GSW13] ont créé un schéma en utilisant une technique appelé *approximate eigenvector* [12]. La clé secrète de leur schéma est un vecteur propre approximatif du texte chiffré représenté sous forme de matrice. Le message clair est la valeur propre du chiffré. Les auteurs de [BV14] et [KGV16] se sont basés sur le schéma de [GSW13] afin d'en créer de nouveaux. Le schéma de [KGV16] est baptisé SHIELD pour *Scalable Homomorphic Implementation of Encrypted Data-Classifiers* [13].

FV et ses variantes

Fan et Vercauteren [FV12] ont fait un schéma basé sur R-LWE en introduisant deux versions optimisées de la méthode de relinéarisation. Ce schéma a été qualifié d'utilisable *en pratique* [14] et depuis, il semble être le schéma le plus intéressant dû à sa faible complexité et sa sécurité. À l'heure actuelle, deux implémentations sont disponibles de FV : [Cry16] et [LCP17]. L'implémentation [Cry16] utilise la bibliothèque NFLlib [AMBG+16], employée à effectuer de la cryptographie à base de réseaux. SEAL [LCP17] pour *Simple Encrypted Arithmetic Library* [15], développée par Microsoft Research, implémente, depuis la version 2.3 (2017), une variante de FV faite par les auteurs de [BEHZ16]. Cette variante de FV est plus performante grâce à l'utilisation du *Residue Number System* [16] ou RNS : les auteurs obtiennent une accélération d'un facteur allant de 5 à 20 (resp. de 2 à 4) pour le déchiffrement (resp. la multiplication) en comparaison avec l'implémentation de FV [Cry16] utilisant la bibliothèque NFLlib [AMBG+16].

Schémas basés sur NTRU

Enfin, plusieurs schémas sont basés sur le chiffrement NTRU [HPS98]. López-Alt et al. [LATV12] furent les premiers à créer un schéma basé sur NTRU. De plus, les auteurs qualifient leur schéma de *multikey* [17]. En effet, leur schéma permet l'évaluation de plusieurs messages chiffrés avec des clés différentes. Le déchiffrement s'effectue avec toutes les clés secrètes liées aux messages chiffrés de départ. Bos et al. [BLLN13] proposent leur schéma appelé *YASHE* pour *Yet Another Somewhat Homomorphic Encryption* [18]. YASHE semblait aussi prometteur que FV : les deux schémas furent alors l'objet d'une comparaison dans [LN14]. Néanmoins, des attaques sur le chiffrement NTRU [ABD16], [KF17] rendent les schémas [BLLN13] et [LATV12] non sûrs. Il en est de même pour le schéma [DS16] appelé *F-NTRU* pour *Flattening-NTRU* [19]

12. Vecteur propre approximatif, en français.
13. Implémentation homomorphe qui peut être mis à l'échelle de classificateurs de données chiffrés, en français.
14. Titre de l'article : *Somewhat **Practical** Fully Homomorphic Encryption Scheme*.
15. Bibliothèque pour de l'artimétique chiffrée simple, en français.
16. Système modulaire de représentation, en français.
17. Clés multiples, en français.
18. Encore un autre chiffrement *somewhat* homomorphe, en français.
19. NTRU aplani, en français.

Performances des schémas

Gentry et al. [GHS12b] ont implémenté une variante de [BGV12] dans le but d'évaluer le chiffrement AES [DR99]. Les auteurs de [GHS12b] ont fait une implémentation C++ basée sur NTL [20] [Sho17] sur une machine équipée d'un processeur Intel Xeon cadencé à 2.0 GHz et possédant 18 Mo de mémoire cache, ainsi que 256 Go de mémoire vive. Avec leur implémentation, le temps d'exécution du premier tour d'AES en homomorphe a pris 7 heures. Suite à cela, le schéma [BGV12] a également été implémenté en C++ [HS14a] avec des optimisations présentes dans [GHS12b] et [SV14]. Cette bibliothèque, appelé HElib [HS14b] est disponible en open-source. En 2014, l'exécution sur un seul cœur de la phase de bootstrapping prend moins de 5.5 minutes avec une sécurité de 76 bits [HS14c]. Ducas et Micciancio [DM15] ont ensuite proposé un schéma avec une méthode de bootstrapping exécutée en moins d'une seconde. D'autres travaux ont également fait l'objet d'une recherche d'un bootstrapping plus rapide à exécuter : [GHS12a], [ASP14], [BR15] et [CGGI16]. Chilotti et al. [CGGI16] arrivent à effectuer un bootstrapping en moins de 0.1 seconde sur une variante du schéma [GSW13]. L'implémentation de leur schéma, basé sur les tores et appelé TFHE pour *Torus-FHE*, est disponible en open-source [CGGI17].

Conclusion

À ce jour, FV [FV12] semble être le schéma suscitant le plus d'intérêt parmi ceux cités dans cette sous-section avec BGV [BGV12]. Cependant, des recherches récentes proposent de nouveaux types de schémas de chiffrement homomorphe pouvant avoir de meilleurs résultats que FV et BGV. En effet, en 2018, Dorös et al. [DHP+18] propose un schéma basé sur un nouveau problème difficile : la difficulté de retrouver un isomorphisme secret entre deux corps finis.

2.3.6 Description du schéma FV

FV [FV12] est apprécié, notamment du fait de sa simplicité en matière d'opérations, et est d'ailleurs déjà utilisé dans plusieurs implémentations de références ([Cry16], [LCP17]). Cette section décrit les primitives de ce schéma de chiffrement homomorphe basé sur R-LWE, d'après [LN14]. Autrement dit, les fonctions de génération de clés, chiffrement, déchiffrement, addition homomorphe et multiplication homomorphe sont décrites. La multiplication comprend également une phase de relinéarisation permettant de donner une forme correcte au chiffré retourné.

Soit $R = \mathbb{Z}[X]/\langle f(X)\rangle$ un anneau de polynômes dans lequel les opérations des primitives de FV vont être effectuées. Le polynôme $f(X)$ est un polynôme cyclotomique. Généralement, $f(X) = X^n + 1$ avec n une puissance de 2. Les calculs de FV sont réalisés dans $R_q = R/qR$. De plus, $R_t = R/tR$ correspond à l'espace des messages clairs, avec $1 < t < q$.

Les paramètres $w > 1$ et $l = \lfloor \log_w(q) \rfloor + 1$ sont également utilisés dans la relinéarisation. Plus précisément, w et l sont utilisés dans deux fonctions appelées *PowersOf* et *WordDecomp*. La fonction *PowersOf*, à partir d'un polynôme P, retourne un vecteur contenant les coefficients de P, chacun multiplié par une puissance de w. La fonction *WordDecomp* prend en entrée un polynôme P ayant n coefficients. Chacun des coefficients est décomposé en base w afin de créer l polynômes. Les coefficients du polynôme P_0 (resp. P_{l-1}) correspondent au bit de poids faible (resp. de poids fort) des coefficients de P en base w. Les coefficients des polynômes P_i sont les bits indice i de la décomposition des coefficients de P en base w. Les l polynômes sont retournés sous forme de vecteur.

20. *Number Theory Library*, une bibliothèque pour la théorie des nombres.

Il est notable que pour a et b des polynômes :

$$\langle WordDecomp(a), PowersOf(b)\rangle = ab \mod q. \tag{2.9}$$

PowersOf. Soit $P \in R_q$. La fonction *PowersOf* retourne $\boldsymbol{A} = (A_0, ..., A_{l-1}) \in R_q^l$ tel que $\forall i$, $A_i = [\boldsymbol{A} \cdot w^i]_q$.

WordDecomp. Soit $P \in R_q$ un polynôme de degré n. Soit $\boldsymbol{D_i} = (D_{i0}, D_{i1}, ..., D_{il})$ la décomposition en base w du coefficient à l'indice i de P. La fonction *WordDecomp* retourne $\boldsymbol{A} = (A_0, ..., A_{l-1}) \in R_q^l$ tel que $\forall j$, $A_j = \sum_{i=0}^{n} D_{ij} \cdot X^i$.

Deux distributions gaussiennes χ_{key} et χ_{err} bornées de déviation respective σ_{key} et σ_{err} sont utilisées dans le schéma FV. Enfin, soit $\Delta = \left\lfloor \frac{q}{t} \right\rfloor$. En terme de notation, $a \longleftarrow \chi$ revient à mettre un échantillon de la distribution χ dans a.

Génération de clés. Soit $s \longleftarrow \chi_{key}$ la clé privée sk. Soit $e \longleftarrow \chi_{err}$ et $a \longleftarrow R_q$. Soit $b = [-(as + e)]_q$. La clé publique pk est (b, a). Soit $\boldsymbol{a} \longleftarrow R_q^l$ et $\boldsymbol{e} \longleftarrow \chi_{err}^l$. La clé d'évaluation $\boldsymbol{evk}$ est :

$$\boldsymbol{evk} = ([PowersOf(sk^2) - (e + a \cdot s)]_q, \boldsymbol{a}). \tag{2.10}$$

Chiffrement. Soit $m \in R_t$ un message clair et soit pk la clé publique. Soit $u \longleftarrow \chi_{key}$, $e_1, e_2 \longleftarrow \chi_{err}$. Le chiffré $\boldsymbol{c}$ est :

$$\boldsymbol{c} = ([\Delta m + bu + e_1]_q, [au + e_2]q). \tag{2.11}$$

Addition. Soit $\boldsymbol{c_1}, \boldsymbol{c_2}$ deux messages chiffrés tels que $\boldsymbol{c_1} = (c_{10}, c_{11})$ et $\boldsymbol{c_2} = (c_{20}, c_{21})$. L'addition de deux chiffrés est $\boldsymbol{c} = ([c_{10} + c_{20}]_q, [c_{11} + c_{21}]_q)$.

Multiplication. Soit $\boldsymbol{c_1}, \boldsymbol{c_2}$ deux messages chiffrés tels que $\boldsymbol{c_1} = (c_{10}, c_{11})$ et $\boldsymbol{c_2} = (c_{20}, c_{21})$. Soit $evk = (\boldsymbol{evk_b}, \boldsymbol{evk_a})$ la clé d'évaluation. Soit $\boldsymbol{c_{mult}}$ tel que :

$$\begin{aligned} \boldsymbol{c_{mult}} &= (c_{mult0}, c_{mult1}, c_{mult2}) \\ &= \left(\left[\left\lfloor \frac{t}{q} \cdot c_{10} \cdot c_{20} \right\rceil \right]_q, \left[\left\lfloor \frac{t}{q} \cdot (c_{10} \cdot c_{21} + c_{11} \cdot c_{20}) \right\rceil \right]_q, \left[\left\lfloor \frac{t}{q} \cdot c_{11} \cdot c_{21} \right\rceil \right]_q \right). \end{aligned} \tag{2.12}$$

Une phase de relinéarisation est ensuite appliquée à $\boldsymbol{c_{mult}}$ dans le but qu'il ait une forme de chiffré *c-à-d* que $\boldsymbol{c_{mult}}$ soit de la forme (c_{mult0}, c_{mult1}). Cette phase consiste en les calculs suivants :

$$\begin{aligned} c_{mult0} &= [c_{mult0} + \langle WordDecomp(c_{mult2}), \boldsymbol{evk_b}\rangle]_q \\ c_{mult1} &= [c_{mult1} + \langle WordDecomp(c_{mult2}), \boldsymbol{evk_a}\rangle]_q. \end{aligned} \tag{2.13}$$

Déchiffrement. Soit $\boldsymbol{c} = (c_0, c_1)$ un message chiffré et soit sk la clé secrète. Le déchiffrement retourne m le message clair tel que :

$$m = \left[\left\lfloor \frac{t}{q} \cdot [c_0 + c_1 s]_q \right\rceil \right]. \tag{2.14}$$

2.3.7 Limitations

Le chiffrement homomorphe a évolué rapidement depuis 2009 dans l'objectif de sécuriser les traitements de données numériques par des tiers. Aujourd'hui, de nombreux schémas de chiffrement homomorphe existent dont certains sont utilisables grâce aux implémentations open-sources. Basés sur des problèmes difficiles, il existe différentes variétés de schémas. Cependant, certaines limitations sont notables et n'ont pas permis au chiffrement homomorphe de s'imposer comme un standard parmi les algorithmes cryptographiques.

Cette sous-section présente les limitations et intérêts de recherche pour améliorer cette méthode de chiffrement. Cette sous-section aborde notamment la difficulté à choisir un schéma pour une application donnée et à choisir les paramètres d'entrées. De plus, le chiffrement homomorphe est connu pour avoir une grande complexité et un coût mémoire important. Les causes de ces inconvénients majeurs, incluant les techniques de gestion d'erreur utilisées *p. ex.* bootstrapping et les objets mathématiques manipulés dans les schémas, sont exposées dans cette sous-section. Enfin, le choix des paramètres d'entrée pour obtenir une sécurité donnée est constamment remis en question, ce qui questionne l'applicabilité du chiffrement homomorphe pour une utilisation quotidienne.

Gestion de l'erreur : comment en réduire les coûts ?

En créant le premier schéma, Gentry [Gen09] a introduit la méthode de bootstrapping permettant, à partir d'un schéma *somewhat* homomorphe, d'obtenir un schéma complètement homomorphe *c-à-d* un schéma avec lequel il est possible d'effectuer un nombre illimité de multiplications. Cette méthode a été introduite afin de réduire le bruit engendré après l'application d'une multiplication.

Cependant, cette méthode demande beaucoup de temps de calcul et correspond à l'opération la plus coûteuse d'un schéma. Pour permettre une meilleure gestion du bruit, les auteurs se sont intéressés d'une part à réduire le coût du bootstrapping et d'autre part, à trouver d'autres méthodes moins coûteuses.

Les travaux décrits dans [GHS12a], [ASP14], [DM15], [BR15] et [CGGI16] visent à réduire le temps d'exécution du bootstrapping. D'un autre côté, les auteurs de [BV11a], [CNT12], [BGV12], [Bra12], [BLLN13], [BV14], [CLT14] et, plus récemment, [RVVV17], [CKKS17] et [CHK+18] utilisent des méthodes moins coûteuses telles que la relinéarisation, la *scale invariance*, le *dimension-modulus reduction* et sa variante, le *modulus switching*. Toutes ces méthodes sont décrites dans la sous-section 2.3.5. Les auteurs de [RVVV17] introduisent une *recryption box* [21] permettant de réduire le bruit des chiffrés sans que le schéma n'ait besoin d'une profondeur minimale, comme cela est le cas pour le bootstrapping.

Complexité : comment optimiser les schémas ou les applications ?

En écartant la gestion du bruit et donc par conséquent le bootstrapping, les opérations effectuées dans un schéma de chiffrement homomorphe restent tout de même coûteuses. Des travaux de recherche ont donc été menés dans l'optique d'accélérer les dernières.

Notamment, en effectuant des opérations simultanément à l'aide d'opérateurs SIMD (*Single Instruction Multiple Data* [22]), les auteurs de [SV14] et [GHS12b] optimisent l'exécution de leurs schémas de chiffrement homomorphe.

Migliore et al. [MRL+18] proposent une implémentation co-design, *c-à-d* logicielle et matérielle, de FV [FV12] qui inclus une accélération matérielle de la multiplication

21. Boîte de "re-chiffrement", en français.
22. Instruction unique, données multiples, en français.

polynomiale en utilisant l'algorithme de Karatsuba [KO63]. Toujours dans le but d'accélérer FV, Bajard et al. [BEHZ16] ont introduit des techniques de RNS (voir sous-section 2.3.5) dans ce schéma. Enfin, la *recryption box*, créée par Roy et al., a été implémentée [RVVV17] sur FPGA. Cette *recryption box*, testée par les auteurs de [RVVV17] sur FV, permet de rafraîchir un texte chiffré *c-à-d* réduire le bruit du texte chiffré.

De son côté, le CEA (Commissariat à l'Énergie atomique et aux Énergies alternatives) a développé le compilateur Cingulata [Tea17], aussi connu sous le nom de Armadillo [CDS15], permettant d'optimiser les applications. Le compilateur propose une interface de programmation pour décrire une application. Cette application est ensuite transformée sous la forme d'un circuit booléen. Ce circuit est optimisé afin de réduire la profondeur multiplicative de l'application. Le compilateur propose également l'exécution de l'application avec des schémas homomorphes. Pour l'instant, seul FV est disponible dans Cingulata.

Coût mémoire : comment réduire le coût d'envoi des données chiffrées ?

Le chiffrement homomorphe souffre également d'une importante expansion de chiffré *c-à-d* que la taille des chiffrés est considérable face à la taille des clairs. L'intérêt du chiffrement homomorphe est de pouvoir déporter des calculs sur un serveur tiers. L'envoi des données chiffrées va alors devenir un facteur limitant du fait de leur importante taille.

La méthode du *batching*, utilisé notamment dans [CCK+13], regroupe plusieurs messages en clair un seul message chiffré. Cette méthode évite la création multiple de messages chiffrés. Par conséquent, en regroupant k messages clairs en un message, seul un chiffrement est requis et non k chiffrements. Autrement dit, en fonction du nombre k de messages clairs regroupés, le nombre de chiffrés à transmettre est réduit par k.

Naehrig et al. [NLV11] propose une solution pour réduire le coût de transfert entre *p. ex.* Alice et un serveur cloud. Alice chiffre ses données avec AES [DR99] et chiffre sa clé symétrique AES avec un schéma de chiffrement homomorphe. En passant par AES, les tailles des chiffrées sont relativement petites face à celles des chiffrées produits par un schéma homomorphe. Alice envoie ensuite les données chiffrées au serveur cloud. Le serveur doit alors transformer les messages chiffrés en AES par des messages chiffrés en homomorphe. Pour cela, le serveur déchiffre homomorphiquement (voir sous-section 2.3.4) les chiffrés AES avec la clé symétrique chiffrée en homomorphe.

Canteaut et al. [CCF+16] sont partis de cette méthode afin de créer le *trans-chiffrement*. Les auteurs ont utilisés comme chiffrement symétrique une variante du chiffrement Trivium [DCP08]. Cette variante, appelée *Kreyvium*, possède un niveau de sécurité supérieure à Trivium (128 bits contre 80 bits) avec une profondeur multiplicative de 12, identique à Trivium. En reprenant l'exemple d'Alice et d'un serveur cloud, le trans-chiffrement permet aux deux entités de générer un *keystream* [23] en utilisant *Kreyvium* et la clé symétrique k pour Alice et en utilisant *Kreyvium* (en remplaçant toutes les additions et multiplications par des additions homomorphes et des multiplications homomorphes) et $Encrypt_{HE}(k)$ (k chiffré en homomorphe) pour le serveur. Alice possède donc k, $keystream$ et m le message. Le serveur cloud possède $Encrypt_{HE}(k)$ et $Encrypt_{HE}(keystream)$. Alice envoie $c = m \oplus keystream$ au serveur qui calcule ensuite $c \oplus Encrypt_{HE}(keystream)$ dans le domaine homomorphe *c-à-d* avec des additions homomorphes.

Ce résultat correspond à $Encrypt_{HE}(m)$. Le serveur peut alors effectué l'application demandée par Alice. Afin de l'effectuer et pour que le déchiffrement soit correct, l'application doit posséder un nombre maximal de successions de multiplications équivalent à $d - 12$ avec d le nombre de multiplications que le schéma homomorphe

23. Flux de clé, en français. Il s'agit d'une suite de bits.

peut effectuer au maximum et 12 le nombre de multiplications successives déjà effectuées sur $Encrypt_{HE}(m)$. En effet, $Encrypt_{HE}(keystream)$ est passé par 12 multiplications successives en exécutant Kreyvium, de profondeur 12, homomorphiquement.

Le trans-chiffrement permet seulement de réduire le coût d'envoi des messages d'entrée chiffrés, mais pas la taille du résultat chiffré retourné par le serveur distant. Afin de réduire le coût d'envoi des données chiffrés du serveur au client, les auteurs de [CS15] ont créé une méthode pour empaqueter plusieurs chiffrées en un seul et la méthode inverse servant à extraire les chiffrés à partir d'un chiffré empaqueté. Ces méthodes, également applicables sur les messages clairs, permettent de réduire les coûts d'envoi de messages chiffrés du client au serveur et du serveur au client.

Sécurité : quels paramètres pour une sécurité λ ?

Lorsqu'un algorithme de chiffrement est créé, celui-ci est attaqué afin d'en trouver les failles pour les corriger par la suite. Il en a été de même dans le domaine de l'homomorphie. Les attaques recherchées se sont portées sur les problèmes LWE et R-LWE ([LP11], [APS15], [BF17]). Albrecht [Alb17c] s'est focalisé, quant à lui, à attaquer HElib [HS14a] et SEAL [LCP17] lors de l'utilisation de petits paramètres.

Les auteurs de [APS15] ont développés sur Sage [SAA16] un estimateur [Alb17a] et un générateur [Alb17b] LWE. L'estimateur estime les sécurités à partir d'une instance de LWE. Le générateur donne des échantillons de distributions LWE tels que proposées dans la littérature *p. ex.* [Reg05], [LP11] et [LPR10].

Migliore et al. [MBF17] étudient également trois schémas afin d'indiquer quels paramètres sélectionner pour une sécurité fixée (80 bits et 128 bits).

Complexité et coût mémoire : comment les estimer ?

La publication d'un nouveau schéma s'accompagne d'une analyse de ses performances. Seulement, les performances données correspondent aux temps d'exécution de leur schéma pour quelques paramètres d'entrée. L'implémentation des auteurs est rarement disponible en ligne, limitant ainsi l'application du schéma sur d'autres paramètres d'entrée.

Le nombre de paramètres d'entrée d'un schéma de chiffrement homomorphe est en général assez important par rapport aux chiffrements actuels. En effet, d'un côté, le schéma FV (décrit dans la section 2.3.6), celui-ci possède 6 paramètres d'entrée *c-à-d* q, n, t, w, σ_{err} et σ_{key}. D'un autre côté, le chiffrement AES (comme pour le chiffrement RSA) prend en entrée seulement la taille de la clé. La variation des paramètres des chiffrements homomorphes peut engendrer des résultats très différents allant d'un temps d'exécution très faible (< 1 secondes) à très élevé (plusieurs minutes).

Les travaux de Lepoint et Naehrig [LN14] analysent et comparent les performances de deux schémas : FV [FV12] et YASHE [BLLN13]. De plus, Cathébras et al. [CCSV17] ont effectué une analyse centrée sur les paramètres de FV en visant une accélération matérielle. Migliore et al. [MBF17] déterminent des paramètres d'entrée de FV [FV12] et SHIELD [KGV16] impliquant une sécurité fixée de 80 bits ou de 128 bits. Plusieurs jeux de paramètres sont données en fonction du type de circuit sur lequel sont appliqués les schémas. Les deux types de circuit sont : arbitraire et optimisé. Le circuit optimisé correspond au circuit arbitraire réarrangé de telle sorte qu'il comprenne davantage de multiplications entre un chiffré et un chiffré frais (obtenu juste après un chiffrement). Dans le schéma SHIELD, le bruit d'une multiplication prenant en entrée au moins un chiffré frais grandit faiblement en comparaison du bruit généré par une multiplication entre deux chiffrés non frais. Cette particularité permet à SHIELD d'avoir une profondeur multiplicative plus grande avec des paramètres d'entrée plus petits. Autrement dit, SHIELD peut être utilisé pour ce circuit

optimisé avec des paramètres d'entrée moins importants impliquant une consommation mémoire moins importante que celle de FV. Dans le cas du circuit arbitraire, FV donne de meilleurs résultats en matière de consommation mémoire que SHIELD.

Les travaux cités ci-dessus présentent des résultats suite à de multiples exécutions de leur implémentation. Ces exécutions permettent de déterminer les coûts (algorithmique et mémoire) des schémas sélectionnés. Sans exécution concrète, aucune comparaison entre schémas n'est possible. Dans ce manuscrit de thèse, nos travaux, notamment publiés dans [FLL17] et [FLL18], proposent une méthode dans le but d'analyser de schémas, sans exécutions complètes de ceux-ci. Cette méthode propose tout d'abord un calcul théorique des schémas en matière de complexité algorithmique et consommation mémoire dans le pire. Le calcul théorique de la complexité (resp. consommation mémoire) est ensuite estimé en temps d'exécution et en mébioctects, en se basant sur quelques exécutions concrètes de schémas. Cette méthode retourne les analyses en temps d'exécution et consommation mémoire des schémas plus rapidement qu'une exécution complète des schémas.

Diversité des schémas et choix des paramètres : quel schéma choisir pour une application donnée ?

La recherche du schéma homomorphe utilisable en pratique a entraîné la création de nombreux schémas. Des techniques ont été inventées afin de réduire les temps d'exécutions et les coûts mémoire des schémas. Certaines recherches portent sur l'utilisation de schémas dans des applications précises.

Carpov et al. [CNS+16] utilise FV [FV12] dans une application médicale. Cette application, à partir d'informations personnelles du patient, retourne un facteur de risque cardiaque. Plus ce facteur, compris entre 0 et 9, est grand et plus le risque d'avoir une maladie cardiaque est important. Pour le côté client, les auteurs ont notamment développé une application Android. FV a également été utilisé pour une application de compression *run-length encoding* [24] (RLE) dans [CCKS17]. Tajan et al. [TKW18] se sont intéressés à trouver des paramètres d'entrée pour optimiser l'exécution de chiffrement homomorphe dans des applications d'audit de clouds.

Les travaux cités ci-dessus ont sélectionné un schéma afin de l'utiliser ou d'y trouver des optimisations pour leur application précise. Aucune étude n'a été menée dans le but de comparer l'efficacité de plusieurs schémas pour une application. Les travaux de Lepoint et Naehrig [LN14] permettent une première comparaison de schémas, en l'occurrence FV [FV12] et YASHE [BLLN13]. Cette comparaison donne des indications sur quel schéma semble le moins coûteux en temps et mémoire mais n'est pas forcément représentative de leurs mises en pratique pour une application donnée.

Un workshop [25] sur le chiffrement homomorphe a eu lieu pour la seconde fois en 2018. Ce workshop vise à rassembler des experts et étudiants travaillant sur le chiffrement homomorphe afin de mettre leurs travaux en commun. Suite à cette rencontre, un standard a été écrit [ACC+18] décrivant les schémas et les paramètres d'entrée à utiliser.

Actuellement, il est difficile de pouvoir comparer tous les schémas de la littérature. Seules les implémentations open-sources nous permettent d'effectuer des comparaisons de schéma et de déterminer quel schéma (et paramètres d'entrée) semble avoir les meilleures performances pour une application donnée. Ce manuscrit de thèse propose une solution à cette problématique.

24. Codage par plages, en français.
25. https://projects.csail.mit.edu/HEWorkshop/index.html

2.4 Conclusion

Utilisée au quotidien, la cryptographie est une science permettant la sécurisation de des données personnelles. Présente dans les cartes bancaires comme sur les sites internet, elle rend possible en particulier le stockage de données chiffrées d'utilisateurs par des tiers de confiance ou non sur des serveurs distants. Cependant, les algorithmes utilisés actuellement ne sont pas adaptés à la manipulation de données chiffrées. Cette fonctionnalité permettrait de déléguer des calculs à des services de type cloud computing sans perte de confidentialité des données.

Le chiffrement homomorphe vise à répondre à cette problématique. Le premier schéma de chiffrement homomorphe créé en 2009 par Gentry [Gen09] a ouvert des portes quant à la recherche d'un schéma utilisable en pratique. Cependant, de multiples limitations sont encore présentes pour la cryptographie homomorphe, bien que celle-ci présente un grand potentiel pour la sécurisation future des données numériques. L'engouement pour trouver l'algorithme homomorphe utilisable en pratique a engendré la création de nombreux schémas ayant des propriétés relativement similaires. La complexité algorithmique et le coût mémoire sont les deux principaux défis du chiffrement homomorphe. Malgré leur calculabilité et la similarité des schémas, aucune méthode permettant d'analyser et/ou comparer les schémas n'existe à ce jour. Autrement dit, face à tous les schémas de la littérature, il est difficile d'analyser un schéma sans l'exécuter. Par conséquent, la sélection d'un schéma pour une application donnée est une procédure longue et fastidieuse car elle requiert l'exécution de tous les schémas et de leurs paramètres d'entrée.

Les travaux menés durant cette thèse se sont tournés sur la création d'une méthode d'analyse des schémas de chiffrement homomorphe. Afin de pouvoir créer cette méthode, une étape de modélisation a été créée dans le but de pouvoir représenter rapidement chaque schéma sous un même format. Cette modélisation peut donner suite à différentes analyses de schémas dont notamment une analyse de la complexité algorithmique et du coût mémoire. Les résultats d'analyse de plusieurs schémas permettent une comparaison des schémas sans nécessité d'exécutions de ceux-ci.

Chapitre 3

Modélisation et analyse théorique de schémas de chiffrement homomorphe

Sommaire

Indispensable dans notre société, la cryptographie est omniprésente pour sécuriser les données numériques. Parmi les cryptosystèmes existants, le chiffrement homomorphe est une solution pour contrer le manque de confidentialité lors de l'utilisation d'un service tiers, par exemple, dans un environnement de *cloud computing*. De part sa construction, le chiffrement homomorphe continuera d'être utilisable après l'émergence de l'informatique quantique. Cependant, le grand nombre de paramètres d'entrée pour chaque schéma ainsi que les coûts importants en temps et mémoire pendant l'exécution rendent le chiffrement homomorphe non utilisable en pratique actuellement. De plus, la diversité des schémas ne cesse de croître ce qui complique davantage la comparaison des schémas entre eux. Il devient alors difficile de sélectionner un schéma et ses paramètres d'entrée impliquant une faible complexité et un faible coût mémoire pour une application donnée. Une méthode générique semble nécessaire pour pouvoir analyser les schémas. Cette méthode doit pouvoir être alors appliquée sur tout type de schémas de chiffrement homomorphe.

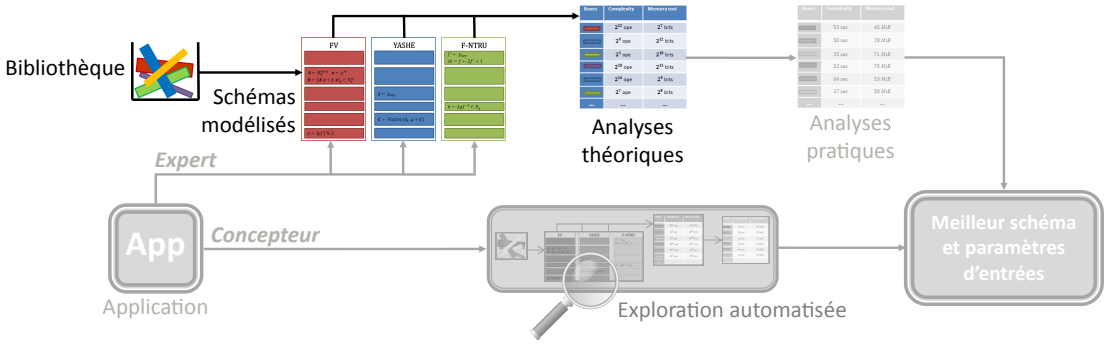


FIGURE 3.1 – Différentes étapes de PAnTHErS avec la modélisation et l'analyse théorique mises en avant.

Pour ce faire, ce chapitre présente deux méthodes : une méthode permettant de modéliser des schémas de chiffrement homomorphe et une méthode permettant de les analyser rapidement de manière théorique. La modélisation permet de représenter les schémas sous un format commun afin de pouvoir ensuite appliquer différentes analyses. Ces deux méthodes constituent les deux étapes mises en avant sur la Figure 3.1 représentant le schéma récapitulatif de l'outil PAnTHErS final. Les méthodes présentées dans ce chapitre ont été publiés dans [FLL17].

3.1 Modélisation de schémas de chiffrement homomorphe

Les cryptosystèmes homomorphes peuvent être une solution pour protéger les données utilisées ou modifiées par une tierce personne *p. ex.* un service cloud computing. Pour l'instant inexploitables commercialement, les recherches autour de ce mode de chiffrement sont actives, ce qui entraîne la création de nombreux schémas. Par conséquent, une des limitations du chiffrement homomorphe est la difficulté de choisir un schéma plutôt qu'un autre pour une application donnée. Or, ces schémas sont basés sur des problèmes similaires *p. ex.* LWE ou R-LWE (voir chapitre 2). Ces similarités, chacune modélisée sous la forme d'un algorithme, peuvent aider à implémenter rapidement plusieurs schémas de la littérature.

Cette section présente une méthodologie utilisée pour identifier des similarités entre plusieurs schémas afin de les modéliser efficacement. La méthode est ensuite illustrée en l'appliquant sur deux schémas ([FV12] et [BLLN13]) de chiffrement homomorphe.

3.1.1 Liste des schémas considérés dans ce manuscrit

Depuis la création du premier schéma de chiffrement complètement homomorphe, basé sur les réseaux euclidiens, par Gentry en 2009 [Gen09], de nombreux schémas ont été créés en se basant sur plusieurs problèmes NP-complets différents *p. ex.* LWE ou approximate-GCD.

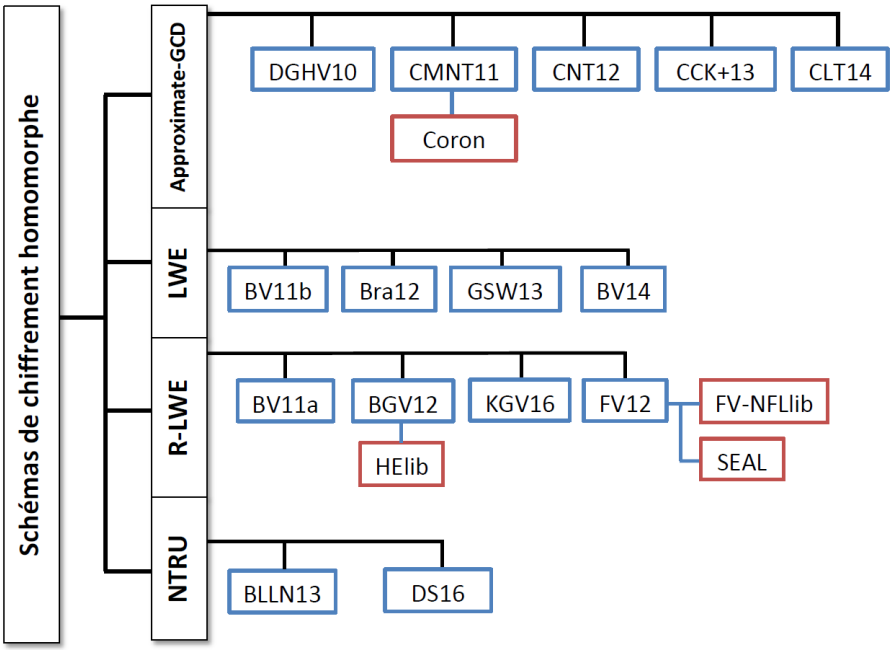


FIGURE 3.2 – Taxonomie de schémas de chiffrement homomorphe créés entre 2010 et 2016.

Une classification naturelle de ces schémas s'effectue selon les problèmes sur lesquels ils sont basés. Un telle classification est visible sur la Figure 3.2 qui présente 15 schémas, rendus publiques entre 2010 et 2016, et quatre implémentations open-sources.

Tous les schémas reposant sur un même problème vont posséder des instructions communes liées à ce problème précis. À partir de cette constatation, nous nous sommes penchés sur la réalisation d'une méthode de modélisation permettant de modéliser les 15 schémas présents sur la Figure 3.2 mais également tout nouveau schéma qui pourrait être créé dans le futur.

3.1.2 Présentation de la méthode de modélisation

La classification effectuée dans la section précédente permet d'établir une méthode de modélisation efficace des schémas. En effet, reposant sur des problèmes mathématiques difficiles similaires, les schémas de chiffrement homomorphe possèdent des séquences d'instructions identiques. Cette particularité fonde notre méthode de modélisation en fixant certains objectifs et exigences.

Objectif & exigences

L'objectif principal est d'établir une modélisation simple à comprendre et rapide à utiliser. Afin de pouvoir modéliser tout schéma de chiffrement homomorphe, la méthode doit être applicable sur toutes sortes de schémas reposant sur des problèmes difficiles. Autrement dit, la méthode doit pouvoir être appliquée sur les schémas de la littérature, mais aussi sur de futurs schémas.

De surcroît, la méthode de modélisation utilisée doit pouvoir être étendue : elle doit permettre de réaliser des études approfondies telles que l'analyse de la complexité ou de l'empreinte mémoire de schémas de chiffrement homomorphe.

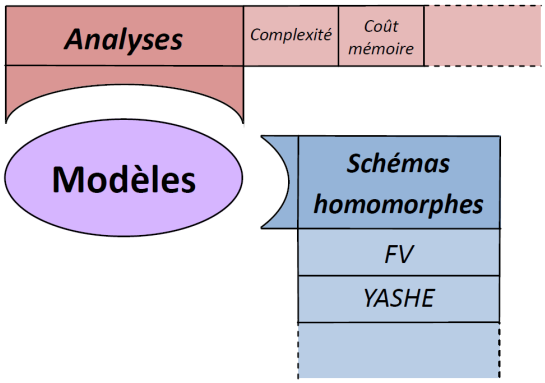


FIGURE 3.3 – Représentation de l'interaction entre les modèles d'évaluation et modèles algorithmiques.

Nous avons conçu un modèle d'abstraction (Figure 3.3) se basant sur les objectifs énoncés précédemment. La Figure 3.3 schématise l'interaction entre :

• des modèles algorithmiques représentant des schémas de chiffrement homomorphe (*p. ex.* FV [FV12] et YASHE [BLLN13]),

• des modèles d'évaluation tels que l'analyse de la complexité algorithmique ou l'analyse du coût mémoire.

Modélisation

Une étude préalable des schémas a été nécessaire pour trouver une méthode répondant aux objectifs et exigences fixés en amont. L'étude consistait à comparer les schémas c'est-à-dire à identifier des similitudes et différences sur le plan algorithmique.

Lors de cette étude, trois niveaux de similitudes ont pu être constatés :

• À haut niveau, les schémas de chiffrement homomorphe sont constitués de cinq fonctions principales : génération de clés, chiffrement, addition homomorphe, multiplication homomorphe et déchiffrement.

• À niveau intermédiaire, des similitudes sont visibles entre plusieurs schémas basés sur un même problème NP-complet. Ces similitudes correspondent à des séquences d'instructions relativement similaires.

• À bas niveau, tous les schémas utilisent des opérations de base telles que l'addition, la multiplication, le modulo, la division, etc... Ces opérations sont effectuées sur des entiers, des polynômes ou des matrices.

Pour établir notre modélisation, nous avons adopté une approche agile [Mar02]. L'idée de l'approche agile est d'avancer de manière itérative. Le concept est de ne développer qu'une fonctionnalité à la fois. Chaque fonctionnalité est ensuite validée en s'assurant de son bon fonctionnement et de sa bonne intégration dans l'environnement existant.

Dans un premier temps, en suivant ces principes, nous avons validé une première méthode de modélisation sur deux schémas ([FV12] et [BLLN13]). Dans un second temps, nous avons affiné notre méthode de modélisation en intégrant un troisième schéma. Lors des premières itérations, notre méthode reçoit des adaptations successives. Les algorithmes déjà modélisés subissent un *refactoring* : ils doivent être adaptés pour prendre en compte certains aspects introduits par le nouvel algorithme. Par exemple, des opérations ou des

paramètres d'entrée peuvent être rajoutés à un algorithme, ce qui permet de le rendre plus générique.

Ainsi de suite, plus le nombre de schémas modélisés sont intégrés, plus la méthode de modélisation devient stable : il n'est plus nécessaire d'apporter des adaptations d'une itération à la suivante. Ainsi, les algorithmes créés lors d'une itération n'ont plus besoin d'être modifiés pour une itération ultérieure.

La méthode de modélisation finalement obtenue se base sur les trois constatations décrites précédemment. Elle fait intervenir trois types d'algorithmes : atomiques, spécifiques et basiques.

***Atomiques* ou algorithmes atomiques :** Un algorithme atomique représente une opération algébrique sur un ou plusieurs objet mathématiques. Les algorithmes atomiques englobent les opérations algébriques sur les entiers, les polynômes et les matrices dont l'addition, la multiplication, la soustraction, le modulo, le tirage aléatoire et l'arrondi. Le *tirage aléatoire* consiste à extraire un élément aléatoire d'un ensemble. Les algorithmes d'addition et de multiplication d'entiers, polynômes ou matrices vont être stockés tels que représentés par les Algorithmes 1 et 2. La conversion en binaire et la puissance sont deux Atomiques applicables seulement sur les entiers. L'Atomique *produit scalaire* est réservé aux vecteurs et les Atomiques *division* et *inverse* sont définis sur les entiers et les polynômes. Par la suite, des algorithmes plus complexes *c-à-d* des algorithmes spécifiques vont être créés à partir des Atomiques qui en sont des blocs de base.

Algorithme 1 $add(a, b, c)$

Entrées : a, b entiers ou polynômes ou matrices.
Sorties : c entier ou polynôme ou matrice.
1: $c = a + b$

Algorithme 2 $mult(a, b, c)$

Entrées : a, b, c entiers ou polynômes ou matrices.
Sorties : c entier ou polynôme ou matrice.
1: $c = a \times b$

***Spécifiques* ou algorithmes spécifiques :** Un algorithme spécifique est une combinaison d'algorithmes atomiques et/ou spécifiques. Ces algorithmes spécifiques vont être créés suite à l'étude de schémas de chiffrement homomorphe reposant sur des problèmes similaires. Des séquences d'instructions identiques entre deux schémas pourront être modélisées sous la forme d'un Spécifique. Comme une séquence d'instructions est un enchaînement d'opérations unitaires, chacune de ces opérations sera alors représentée par un algorithme atomique. Un bon algorithme spécifique est une séquence d'instructions pouvant être réutilisée entre plusieurs schémas de chiffrement homomorphe. Pour illustrer ces propos, l'équation

$$addTimes(\boldsymbol{A}, b, \boldsymbol{C}) = \boldsymbol{A} + b \times \boldsymbol{C} \tag{3.1}$$

avec A, C des matrices et b un entier, décrit un algorithme spécifique qui sera représenté de manière généralisée par l'Algorithme 3. L'algorithme *addTimes* utilise alors deux algorithmes atomiques correspondant aux Algorithmes 1 et 2.

Comme présenté précédemment, nous avons employé une approche agile pour modéliser les différents algorithmes. Ainsi, au cours du processus de modélisation, certains algorithmes spécifiques ont été modifiés afin d'être généralisés (étape de *refactoring*).

Algorithme 3 *addTimes*(a, b, c, d)

Entrées : a, b, c entiers ou polynômes ou matrices. Condition : b ou c est une matrice $\Leftrightarrow$ a est une matrice.
Sorties : d entier ou polynôme ou matrice.
1: mult(b, c, d) // $d \leftarrow b \times c$
2: add(a, d, d) // $d \leftarrow a + d$

Par exemple, l'Algorithme 4 appelé *distriLWE* permettant la création d'une distribution LWE (voir section 2.2), d'abord modélisé en l'Algorithme 5, a été généralisé en l'Algorithme 6. L'Algorithme 4 a d'abord été créé pour le schéma [FV12]. Entièrement modélisé, l'Algorithme 4 devient l'Algorithme 5, correspondant à un enchaînement d'Atomiques et Spécifiques. Chaque opération unitaire est remplacée par son équivalent en Atomique. Chaque Spécifique ou Atomique utilisé dans l'Algorithme 5 possède son équivalent mathématique en commentaire. L'Algorithme 5 prenait alors six paramètres en entrée : l'entrée k, scalaire apparaissant dans la 3$^{\text{ème}}$ instruction de l'Algorithme 4, n'était pas présente. Créé sans k, ce paramètre fut rajouté lors de la modélisation de [BV11a]. En effet, dans le schéma [BV11a], le paramètre d'entrée k est utile car $k \neq 1$. Cependant, pour le schéma [FV12], $k = 1$ et donc k est facultatif. L'Algorithme 5 a donc été généralisé en rajoutant un paramètre d'entrée. Cet algorithme est utilisé dans [FV12] et peut être utilisé également pour [BV11a].

Une autre solution aurait été de créer un autre Spécifique *c-à-d* de conserver les Algorithmes 5 et 6. Par conséquent, deux Spécifiques auraient été très ressemblants. Si deux algorithmes se ressemblent, alors la bibliothèque d'algorithmes devient plus importante et le choix d'un algorithme plutôt qu'autre peut ralentir le processus de modélisation. Ne perdant pas de vue l'objectif de créer une méthode de modélisation rapide, il est plus judicieux de généraliser les algorithmes spécifiques si possible.

Algorithme 4 *distriLWE* (séquences d'instructions mathématiques)

Entrées : q, n, m, k entiers, χ distribution gaussienne, R un anneau, $\boldsymbol{s}$ un vecteur de taille n.
Sorties : *distriLWE*$(q, n, m, k, \chi, R, \boldsymbol{s})$
1: $\boldsymbol{A} \leftarrow R_q^{m \times n}$
2: $\boldsymbol{b} \leftarrow \chi^m$
3: $\boldsymbol{b} = (\boldsymbol{A}.\boldsymbol{s} + k.\boldsymbol{b}) \mod q$
4: **return** $\boldsymbol{A} \in R_q^{m \times n}, \boldsymbol{b} \in R_q^m$

Algorithme 5 *distriLWE*$(q, n, m, \chi, R, \boldsymbol{s}, (A, b))$ non généralisé

Entrées : q, n, m entiers, χ distribution gaussienne, R un anneau, $\boldsymbol{s}$ un vecteur de taille n.
Sorties : A matrice de taille $m \times n$, b vecteur de taille m
1: rand$(R, m, n, \boldsymbol{A})$ // $\boldsymbol{A} \leftarrow R^{m \times n}$
2: rand$(\chi, m, 1, \boldsymbol{b})$ //$\boldsymbol{b} \leftarrow \chi^m$
3: mult$(\boldsymbol{A}, \boldsymbol{s}, \boldsymbol{c})$ //$\boldsymbol{c} \leftarrow \boldsymbol{A}.\boldsymbol{s}$
4: add$(\boldsymbol{c}, \boldsymbol{b}, \boldsymbol{b})$ //$\boldsymbol{b} \leftarrow (\boldsymbol{c} + \boldsymbol{b})$
5: mod$(\boldsymbol{b}, q, \boldsymbol{b})$ //$\boldsymbol{b} \leftarrow \boldsymbol{b} \mod q$

***Homomorphes basiques* ou algorithmes homomorphes basiques :** Ces algorithmes vont correspondre aux cinq fonctions haut niveau d'un schéma de chiffrement homomorphe classique *c-à-d* génération de clé, chiffrement, addition homomorphe,

Algorithme 6 *distriLWE*$(q, n, m, k, \chi, R, \boldsymbol{s}, (A, b))$ modélisé et généralisé

Entrées : q, n, m, k entiers, χ distribution gaussienne, R un anneau, $\boldsymbol{s}$ un vecteur de taille n.
Sorties : A matrice de taille $m \times n$, b vecteur de taille m

1: rand$(R, m, n, \boldsymbol{A})$ // $\boldsymbol{A} \leftarrow R^{m \times n}$
2: rand$(\chi, m, 1, \boldsymbol{b})$ //$\boldsymbol{b} \leftarrow \chi^m$
3: mult$(\boldsymbol{A}, \boldsymbol{s}, \boldsymbol{c})$ //$\boldsymbol{c} \leftarrow \boldsymbol{A}.\boldsymbol{s}$
4: addTimes$(\boldsymbol{c}, k, \boldsymbol{b}, \boldsymbol{b})$ //$\boldsymbol{b} \leftarrow (\boldsymbol{c} + k.\boldsymbol{b})$
5: mod$(\boldsymbol{b}, q, \boldsymbol{b})$ //$\boldsymbol{b} \leftarrow \boldsymbol{b} \mod q$

multiplication homomorphe et déchiffrement. Ces cinq fonctions doivent toujours être présentes dans la modélisation d'un schéma de chiffrement homomorphe. Si nécessaire, d'autres algorithmes peuvent être disponibles pour certains schémas *p. ex.* relinéarisation (voir section 2.3). Un Homomorphe basique est un enchaînement d'Atomiques et/ou Spécifiques. Propres à un schéma, les algorithmes homomorphes basiques ne sont pas réutilisables dans d'autres schémas, sauf exception. Parfois, une génération de clés est identique dans deux schémas de chiffrement homomorphe *p. ex.* [BLLN13] et [DS16] génèrent leurs clés de chiffrement et déchiffrement de la même façon.

Les algorithmes décrits ci-dessus servent désormais à la modélisation de tout type de schémas de chiffrement homomorphe. D'après notre analyse, un schéma de chiffrement homomorphe peut être vu comme une combinaison de cinq fonctions haut niveau. Ces cinq fonctions seront vues comme des algorithmes homomorphes basiques. Les opérations visibles à bas niveau seront, quant à elles, appelées Atomiques. Et enfin, les séquences d'instructions identifiées dans plusieurs schémas correspondront à des Spécifiques. L'utilisation des Spécifiques permet de rendre la modélisation plus rapide et efficace.

Algorithme 7 *prixTotal*(n_1, p_1, n_2, p_2)

Entrées : n_1, n_2 quantité de produits de prix respectivement p_1 et p_2 achetés.
Sorties : Prix total de l'achat.

1: $sk, pk = KeyGen()$
2: $C_{n_1} = Encrypt(n_1, pk)$
3: $C_{n_2} = Encrypt(n_2, pk)$
4: $C_{p_1} = Encrypt(p_1, pk)$
5: $C_{p_2} = Encrypt(p_2, pk)$
6: $C_{produit_1} = Mult(C_{n_1}, C_{p_1})$
7: $C_{produit_2} = Mult(C_{n_2}, C_{p_2})$
8: $C_{prixTotal} = Add(C_{produit_1}, C_{produit_2})$
9: $prixTotal = Decrypt(C_{prixTotal}, sk)$

Cette méthode permet également la modélisation d'applications dès lors que des schémas sont modélisés. Une application doit pouvoir être représentée comme un enchaînement d'additions et de multiplications. Puis, l'application doit être modélisée sous la forme d'une succession d'algorithmes homomorphes basiques. Par exemple, l'application $prixTotal$, modélisée dans l'Algorithme 7, retourne le prix total de l'achat de n_1 produits de prix p_1 et n_2 produits de prix p_2. Le calcul du prix total est le suivant : $n_1 \times p_1 + n_2 \times p_2$.

Si l'application *prixTotal* est utilisée en pratique, les opérations de la ligne 1 à 5 de l'Algorithme 7 sont effectuées chez l'utilisateur. Les données chiffrées, *c-à-d* $C_{n_1}, C_{n_2}, C_{p_1}$ et C_{p_2}, sont ensuite envoyées sur le serveur distant pour qu'il puisse effectuer les opérations

distriLWE	*WordDecomp*	*addTimes*	*doubleMod*
[BV11b], [BV11a], [BGV12], [Bra12], [BLLN13], [DS16], [FV12], [GSW13], [KGV16]	[Bra12], [FV12], [GSW13], [BLLN13], [BV14], [KGV16] [BGV12], [DS16]	[LP11], [BV11b], [Bra12], [FV12], [DS16], [KGV16], [BV11a], [BGV12]	[BV11a], [CLT14] [CNT12], [CCK+13] [CMNT11], [DGHV10]
pubKey	*WordDecompInv*	*Flatten*	*PowersOf*
[Bra12], [BGV12], [GSW13]	[GSW13], [BV14], [KGV16], [DS16]	[GSW13], [BV14], [DS16]	[BLLN13], [FV12], [GSW13], [BV14] [BGV12], [Bra12]
prodOfAdd	*ctxtInt*	*changeMod*	*prodScalMod*
[BGV12], [FV12]	[DGHV10], [CMNT11], [CNT12]	[FV12], [BLLN13], [KGV16]	[Bra12], [BGV12], [BV14]
doubleMultInv	*compareTo*	*randMultMod*	
[BLLN13], [DS16]	[LP11], [BV14]	[BLLN13], [KGV16]	

TABLE 3.1 – Les 15 algorithmes spécifiques associés à la liste des schémas modélisés qui les utilisent.

des lignes 6 à 8. Enfin, le serveur retourne $C_{prixTotal}$ à l'utilisateur qui peut ensuite déchiffrer chez lui et retrouver le résultat attendu (ligne 9).

L'association des trois types d'algorithmes (atomique, spécifique et homomorphe basique) rend possible la modélisation de tout schéma de chiffrement homomorphe et de toute application visant une utilisation avec de l'homomorphe.

Par ailleurs, un schéma ou une application peuvent être modélisés seulement avec des Atomiques et des Homomorphes basiques. De cette manière, une connaissance des Spécifiques n'est pas requise pour pouvoir modéliser un schéma ou une application. Les algorithmes atomiques ne sont pas modifiables, ce qui n'est pas le cas des Spécifiques qui peuvent être généralisés. Le nombre de paramètres d'entrée et/ou de sortie des algorithmes spécifiques peut être modifié. Par conséquent, l'utilisation des Spécifiques doit être effectuée avec précaution. Suite à la généralisation d'un Spécifique, tous les appels de cet algorithme doivent être revus pour concorder avec sa nouvelle version.

Répartition des fonctions sur plusieurs schémas

Notre méthodologie a abouti sur la création de 11 algorithmes atomiques. Ils représentent les opérations atomiques ou unitaires (sur des entiers, polynômes ou matrices) suivantes : addition, multiplication, soustraction, modulo, tirage aléatoire, produit scalaire, division, arrondi, puissance, inverse et décomposition binaire.

De plus, 25 algorithmes spécifiques ont été créés pour modéliser les 15 schémas présents sur la Figure 3.2. Comme chaque schéma est une combinaison de 5 algorithmes homomorphes basiques, $75 = 5 \times 15$ algorithmes homomorphes basiques ont été modélisés.

Parmi les 25 algorithmes spécifiques créés, 15 sont utilisés dans plusieurs modélisations de schémas : 60 % des algorithmes spécifiques créés sont utilisés dans la modélisation d'au moins deux schémas. La Table 3.1 mentionne les 15 algorithmes en question. Sous chaque algorithme de la Table 3.1 est notée la liste des schémas utilisant cet algorithme.

De façon plus illustrée, la Figure 3.4 donne le nombre d'algorithmes spécifiques partagés entre quatre schémas : FV [FV12], YASHE [BLLN13], F-NTRU [DS16] et SHIELD [KGV16]. Dans cette figure, chaque cercle représente un schéma. Chaque chiffre à l'intersection de plusieurs cercles correspond au nombre d'algorithmes spécifiques

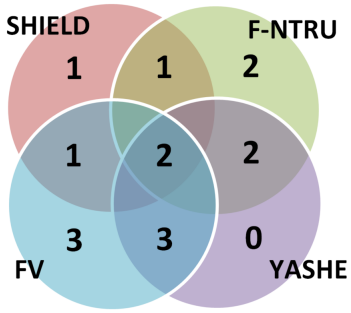


FIGURE 3.4 – Distribution des algorithmes spécifiques entre les schémas FV [FV12], YASHE [BLLN13], F-NTRU [DS16] et SHIELD [KGV16].

partagés entre les schémas. En partant de zéro, les modélisations de FV [FV12], F-NTRU [DS16] et SHIELD [KGV16] entraîne la création de 11 Spécifiques. Parmi ces 11 algorithmes, 4 sont déjà partagés entre les trois schémas. Pour modéliser le dernier schéma YASHE [BLLN13], aucun algorithme spécifique supplémentaire n'a besoin d'être créé. La réutilisation des algorithmes existants rend la modélisation plus rapide.

La méthode de modélisation présentée dans cette section permet de représenter les schémas sous la forme d'une combinaison d'algorithmes atomiques, spécifiques et homomorphe basiques. Sans algorithme spécifique, chaque schéma de chiffrement homomorphe peut être vu comme cinq algorithmes homomorphes basiques chacun composé d'algorithmes atomiques. Dans le but de faciliter la modélisation, des algorithmes spécifiques ont été imaginés : ils correspondent à des séquences d'algorithmes atomiques présents dans plusieurs schémas. L'intérêt est de pouvoir les réutiliser afin accélérer le processus de modélisation sur plusieurs schémas.

3.1.3 Application sur un schéma de chiffrement homomorphe

La méthode générale de modélisation consiste à représenter les schémas de chiffrement homomorphe comme un enchaînement d'algorithmes homomorphes basiques (*p. ex.* génération de clé, chiffrement), spécifiques (*p. ex.* Algorithme 6) et atomiques (*p. ex.* Algorithmes 1 et 2). Cette méthode va être appliquée sur le schéma de chiffrement homomorphe appelé FV [FV12].

Après une présentation générale du schéma FV [FV12], les cinq algorithmes homomorphes basiques (*c-à-d* génération de clés, chiffrement, addition et multiplication homomorphes et déchiffrement) vont être modélisés afin d'illustrer, sans perte de généralité, la méthode proposée.

Définitions du schéma

FV a été créé en 2012 par Fan et Vercauteren [FV12]. Actuellement, il est l'un des schémas les plus compétitifs car son temps d'exécution est l'un des plus intéressant. FV est dit *somewhat* homomorphe (voir section 2.3) et basés sur Ring-LWE (voir section 2.3).

Côté mathématique, les opérations de FV sont réalisées dans $R_q = R/qR$ avec $R = \mathbb{Z}[X]/\langle\Phi_d(X)\rangle$, q est le module et $\Phi_d(X)$ correspond au $d^{\text{ème}}$ polynôme cyclotomique irréductible. Les polynômes de R_q ont un degré maximal de $n = \varphi(d)$ avec φ représentant l'indicatrice d'Euler. Tous les messages clairs sont définis dans $R_t = R/tR$. Dans les schémas, un entier w sert de base pour des décompositions de mots. Les schémas requièrent

également deux distributions gaussiennes χ_{key} et χ_{err} bornées respectivement par B_{key} et B_{err}. Enfin, deux paramètres sont calculés : $l = \lfloor \log_w(q) \rfloor + 1$ et $delta = \left\lfloor \frac{q}{t} \right\rfloor$.

Application de la méthode

La méthode de modélisation va être appliquée sur les cinq algorithmes homomorphes basiques de FV. Ici, nous partirons du principe qu'aucun algorithme spécifique n'existe. Seuls les Atomiques (*p. ex.* modulo, aléatoire, addition, multiplication, soustraction,...) sont disponibles. Ceux-ci pourront permettre la création de Spécifiques si besoin et des Homomorphes basiques.

Algorithme 8 *FV-Add*(q, c_1, c_2)

Entrées : q un entier, c_1, c_2 messages chiffrés
Sorties : (c, d)

1: $c = c_1[0] + c_2[0]$
2: $d = c_1[1] + c_2[1]$
3: $c = [c]_q$
4: $d = [d]_q$

Pour commencer, l'addition homomorphe du schéma FV étant relativement simple, seuls des Atomiques sont nécessaires à sa modélisation. L'Algorithme 8 représente cet algorithme nommé *FV-Add*. Les Atomiques *add* et *mod*, permettant d'effectuer respectivement une addition et un modulo, sont utilisés pour modéliser *FV-Add*.

Algorithme 9 *FV-KeyGen*$(q, w, l, \chi_{key}, \chi_{err}, R, quotient, (sk, pk, evk))$

Entrées : q, w, l entiers, χ_{key}, χ_{err} distributions gaussiennes, R anneau, $quotient$ polynôme quotient
Sorties : $sk, pk = (pk_a, pk_b), evk = (evk_a, evk_b)$

1: $sk \leftarrow \chi_{key}$
2: $pk_a \leftarrow [R]_q$
3: $e \leftarrow [\chi_{err}]_q$
4: $pk_b = pk_a \times sk + e$ } $distriLWE(q, 1, 1, 1, \chi_{err}, R, sk, (pk_a, pk_b))$
5: $pk_b = [[-pk_b]_q]_{quotient}$ } $doubleMod(-pk_b, q, quotient, pk_b)$
6: $pk_a = [pk_a]_{quotient}$
7: $evk_a \leftarrow [R^l]_q$
8: $e \leftarrow [\chi_{err}^l]_q$
9: $evk_b = evk_a \times sk + e$ } $distriLWE(q, 1, l, 1, \chi_{err}, R, sk, (evk_a, evk_b))$
10: $sk_2 = sk \times sk$
11: $sk_3 = PowersOf(sk_2, w, q)$
12: $evk_b = [[sk_3 - evk_b]_q]_{quotient}$ } $doubleMod(sk_3 - evk_b, q, quotient, evk_b)$
13: $evk_a = [evk_a]_{quotient}$

Ensuite, en étudiant l'algorithme de génération de clés de FV, il est possible de remarquer plusieurs séquences d'instructions identiques. Ces séquences ont permis la création des Algorithmes *distriLWE* (6) et *doubleMod*. L'algorithme spécifique *doubleMod* permet d'effectuer le calcul suivant :

$$(a \mod b) \mod c$$

avec a entier, polynôme ou matrice, b et c entiers ou polynômes. En terme de notation, l'opération $a \mod b$ est notée $[a]_b$. Ces deux algorithmes permettent la modélisation l'algorithme de générations de clés *FV-KeyGen*, détaillée dans l'Algorithme 9. Chaque accolade

indique le remplacement d'un groupe d'instructions par un Spécifique. Les instructions isolées, représentant chacune une opération, sont modélisées en Atomiques. L'algorithme nommé *PowersOf* dans *FV-KeyGen* est considéré également comme un algorithme spécifique. Il en est de même pour l'algorithme *WordDecomp* qui est utilisé dans la multiplication homomorphe de FV.

Algorithme 10 *FV-Decrypt*$(q, c, sk, quotient)$

Entrées : q un entier, c le message chiffré, sk la clé privée, $quotient$ polynôme quotient
Sorties : m

1: $m = c[0] + c[1] \times sk$ } $addTimes(c[0], c[1], sk, m)$
2: $m = [m]_q$
3: $m = \left[\left\lfloor \frac{t}{q}.m \right\rceil\right]_q$ } $changeMod(t, q, m, m)$
4: $m = [m]_{quotient}$

En regardant les trois Homomorphes basiques restants *c-à-d* le chiffrement, la multiplication homomorphe et le déchiffrement, des similitudes sont visibles et permettent la création de l'Algorithme *addTimes* (Algorithme 3) et de l'algorithme nommé *changeMod*. Le Spécifique *changeMod*, à partir de t et q deux entiers et m un entier ou un polynôme, effectue le calcul suivant :

$$\left[\left\lfloor \frac{t}{q}.m \right\rceil\right]_q .$$

L'opération $\lfloor a \rceil$ retourne l'entier le plus proche de a. La création de ces deux Spécifiques permet de modéliser entièrement le déchiffrement dans l'Algorithme 10 appelé *FV-Decrypt*. Ce dernier utilise également l'Atomique *mod*.

Algorithme 11 *FV-Encrypt*$(q, \chi_{key}, \chi_{err}, m, pk, quotient, (c_1, c_2))$

Entrées : q entier (module), χ_{key}, χ_{err} distributions gaussiennes, m le message clair, pk la clé publique, $quotient$ polynôme quotient, $delta = \lfloor \frac{q}{t} \rfloor$
Sorties : (c_1, c_2)

1: $u \leftarrow \chi_{key}$
2: $e_1 \leftarrow \chi_{err}$
3: $e_2 \leftarrow \chi_{err}$
4: $c_1 = pk_a \times u + e_1$
5: $c_2 = pk_b \times u + e_2$
} $doubleDistriLWE(pk_a, pk_b, 1, 1, \chi_{key}, \chi_{err}, (c_1, c_2))$
6: $c_1 = c_1 + delta \times m$ } $addTimes(c_1, delta, m, c_1)$
7: $c_1 = [[c_1]_q]_{quotient}$ } $doubleMod(c_1, q, quotient, c_1)$
8: $c_2 = [[c_2]_q]_{quotient}$ } $doubleMod(c_2, q, quotient, c_2)$

La modélisation complète d'un schéma de chiffrement homomorphe ne doit pas être composée seulement d'Atomiques. Le but est de pouvoir créer des Spécifiques et de les réutiliser dans plusieurs modélisations de schémas. À ce stade, les 6 Spécifiques créés jusqu'à maintenant (*c-à-d distriLWE*, *doubleMod*, *PowersOf*, *WordDecomp*, *addTimes* et *changeMod*) ne permettent pas de modéliser efficacement *FV-Mult* et *FV-Decrypt*. Par conséquent, deux nouveaux Spécifiques appelés *doubleDistriLWE* et *prodOfAdd* sont créés. Dans *FV-Mult*, l'opération $\langle a, b \rangle$ correspond au produit scalaire qui est considéré, dans nos travaux, comme un algorithme atomique.

Algorithme 12 *FV-Mult*$(q, w, t, c_1, c_2, quotient)$

Entrées : q, w, t entiers, c_1, c_2 messages chiffrés, $quotient$ polynôme quotient
Sorties : C_1, C_2

1: $ctmp_0 = c_1[0].c_2[0]$
2: $ctmp_1 = c_1[0].c_2[1] + c_1[1].c_2[2]$ } $prodOfAdd(c_1, c_2, (ctmp_0, ctmp_1, ctmp_2))$
3: $ctmp_2 = c_1[1].c_2[1]$
4: $ctmp_0 = \left[\left\lfloor \frac{t}{q}.ctmp_0 \right\rceil\right]_q$
5: $ctmp_1 = \left[\left\lfloor \frac{t}{q}.ctmp_1 \right\rceil\right]_q$ } $changeMod(t, q, (ctmp_0, ctmp_1, ctmp_2), (ctmp_0, ctmp_1, ctmp_2))$
6: $ctmp_2 = \left[\left\lfloor \frac{t}{q}.ctmp_2 \right\rceil\right]_q$
7: $ctmp_2 = [ctmp_2]_{quotient}$
8: $c_3 = WordDecomp(ctmp_2, w, q)$
9: $C_1 = \langle c_3, evk_a \rangle$
10: $C_2 = \langle c_3, evk_b \rangle$
11: $C_1 = ctmp_0 + C_1$
12: $C_2 = ctmp_1 + C_2$
13: $C_1 = [[C_1]_q]_{quotient}$ } $doubleMod(C_1, q, quotient, C_1)$
14: $C_2 = [[C_2]_q]_{quotient}$ } $doubleMod(C_2, q, quotient, C_2)$

doubleMod	*addTimes*	*distriLWE*	*changeMod*
6	2	2	2
WordDecomp	*PowersOf*	*doubleDistriLWE*	*prodOfAdd*
1	1	1	1

TABLE 3.2 – Nombre d'utilisations des algorithmes spécifiques présents dans le schéma FV.

Les Algorithmes 10 et 12 correspondent respectivement à *FV-Decrypt* et *FV-Mult* modélisés avec l'aide des Spécifiques créés. Comme précédemment, les accolades présentes dans les algorithmes regroupent les instructions remplacées par un algorithme spécifique.

Pour la modélisation des cinq Homomorphes basiques, quatre Spécifiques sont utilisés plusieurs fois. La Table 3.2 donne le nombre de fois où les Spécifiques en question ont été employés dans cet exemple. Quatre autres Spécifiques sont présents dans les modélisations mais seulement une seule fois. D'après la Table 3.1, trois de ces algorithmes *c-à-d PowersOf*, *WordDecomp* et *prodOfAdd* peuvent être ré-employés pour la modélisation d'autres schémas parmis les 15 considérés dans ce manuscrit.

Conclusion

Les nombreux schémas de chiffrement homomorphe créés rendent difficile leur comparaison. De plus, l'exécution des schémas avec plusieurs jeux de paramètres d'entrée différents reste fastidieuse du fait des coût importants en matière de temps et d'occupation mémoire. Afin de pouvoir déterminer le schéma le plus intéressant (*p. ex.* ayant la complexité la plus faible) pour une application donnée, il semble de nécessaire de créer une méthode d'analyse qui ne requiert aucune exécution des schémas. Pour cela, les schémas doivent être modélisés dans un format commun.

L'étude de schémas de chiffrement homomorphe de la littérature a montré une ressemblance de séquences d'instructions entre plusieurs schémas. À partir de cette constatation, une méthodologie de modélisation a donc été définie dans le but de faciliter la définition des schémas et leur implémentation future. Notre méthode va permettre de représenter tout schéma sous la forme d'un enchaînement de blocs algorithmiques. Nous avons identifié trois types de blocs algorithmiques : atomique, spécifique et homomorphe basique. Ces blocs algorithmiques sont alors réutilisables entre plusieurs modélisations de schémas. De plus, ils sont plus simples à évaluer indépendamment, cela permet de faciliter l'application de modèles d'évaluation sur un schéma donné.

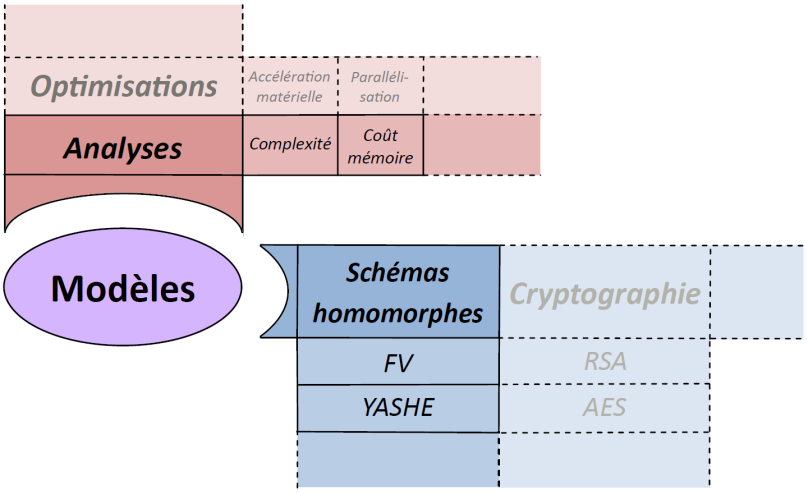


FIGURE 3.5 – Extensions possibles pour les modèles d'évaluation et modèles algorithmiques.

Cette approche générique laisse la porte ouverte à l'extension des différents modèles. En effet, il est possible d'ajouter de nouvelles familles de modèles d'évaluation et/ou modèles algorithmiques. La Figure 3.5 présente un exemple d'ajout de modèles supplémentaires :

- un modèle d'évaluation permettant l'ajout d'optimisations diverses sur l'exécution des modèles algorithmiques.
- un modèle algorithmique introduisant une nouvelle famille d'algorithmes.

La section suivante présente une méthode d'analyse retournant des résultats sur la complexité algorithmique et la consommation mémoire des schémas.

3.2 Analyse théorique des schémas de chiffrement homomorphe

Dans l'optique d'analyser efficacement plusieurs schémas de chiffrement homomorphe, une première méthode permettant la modélisation des schémas a été réalisée. Cette méthode basée sur trois types d'algorithmes (*c-à-d* atomiques, spécifiques et homomorphes basiques) rend possible la modélisation de tout schéma. L'intérêt premier de cette méthode est la réutilisation des algorithmes créés afin de modéliser rapidement tout nouveau schéma de la littérature. L'une des exigences posées avant la réalisation de la modélisation était de pouvoir étendre la méthode afin d'analyser des schémas de chiffrement homomorphe. En effet, chaque algorithme présent dans la bibliothèque d'algorithmes modélisés peut être analysé.

Cette section présente les analyses réalisées sur chacun des algorithmes modélisés. D'abord, les métriques d'analyses sont choisies afin de répondre aux problématiques du chiffrement homomorphe *p. ex.* complexité algorithmique et coût mémoire. Puis, les méthodes appliquées pour les analyses sont décrites pour chaque métrique.

3.2.1 Choix des métriques

D'après la section 2.3.7 présentant les limitations du chiffrement homomorphe, il a été montré que les défis de l'homomorphe sont divers : ils peuvent reposer sur l'algorithmique comme sur la sécurité ou bien la gestion de l'erreur.

Les travaux de recherche présentés dans ce manuscrit sont axés sur l'algorithmique des schémas. Les algorithmes de chiffrement homomorphe sont encore trop gourmands en mémoire et trop coûteux en temps d'exécution. Dans cette section et à partir de la méthode de modélisation, les schémas de chiffrement homomorphe sont analysés en matière de complexité algorithmique et de consommation mémoire dans le pire cas. Le pire cas nous permet d'obtenir une borne supérieure du nombre d'opérations maximal que l'algorithme exécutera.

Ces analyses retournent alors des informations utiles à un expert en chiffrement homomorphe sur les schémas. À partir des analyses, ce dernier peut cibler les opérations coûteuses. Néanmoins, les analyses pire cas ne sont pas toujours représentatives des exécutions concrètes. Autrement dit, une comparaison d'analyses pire cas de différents schémas ne reflètent une comparaison de l'utilisation des schémas au quotidien. Dans la suite des travaux de ce manuscrit, les analyses pire cas sont ramenées dans le cas moyen à l'aide d'une calibration (voir chapitre 4) afin de mieux comparer les algorithmes entre eux.

3.2.2 Représentation des variables

L'un des principaux objectifs de ce chapitre est d'obtenir des analyses de schémas en matière de complexité algorithmique et de consommation mémoire. L'exécution des schémas permet de recueillir les analyses voulus : cependant, cette méthode est fastidieuse. Afin de réaliser différentes analyses, les méthodes présentées dans ce chapitre ne requièrent pas l'exécution exécution des schémas. Autrement dit, le schéma n'a pas besoin d'être exécuté pour être analysé. Cela implique donc que les paramètres et variables manipulés par les algorithmes ne seront pas modifiés en pratique. Afin de réaliser les analyses, des informations sur les paramètres d'entrée et de sortie doivent être sauvegardées car elles sont indispensables à l'analyse. Cette sous-section présente comment les paramètres des algorithmes atomiques, spécifiques et homomorphes basiques vont être représentés.

Jusqu'ici, une nécessité pour les algorithmes est de mettre en paramètre d'entrée le type du paramètre attendu. La valeur seule permet de déterminer s'il s'agit par exemple d'un entier, d'un polynôme ou d'une matrice. Ceci ne sera plus vrai dans la suite de ces travaux et plus particulièrement pour les analyses.

Pour chaque paramètre d'entrée et de sortie d'un algorithme, les informations suivantes sont conservées : type, dimensions (lignes et colonnes) et degré. Les paramètres des algorithmes atomiques, spécifiques et homomorphe basiques possèdent également la valeur des paramètres.

Le *type* correspond au type d'objet dans le paramètre. En l'occurrence, il s'agit soit d'entiers soit de polynômes. *Exemple :* une matrice de polynômes a comme type *polynôme*.

Les *dimensions* permettent d'avoir le nombre de lignes et de colonnes d'un paramètre. Si le paramètre est seulement un entier ou un polynôme, le nombre de lignes et de colonnes est 1.

Le *degré* représente le degré maximal du paramètre. Si le paramètre en question est un entier, alors le degré est 0.

La *valeur* d'un paramètre est l'objet concret à manipuler. La valeur est ce qui est utilisée en entrée dans les algorithmes atomiques, spécifiques et homomorphe basiques. Dans les analyses, cette information est obsolète.

En reprenant l'exemple du paramètre $a = 89$, le type de a est *entier*, les dimensions de a sont (1,1), le degré est 0 et la valeur est 89. Un autre exemple, soit :

$$b = \begin{pmatrix} x+1 & 1 \\ x^3+6 & x^2+9x+8 \end{pmatrix} \tag{3.2}$$

Le type de b est *polynôme*, les dimensions de b sont (2,2), le degré est 3 et la valeur est :

$$\begin{pmatrix} x+1 & 1 \\ x^3+6 & x^2+9x+8 \end{pmatrix} \tag{3.3}$$

Les paramètres d'entrée représentant des anneaux, des corps, des ensembles ou des distributions ne possèdent pas de type, dimensions, degré et valeur. En effet, ces paramètres ne sont pas des variables et donc n'interviennent pas dans le calcul d'une complexité algorithmique et consommation mémoire.

3.2.3 Analyse de la complexité algorithmique

La complexité algorithmique ou *complexité* [1] d'un algorithme peut être estimée dans le pire cas. Pour que cette estimation pire cas soit au plus juste, le nombre d'opérations effectuées au pire cas pendant l'algorithme doit être déterministe. Par exemple, afin de calculer précisément la complexité au pire cas d'un algorithme, celui-ci ne doit pas posséder de boucle ayant un nombre non borné d'itérations non déterministe *c-à-d* qui diffère d'une exécution à l'autre de l'algorithme.

Parmi les schémas cités dans 3.2, seul le schéma [DGHV10] possède une boucle non bornée. Dans ce schéma, les opérations sont effectuées modulo p. La boucle est présente dans la génération de clés : des entiers doivent être échantillonnés à partir d'une distribution tant que le plus grand d'entre eux est impair et que son reste modulo p est pair. La condition étant probabiliste et non bornée, la complexité de cette boucle est difficile à quantifier. Une solution pour gérer ce problème est d'estimer le nombre d'itérations moyen en exécutant plusieurs fois le calcul.

Coron et al. [CMNT11] ont créé une variante du schéma [DGHV10] avec une clé publique compressée. Cette version ne possède plus la boucle non-déterministe décrite dans ce paragraphe.

Les schémas de chiffrement homomorphe possèdent donc que très rarement des boucles non bornées. Afin de pouvoir effectuer une analyse de complexité de schémas de chiffrement homomorphe au pire cas, ce type de boucle peut être éviter soit en estimant un nombre d'itérations de la boucle en moyenne, soit en modifiant le schéma afin de retirer la boucle. L'objectif est de pouvoir analyser plusieurs schémas en fonction de leurs paramètres d'entrée. Ces analyses permettront d'évaluer l'influence des paramètres d'entrée. De plus, elles faciliteront la comparaison des schémas entre eux.

Dans nos travaux, la complexité est le nombre d'opérations effectuées lors d'un algorithme. Ce nombre est déterminé sur la base de 8 opérations : addition, soustraction, multiplication, division, modulo, tirage aléatoire, arrondi et puissance. Ces opérations correspondent aux algorithmes atomiques utiles à la modélisation sauf les algorithmes de l'inverse, du produit scalaire et de décomposition binaire. En effet, la complexité de ces

1. Dans ce manuscrit, la complexité fera toujours référence à la complexité algorithmique.

trois algorithmes peut être calculée avec les 8 opérations citées ci-dessus. Les complexités produites par ces opérations sont différentes si elles sont réalisées avec des entiers ou des polynômes. Par conséquent, la complexité est représentée sous la forme d'un tableau à deux dimensions : une dimension pour les 8 opérations citées et une autre dimension pour le type des objets manipulés (INT pour entiers et POLY pour polynômes). Ce tableau est appelée *le tableau de la complexité* ou *complexity* dans les algorithmes.

TABLE 3.3 – Représentation de la complexité après l'analyse de l'Algorithme 6 appelé *distriLWE* qui retourne une distribution LWE.

	Mult	**Add**	**Sous**	**Div**	**Mod**	**Puis**	**Aléa**	**Arrondi**
INT	$m \times d$	0	0	0	0	0	0	0
POLY	$n \times m$	$n \times m$	0	0	m	0	$(n+1) \times m$	0

Chaque Atomique, Spécifique et Homomorphe basique va être lié à un algorithme d'analyse de la complexité. Ces algorithmes d'analyse ont pour objectif de mettre à jour le tableau de la complexité. La mise à jour consiste en l'incrémentation de certaines cellules en fonction de l'algorithme d'origine. A titre d'exemple, la Table 3.3 montre la représentation de la complexité de l'Algorithme 6.

Une seule cellule du tableau de la complexité est incrémentée lors de l'utilisation d'un algorithme atomique à l'exception de l'inverse, du produit scalaire et la décomposition binaire. En effet, un algorithme atomique ne contient qu'une opération unitaire. Lors de l'utilisation de l'Algorithme 1 nommé *add*, seule une cellule de la colonne *Add* (pour *addition*) sera incrémentée.

Algorithme 13 Complexity.add(a, b, c)

Entrées : a, b entiers ou polynômes ou matrices.
Sorties : c entier ou polynôme ou matrice.

1: n = b.rows() //retourne le # de lignes de a
2: m = b.cols() //retourne le # de colonnes de a
3: D_a = a.degree() // retourne le degré de a
4: D_b = b.degree() // retourne le degré de b
5: complexity[INT][ADD] += $n \times m \times min(D_a + 1, D_b + 1)$
6:
7: T_a = a.type() //retourne le type d'objets contenus dans a
8: T_b = b.type() //retourne le type d'objets contenus dans b
9: **Si** $T_a == INT$ **et** $T_b == INT$:
10: c.type = INT //change le type de c en INT
11: **Sinon** :
12: c.type = POLY //change le type de c en POLY
13: c.degree = $max(D_a, D_b)$ //change le degré de c
14: c.rows = n //change le # de lignes de c
15: c.cols = m //change le # de colonnes de c

L'Algorithme 13 représente l'algorithme d'analyse de l'Algorithme 1 appelé *add*. L'Algorithme 13 commence par la récupération des informations concernant les paramètres d'entrée. Cette phase est effectuée de la ligne 1 à la ligne 5 de l'algorithme. Que les entrées soient des entiers, des polynômes ou des matrices (d'entiers ou de polynômes), le nombre d'additions d'entiers sera toujours $n \times m \times min(D_a + 1, D_b + 1)$ avec D_a (resp. D_b) le degré maximal de l'entrée a (resp. b) et n et m les dimensions des entrées (le nombre de lignes et de colonnes sont identiques pour a et b). Le calcul $min(D_a + 1, D_b + 1)$ retourne

le plus petit nombre maximal de coefficients atteint par a ou b. Une autre possibilité est de différencier deux cas : soit a et b ont tous deux comme type POLY (resp. INT), soit a ou b a le type INT et l'autre le type POLY. Dans le premier cas, la cellule [POLY][ADD] (resp. [INT][ADD]) aurait été incrémentée par $n \times m$ seulement. Dans le second cas, la formule est identique à celle indiquée dans l'Algorithme 13, ligne 5. Les deux manières d'effectuer Complexity.add(a,b,c) sont équivalentes en matière de complexité. Afin d'alléger l'Algorithme 13, la première méthode est décrite dans cet algorithme.

Après l'incrémentation du tableau de la complexité, il est nécessaire de mettre à jour les informations concernant le (ou les) paramètre(s) de sortie. De cette façon, quand ceux-ci seront en entrée d'un autre algorithme, toutes les informations seront disponibles pour faire l'incrémentation du tableau de la complexité.

La seconde phase de l'Algorithme 13 *c-à-d* la séquence d'instructions allant de la ligne 7 à la ligne 15, correspond à la mise à jour du paramètre de sortie c. Pour cela, le type, les dimensions et le degré de c doivent être modifiés. En l'occurrence, le type est INT si les types de a et b sont INT. Les dimensions de c sont identiques à celles de a et b. Enfin, le degré est le degré maximal entre les degrés maximaux de a et b.

Les algorithmes d'analyse des Atomiques sont la base de tous les algorithmes d'analyse. Une fois que les 8 algorithmes d'analyse des atomiques sont créés, l'écriture des algorithmes d'analyse des Spécifiques et des Homomorphes basiques est minime. En effet, un algorithme d'analyse de Spécifiques consiste en le changement des Atomiques en ses algorithmes d'analyses.

Algorithme 14 *Complexity.distriLWE*$(q, n, m, k, \chi, R, \boldsymbol{s}, (A, b))$

Entrées : q, n, m, k entiers, χ distribution gaussienne, R un anneau, $\boldsymbol{s}$ un vecteur de taille n.

Sorties : A matrice de taille $m \times n$, b vecteur de taille m

1: Complexity.rand(R, m, n, A)
2: Complexity.rand($\chi, m, 1, \boldsymbol{b}$)
3: Complexity.mult($\boldsymbol{A}, \boldsymbol{s}, \boldsymbol{c}$)
4: Complexity.addTimes($\boldsymbol{c}, k, \boldsymbol{b}, \boldsymbol{b}$)
5: Complexity.mod($\boldsymbol{b}, q, \boldsymbol{b}$)

L'Algorithme 14 présente l'algorithme d'analyse du Spécifique *distriLWE*. Afin de créer ce nouvel algorithme, toutes les Atomiques et Spécifiques qui composent l'Algorithme 6 ont été remplacés par leurs algorithmes d'analyses.

Conversion du nombre d'opérations sur *polynômes* en nombre d'opérations sur *entiers*

Afin de rendre la comparaison des schémas plus simple, chaque tableau de complexité peut être converti en une complexité totale. Pour cela, les opérations sur les polynômes sont, tout d'abord, converties en opérations sur les entiers.

Certaines opérations telles que l'addition de polynômes sont facilement convertibles en addition d'entiers. En effet, $d+1$ additions d'entiers sont nécessaires pour l'addition de deux polynômes de degré d. Cependant, la conversion est plus difficile pour d'autres opérations telle que la multiplication. En effet, plusieurs procédés sont possibles pour multiplier deux polynômes : naïf, Karatsuba [KO63], Toom-Cook [Too63] encore FFT [2] [Pol71]. Une multiplication de polynômes de degré d a une complexité en $\mathcal{O}(d^2)$ avec l'algorithme naïf, $\mathcal{O}(d^{1.59})$

2. Fast Fourier Transform, en anglais. Transformation de Fourier rapide, en français.

avec l'algorithme de Karatsuba, $\mathcal{O}(d^{1.465})$ avec l'algorithme de Toom-Cook et $\mathcal{O}(d \log d)$ avec l'algorithme de la FFT.

Les travaux par Migliore et al. [MRL+18] montrent que, dans certains cas (dont par exemple un degré de polynôme inférieur à 6144), leur implémentation logicielle/matérielle de Karatsuba a de meilleures performances que l'algorithme de la FFT sur le schéma [FV12].

Parmi les différents procédés listés, nous avons donc choisi de prendre la complexité algorithmique de Karatsuba pour convertir le nombre de multiplication de polynômes en nombre de multiplication d'entiers. Autrement dit, un nombre n de multiplications de polynômes de degré d est donc converti en $n \times d^{1.59}$ opérations sur les entiers. Ce facteur de conversion peut être modifié facilement si l'utilisateur le désire.

Une autre méthode est de modéliser concrètement les algorithmes Karatsuba, Toom-Cook ou FFT dans les schémas de chiffrement homomorphe, en tant qu'algorithme Atomiques. Ces algorithmes atomiques pouvant être ensuite utilisés dans la modélisation des schémas en remplacement des Atomiques de multiplication de polynômes. Chaque multiplication de polynômes serait alors directement estimée en nombre d'opérations sur les entiers.

Conversion du nombre d'*opérations* sur entiers en nombre de *multiplications* sur entiers

Une fois que le nombre d'opérations sur les polynômes est transformé en nombre d'opérations sur les entiers, la dernière étape consiste à convertir chaque nombre d'opérations en nombre de multiplications. Autrement dit, il faut estimer combien valent les opérations addition, soustraction, division, arrondi, tirage aléatoire, puissance, modulo en nombre de multiplications.

TABLE 3.4 – Ratios calculés entre les différents opérations.

Opération	Mult	Add	Sous	Div	Mod	Puis	Aléa	Arrondi
Ratio	1	1.309	1.065	0.115	0.909	0.48	0.53	0.314

Pour cela, les opérations ont été exécutées $N = 10^7$ fois sur des entiers de taille différentes. Ces exécutions ont été effectuées sur une machine équipée d'un processeur Intel Core i5-4310M cadencé à 2.70 GHz et d'une mémoire vive de 8 Go, avec l'utilisation de Sage 7.6 [SAA16], un logiciel libre de mathématiques. La configuration logicielle et matérielle complète de la machine ayant produit ces exécutions est présentée dans la Table 4.1 du chapitre 4.

Calculs de ratios entre les différents opérations

À partir des différentes exécutions de chaque opération, la moyenne, notée m, et la médiane, notée M, des temps d'exécutions ont été calculées. Puis, la moyenne entre m et M a été réalisée. Ce résultat correspond à une moyenne pondérée en mettant un poids de $N+1$ à la médiane avec $N = 10^7$, le nombre d'exécutions des opérations. Cette pondération de la moyenne permet de limiter l'impact des temps d'exécutions aberrants obtenus.

En d'autres termes, chaque opération possède un temps d'exécution équivalent à $t_m = \frac{m+M}{2}$ avec m et M respectivement la moyenne et la médiane des échantillons de temps d'exécutions de cette opération. Les temps d'exécution des opérations sont ensuite normalisés afin de trouver des ratios. Arbitrairement, nous avons pris la multiplication en tant qu'opération référente. Ainsi, la normalisation consiste en la division des temps d'exécutions par le temps de la multiplication. Les ratios finaux sont présentés dans la Table 3.4.

Conversion finale du nombre d'opérations sur les entiers en nombre de multiplications sur les entiers

La Table 3.4 donne les ratios utilisés afin de convertir les nombres d'opérations en nombre de multiplications. À cet effet, chaque nombre d'opérations est donc divisé par le ratio qui lui est associé. Par exemple, 10 moduli d'entiers vont être convertis en $\frac{10}{0.909} = 11.0011$ multiplications d'entiers.

Portée de l'approche

Calculés à partir d'exécutions sur Sage, ces ratios peuvent être re-calculés afin de modifier la conversion de la table de complexité en complexité totale dans le but d'adapter les calculs à la cible pour l'exécution.

Conclusion

Dès lors que la modélisation est terminée, la procédure pour déterminer la complexité d'un schéma S est rapide. Un algorithme d'analyse de S est créé rapidement en remplaçant les Atomiques, Spécifiques et Homomorphes basiques qui le composent par leurs algorithmes d'analyses. L'enchaînement des algorithmes d'analyses permet le remplissage d'un tableau de la complexité. Ce tableau donne le nombre d'opérations que S effectue dans le pire cas pour un jeu de paramètre fixé. Le tableau peut ensuite être converti en une complexité totale en nombre de multiplication d'entiers.

3.2.4 Analyse de coût mémoire

La consommation mémoire d'un algorithme est représentée comme la quantité d'entiers et polynômes stockés. Trois types de coût mémoire sont distincts : le coût mémoire total représentant la somme des entiers et polynômes créés pendant tout l'algorithme, le coût mémoire maximal atteint pendant l'algorithme et le coût mémoire à un instant t. Ces trois cas sont traités par l'analyse mémoire décrite dans cette section.

Chaque type de coût mémoire va être représenté par un tableau contenant les informations des variables temporaires ou de sortie créées pendant un algorithme atomique, spécifique, homomorphe basique ou un schéma. Les informations conservées sur les variables sont leur type, leurs dimensions et leur degré maximal tels que décrit dans la section 3.2.2. Le coût mémoire est donc représenté par trois tableaux, un pour chaque type cité précédemment.

TABLE 3.5 – Représentation de la consommation mémoire après l'analyse de l'Algorithme 6 appelé *distriLWE*, avec R un anneau de polynôme quotienté par $X^{2048}+1$. L'Algorithme 6 retourne une distribution LWE.

Nom	Type	Lignes	Colonnes	Degré maximal
q	INT	1	1	0
n	INT	1	1	0
m	INT	1	1	0
k	INT	1	1	0
s	POLY	n	1	2048
$\boldsymbol{A}$	POLY	m	n	2048
$\boldsymbol{b}$	POLY	m	1	4096
$\boldsymbol{c}$	POLY	m	1	4096

Comme pour l'analyse de la complexité, chaque Atomique, Spécifique et Homomorphe basique va avoir un algorithme d'analyse du coût mémoire. Ces algorithmes d'analyse mettront à jour les informations des variables dans le tableau de la mémoire. La Table 3.5 est une représentation du tableau de la mémoire pour l'Algorithme 6. En partant de l'hypothèse que l'anneau R, mis en entrée de *distriLWE*, est quotienté par $X^{2048}+1$, alors les polynômes générés par celui-ci sont de degré maximal 2048.

Algorithme 15 Memory.add(a, b, c)

Entrées : a, b entiers ou polynômes ou matrices.
Sorties : c entier ou polynôme ou matrice.

1: n = b.rows() //retourne le # de lignes de a
2: m = b.cols() //retourne le # de colonnes de a
3: D_a = a.degree() //retourne le degré de a
4: D_b = b.degree() //retourne le degré de b
5: T_a = a.type() //retourne le type d'objets contenus dans a
6: T_b = b.type() //retourne le type d'objets contenus dans b
7: **Si** $c \notin$ Memory : //si c n'est pas dans le tableau de la mémoire
8: Memory.newline(c) //ajoute une nouvelle ligne au tableau
9: **Si** $T_a == INT$ **et** $T_b == INT$:
10: Memory[c].type = INT //change le type de c en INT
11: **Sinon** :
12: Memory[c].type = POLY //change le type de c en POLY
13: Memory[c].degree = $max(D_a, D_b)$ //change le degré de c
14: Memory[c].rows = n //change le # de lignes de c
15: Memory[c].cols = m //change le # de colonnes de c

Algorithme 16 *Memory.distriLWE*($q, n, m, k, \chi, R, \boldsymbol{s}, (A, b)$)

Entrées : q, n, m, k entiers, χ distribution gaussienne, R un anneau, $\boldsymbol{s}$ un vecteur de taille n.
Sorties : A matrice de taille $m \times n$, b vecteur de taille m

1: Memory.rand(R, m, n, A)
2: Memory.rand($\chi, m, 1, \boldsymbol{b}$)
3: Memory.mult($\boldsymbol{A}, \boldsymbol{s}, \boldsymbol{c}$)
4: Memory.addTimes($\boldsymbol{c}, k, \boldsymbol{b}, \boldsymbol{b}$)
5: Memory.mod($\boldsymbol{b}, q, \boldsymbol{b}$)

L'Algorithme 15 correspond à l'algorithme d'analyse du coût mémoire de l'Atomique *add*. Dans un premier temps, la variable de sortie est ajoutée au tableau si celle-ci n'est pas déjà incluse dedans. Ensuite, les informations de c, variable de sortie, sont mises à jour. Ces mises à jours sont identiques à celles effectuées dans l'Algorithme d'analyse de complexité 13.

Les algorithmes d'analyse de coût mémoire des Spécifiques et Homomorphes basique consistent en l'appel des algorithmes d'analyse *p. ex.* l'Algorithme 16 est celui de l'Algorithme 6. Il est notable que l'Algorithme 16 diffère de l'Algorithme 14 par le nom des appels d'algorithmes : *Complexity* dans l'un et *Memory* dans l'autre.

Le tableau de la mémoire créé comme expliqué ci-dessus va donc permettre d'effectuer un calcul de consommation mémoire total, maximal et à un instant t.

Coût mémoire à un instant t : Ce coût mémoire est représenté par le tableau de la mémoire. Après chaque appel d'algorithme, le tableau évolue. Après chaque appel d'algorithme, il est donc possible de regarder l'état actuel du tableau et donc voir la quantité de ressource mémoire utilisée à un instant t. À la fin de chaque algorithme, les variables temporaires sont supprimées du tableau.

Coût mémoire total : Le coût mémoire total est un autre tableau contenant toutes les variables créées pendant l'algorithme. Si les informations d'une ou plusieurs variables évoluent au cours du temps, alors celles-ci sont modifiées dans le tableau du coût total. Dans ce tableau, toutes les variables temporaires créées sont conservées. Toutes les variables n'étant pas d'entrée ou de sortie d'un algorithme sont temporaires.

Coût mémoire maximal : Ce coût représente le coût maximal atteint pendant l'algorithme. Pour cela, il faut être capable de déterminer si le coût représenté par le tableau de la mémoire à un instant t est plus important que le coût du tableau l'instant $t+1$. Pour ce faire, les informations contenues dans le tableau de la mémoire vont être transformées en un nombre d'entiers. Par exemple, un polynôme de degré maximal 2048 équivaut à 2049 entiers, correspondant aux 2049 coefficients du polynôme.

Dans chaque schéma de chiffrement homomorphe, les opérations sont effectuées modulo un entier q. Cet entier est le référent pour la conversion du tableau de la mémoire en nombre d'entiers. C'est-à-dire que si $q = 2^{100}$, alors le tableau de la mémoire est converti en nombre d'entiers de 100 bits.

Une fois que le tableau à l'instant t et à l'instant $t+1$ sont convertis en nombre d'entiers, il est facile de les comparer et d'estimer lequel contient le coût mémoire le plus important. Une fois la comparaison effectuée, le tableau sélectionné est alors conservé et sera, par la suite, de nouveau comparé après chaque passage dans un algorithme d'analyse Atomique.

Remarque : Un tableau de la mémoire peut donc être converti en un nombre d'entiers de $log(q)$ bits, appelé mem_q. Ce nombre mem_q peut ensuite être modifié afin d'être exprimé en nombre d'entiers de 32 bits de la manière suivante : $mem_{32} = mem_q \times log(q, 2^{32})$.

Conclusion

De la même manière que l'analyse de complexité, l'analyse de coût mémoire est rapide et indépendante des algorithmes d'exécution (atomique, spécifique et homomorphe basique). Suite à la modélisation d'un algorithme, l'algorithme d'analyse de coût mémoire associé peut être créé. La base de l'analyse du coût mémoire est la création des algorithmes d'analyse d'Atomiques. Les algorithmes d'analyse associés aux Spécifiques et Homomorphes basiques sont faciles d'écriture car ils différent peu de leur version exécutable. L'analyse de consommation mémoire permet d'avoir après chaque algorithme le coût mémoire total, maximal et à un instant précis. Deux représentations sont possibles : soit sous la forme d'un tableau contenant le détail des variables créées, soit sous la forme d'un nombre d'entiers.

3.2.5 Métriques secondaires : analyse de bruit, de sécurité

Les méthodes d'analyse de mémoire et de complexité décrites dans les sections précédentes sont similaires. L'analyse est accentuée sur chaque Atomique. Puis, chaque Spécifique et Homomorphe basique possède des algorithmes d'analyses similaires aux algorithmes exécutables. Seul l'appel des algorithmes change : au lieu d'appeler les Atomiques ou Spécifiques exécutables, leur algorithme d'analyse est appelé. Tout autre analyse peut donc être réalisée à partir de la modélisation des schémas de chiffrement homomorphe.

L'ajout d'une analyse doit pouvoir donner des informations intéressantes concernant un schéma de chiffrement homomorphe. Parmi les limitations du chiffrement homomorphe

(voir section 2.3.7), la gestion de l'erreur est un des problèmes majeurs. L'analyse de l'erreur engendrée par un schéma pourrait donc être calculée. Ainsi, les points critiques, *c-à-d* les opérations qui engendrent une erreur importante, pourront être mis en avant. Le schéma pourrait alors être modifié en conséquence afin de limiter l'expansion de l'erreur en ces points.

3.2.6 Résultats d'analyses de schémas

L'un des objectifs fixés de ce chapitre est de pouvoir analyser de manière rapide un schéma de chiffrement homomorphe. Ces analyses permettent alors de déterminer des informations sur plusieurs schémas dont notamment l'impact d'un paramètre sur la complexité/coût mémoire ou encore la comparaison des schémas entre eux. Cette sous-section présente les résultats de la méthode d'analyse présentée précédemment.

Algorithme 17 *cinqHB*(m)

Entrées : m message clair (polynôme aléatoire).
Sorties : $D_{C_m} = (m+m)^2$
1: $sk, pk = KeyGen()$
2: $C_m = Encrypt(m, pk)$
3: $C_m = Add(C_m, C_m)$
4: $C_m = Mult(C_m, C_m)$
5: $D_{C_m} = Decrypt(C_m, sk)$

Trois schémas de chiffrement homomorphe sont analysés dont notamment FV [FV12]. La modélisation de FV a été réalisée entièrement dans la section 3.1. Les deux autres schémas analysés sont YASHE [BLLN13] et F-NTRU [DS16]. Dans cette sous-section, l'analyse d'un schéma correspond à l'analyse de l'application de ces cinq algorithmes homomorphes basiques. Autrement dit, la complexité et la consommation mémoire estimées correspondent à celles engendrées par l'application *cinqHB* décrite dans l'Algorithme 17. Toutefois, il reste possible d'analyser les Homomorphes basiques indépendamment ou tout autre enchaînement que celui proposé.

TABLE 3.6 – Variations des paramètres choisis par schéma pour valider la méthode d'analyse.

Schéma	**Paramètre variant**			**Paramètres fixes**		
	Nom	**Min**	**Max**	$\log_2(q)$	$log_2(w)$	t
FV et YASHE	$\log_2(q)$	100	300	-	2	2
	$\log_2(w)$	2	40	100	-	2
	t	2	40	100	2	-
F-NTRU	$\log_2(q)$	90	140	-	2	-
	$\log_2(w)$	2	40	90	-	-

Afin de tester la méthode d'analyse, nous avons fait varier un paramètre d'entrée à la fois pour montrer l'évolution de la complexité totale et le coût mémoire maximal atteint. Les variations des paramètres d'entrées par schéma sont récapitulées dans la Table 3.6. Trois variables ont été variées : q, w et t. Le module q est présent dans chaque schéma : toutes les opérations des schémas sont effectuées modulo q. De plus, les schémas possèdent une variable nommée w. Cette variable est utilisée pour la décomposition de mot en base w. Enfin, les textes clairs des schémas FV et YASHE sont réduits modulo t.

TABLE 3.7 – Maximum $\log(q)$ en fonction de n pour une sécurité de 80 bits [MBF17].

n	1024	2048	4096	8192
$\log(q)$	46	88	174	348

Les trois schémas sont définis sur $R_q = R/qR$ avec $R = \mathbb{Z}[X]/\langle\Phi_d(X)\rangle$ avec $\Phi_d(X)$ le $d^{\text{ème}}$ polynôme cyclotomique de degré n. Les variables n et q doivent être choisies conjointement afin d'assurer une sécurité de 80 bits. Autrement dit, pour une valeur q fixée, il existe une valeur n telle que la sécurité générée par ces paramètres est de 80 bits. Migliore et al. [MBF17] résument dans un tableau les $\log(q)$ maximum pour un n donné en fonction d'une sécurité λ. Les données de leur tableau pour $\lambda = 80$ sont récapitulés dans la Table 3.7. Ainsi, par exemple, en variant $\log(q)$ pour FV, nous avons pris $n = 2048$ pour tous les $\log(q)$ allant de 100 à 173 inclus.

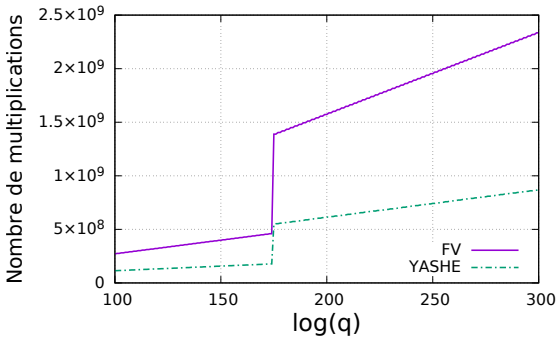
Les résultats des analyses sont illustrés dans les Figures 3.6 et 3.7. Les coupures visibles sur les graphes des schémas FV et YASHE mettent en évidence le changement du degré n. Les calculs pour réaliser les graphiques ont été exécutés en 2 min 25 sec sur une machine équipée d'un processeur Intel Core i5-4310M cadencé à 2.70 GHz et d'une mémoire vive de 8 Go, avec l'utilisation de Sage 7.6 [SAA16] (pour plus de détails sur la configuration de la machine, voir la Table 4.1 du chapitre 4). La complexité est exprimée en nombre de multiplications et le coût mémoire en nombre d'entiers de 32 bits. Le coût mémoire représenté correspond au coût mémoire maximal atteint pendant l'exécution des schémas ; ce coût mémoire est ensuite converti en nombre d'entiers de 32 bits.

Les graphiques montrent trois comportements différents de la complexité et du coût mémoire. En effet, l'augmentation de la valeur de q donne une courbe croissante de la complexité et du coût mémoire pour les trois schémas (Figures 3.6a, 3.6b, 3.7a et 3.7b). Au contraire, plus la valeur de w est grande, plus la complexité et le coût mémoire baissent pour chaque schéma (Figures 3.6c, 3.6d, 3.7c et 3.7d). Cette diminution est plus prononcée pour le schéma F-NTRU concernant la complexité (Figure 3.6d). Enfin, la variable t n'a pas d'impact, que ce soit sur la complexité ou sur le coût mémoire (Figures 3.6c et 3.7c) : en effet, en plus de sa valeur basse, le paramètre t intervient très peu et dans des algorithmes ayant un faible impact dans les schémas, autant sur le plan complexité que coût mémoire.

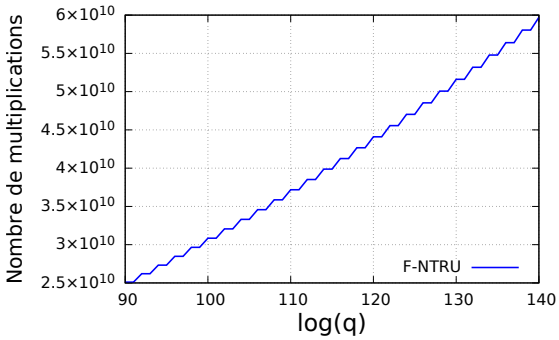
Dans l'optique de comparer les schémas, les graphiques des Figures 3.6 et 3.7 montrent qu'en matière de complexité, F-NTRU et YASHE semblent respectivement le plus et le moins coûteux. F-NTRU est également le plus consommateur de mémoire. Cependant, FV est le moins coûteux en mémoire lorsque q et t varie. Les courbes de FV et YASHE s'entrelacent lorsque w varie (Figure 3.7c) : en effet, FV demande le moins de ressources mémoire pour $\log(w) < 11$ et ensuite, il s'agit de YASHE.

La complexité et le coût mémoire peuvent également être analysés au cours d'un schéma. C'est-à-dire, il est possible d'obtenir la complexité et le coût mémoire après chaque algorithmes d'analyse des Atomiques, Spécifiques et Homomorphes basiques du schéma. Nous avons appliqué cette analyse sur le schéma FV qui est modélisé entièrement dans la section 3.1. Pour ces analyses, nous avons pris les paramètres suivants pour FV : $\log(q) = 100$, $\log(w) = 2$ et $t = 2$.

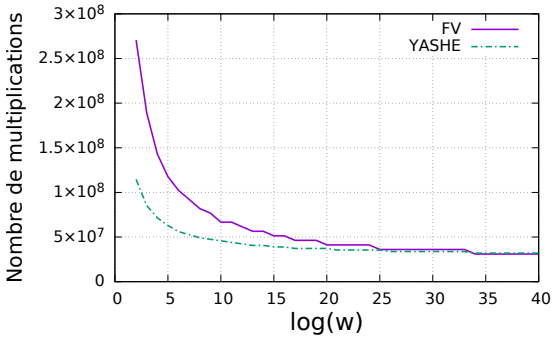
Les résultats d'analyses au cours du schéma FV sont visibles dans la Figure 3.8. Ces résultats sont calculés à partir de la complexité totale et le coût mémoire à un instant t. En effet, l'évolution de la complexité est évaluée en calculant, après chaque algorithme, la complexité totale à partir du tableau de la complexité. L'évolution du coût mémoire est déterminée à partir du tableau de la mémoire à un instant t qui est converti est nombre d'entiers de 32 bits. L'axe des abscisses des graphes de la Figure 3.8 énumère les algorithmes Atomiques, Spécifiques et Homomorphe basiques appelés pendant l'analyse de FV.



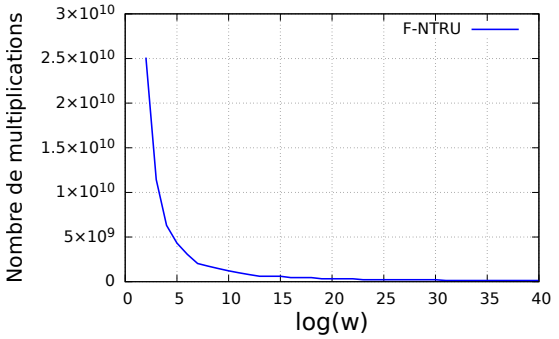
(a) Complexité de FV et YASHE en fonction de q.



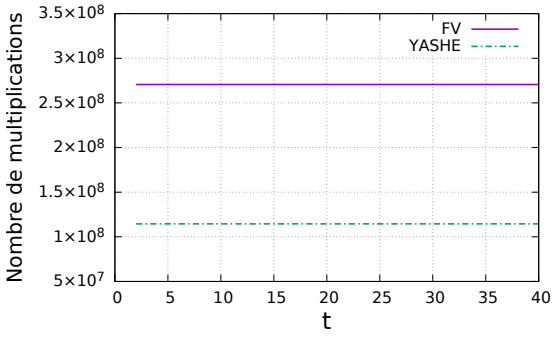
(b) Complexité de F-NTRU en fonction de q.



(c) Complexité de FV et YASHE en fonction de w.

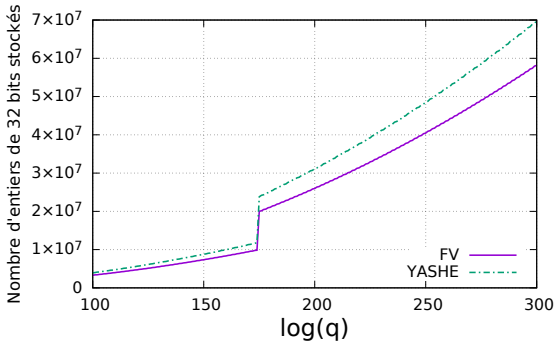


(d) Complexité de F-NTRU en fonction de w.

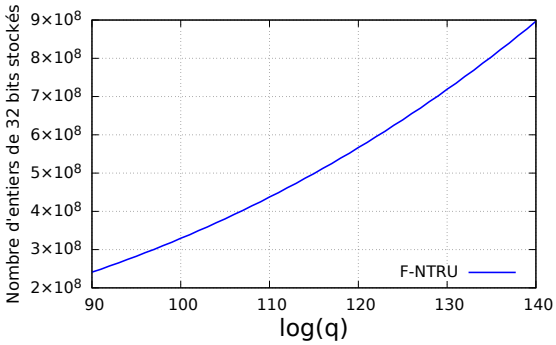


(e) Complexité de FV et YASHE en fonction de t.

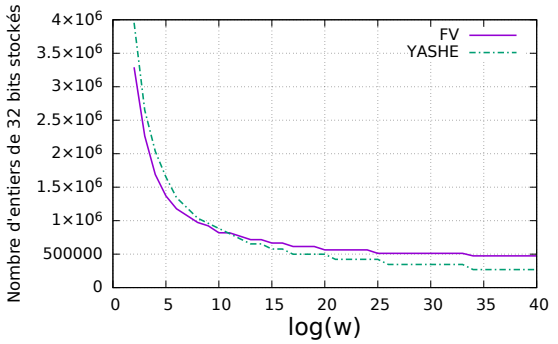
FIGURE 3.6 – Résultats d'analyse de la complexité exprimée en nombre de multiplication.



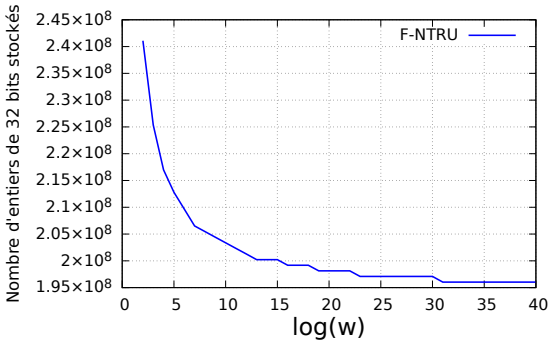
(a) Coût mémoire de FV et YASHE en fonction de q.



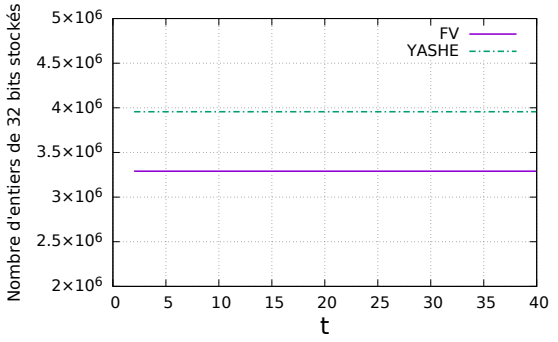
(b) Coût mémoire de F-NTRU en fonction de q.



(c) Coût mémoire de FV et YASHE en fonction de w.

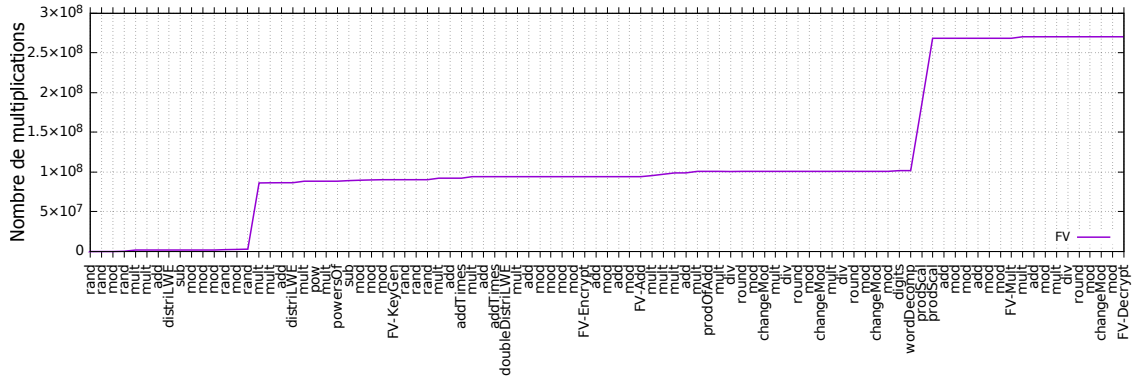


(d) Coût mémoire de F-NTRU en fonction de w.

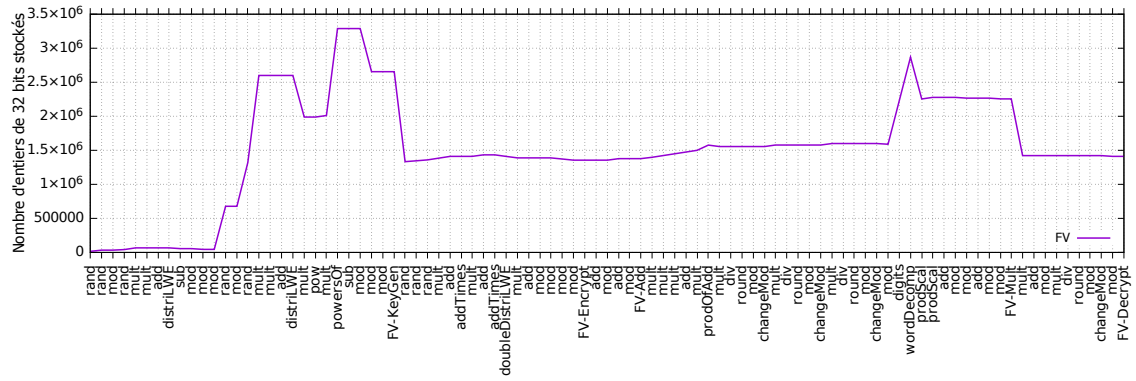


(e) Coût mémoire de FV et YASHE en fonction de t.

FIGURE 3.7 – Résultats d'analyse de la mémoire exprimée en nombre d'entiers de 32 bits stockés.



(a) Évolution de la complexité.



(b) Évolution du coût mémoire

FIGURE 3.8 – Analyse de la complexité et du coût mémoire au cours du schéma FV.

Pour les Figures 3.8a et 3.8b, nous avons récupéré les analyses de complexité et coût mémoire au niveau atomique [3] (regroupant tous types d'algorithmes). Certains Atomiques sont appelées dans les Spécifiques ou Homomorphes basiques. Il en est de même : les Spécifiques sont appelés dans les Homomorphes basiques. Par conséquent, tous les algorithmes compris entre *rand* (le premier algorithme de l'axe des abscisses) et *FV-KeyGen* sont appelés dans *FV-KeyGen*. Puis, les algorithmes compris entre *FV-KeyGen* (exclu) et *FV-Encrypt* sont appelés dans *FV-Encrypt*, etc...

L'évolution de la complexité sur la Figure 3.8a montre que deux opérations engendrent une grande complexité : une Atomique *mult* dans *FV-KeyGen* et les Atomiques *prodScal* dans *FV-Mult*. En effet, d'après l'évolution du coût mémoire, l'algorithme *WordDecomp* a généré des paramètres volumineux. Ces paramètres, utilisés en tant que paramètres d'entrée des *prodScal*, impliquent donc un nombre d'opérations important.

L'évolution du coût mémoire sur la Figure 3.8b révèle que l'Homomorphe basique le plus consommateur de mémoire est *FV-KeyGen* : sur la Figure, cette évolution est visible lors des appels des algorithmes internes à *FV-KeyGen*. C'est d'ailleurs dans cet algorithme que le coût mémoire maximal est atteint (lors de l'appel de *powersOf*). Le point dessiné sur l'abscisse *FV-Decrypt* représente le coût mémoire utile pour stocker les clés, les chiffrés (générés par *FV-Encrypt*, *FV-Add* et *FV-Mult*) et le message déchiffré (généré par *FV-Decrypt*). Le point en l'abscisse *rand*, visible juste après *FV-KeyGen*, correspond aux

3. Nous pouvons également obtenir les résultats au niveau spécifique en récupérant les analyses des Spécifiques et des Homomorphes basiques seulement. Le niveau homomorphe basique comprend que les Homomorphes basiques.

ressources mémoire nécessaires pour stocker les clés et le paramètre créé par *rand*. La comparaison de ces deux coûts mémoire indique que les clés demandent considérablement de la ressource mémoire par rapport aux messages clairs et chiffrés.

Conclusion

Effectuées rapidement ($<$ 3 min), les analyses de complexité et coût mémoire donnent des informations intéressantes sur les schémas. Notamment, la complexité peut être réduite en prenant un grand w. De plus, ces analyses peuvent permettre de cibler des opérations ou séquences d'opérations à optimiser afin de réduire le coût algorithme ou mémoire. Les optimisations peuvent correspondre à l'utilisation d'algorithmes différents (notamment pour la multiplication), à un enchaînement différent des opérations voire à l'accélération des calculs en exécutant sur du matériel (*p. ex.* FPGA[4]) certaines opérations très coûteuses. Cependant, les analyses effectuées le sont dans le pire cas : elle ne représentent pas le cas pratique pouvant montrer dans résultats différents notamment pour la comparaison de schémas. Dans le chapitre suivant, une méthode de calibration va permettre d'obtenir des résultats plus concrets pour une implémentation donnée *p. ex.* la complexité calibrée sera exprimée en secondes.

3.3 Conclusion

Dans ce chapitre, une méthode de modélisation et une méthode d'analyse sont démontrées applicables sur des schémas de chiffrement homomorphe. Les objectifs fixés étaient de pouvoir analyser des schémas préalablement à leur exécution dans le but de comparer plusieurs schémas entre eux et de trouver le schéma ayant les propriétés les plus intéressantes (*p. ex.* une faible complexité algorithmique ou un basse consommation mémoire).

Après avoir modélisé un schéma de chiffrement homomorphe, ce dernier est facilement analysable. Toute fonction créée pour modéliser ce schéma peut bénéficier d'une analyse avancée. Dans ces travaux, chaque schéma modélisé est analysé en matière de complexité algorithmique et de coût mémoire. Toutefois, des analyses secondaires peuvent être réalisées au besoin *p. ex.* l'analyse de l'erreur.

En effet, la méthode de modélisation retenue permet d'intégrer facilement de nouveau modèles d'évaluation ainsi que de nouveaux modèles algorithmiques (Figure 3.5). Toutefois, dans nos travaux, nous ne nous intéressons qu'aux modèles d'évaluation d'analyse de complexité algorithmique et de consommation mémoire.

Ces analyses permettent d'obtenir rapidement des résultats théoriques et dans le pire cas. Par exemple, la complexité totale est exprimée en un nombre de multiplications. Ces résultats donnent des informations sur le schéma. Cependant, ils ne sont pas représentatifs de l'exécution du schéma dans un environnement matériel et logiciel donné en considérant une certaine implémentation.

4. Field-Programmable Gate Array, en anglais. Réseau de portes programmables, en français

Chapitre 4

Applicabilité de l'analyse théorique

Sommaire

Les schémas de chiffrement homomorphe peuvent être représentés sous la forme d'un enchaînement de fonctions *c-à-d* ils peuvent être modélisés à partir d'éléments plus simples. Grâce à cette modélisation, une analyse théorique peut être alors effectuée afin de déterminer la complexité algorithmique et le coût mémoire du schéma. Ces résultats étant théoriques et calculés pour une exécution au pire cas ; ils ne donnent pas d'informations quant à l'exécution sur une cible matérielle donnée ou l'utilisation au quotidien d'un schéma pour une application donnée.

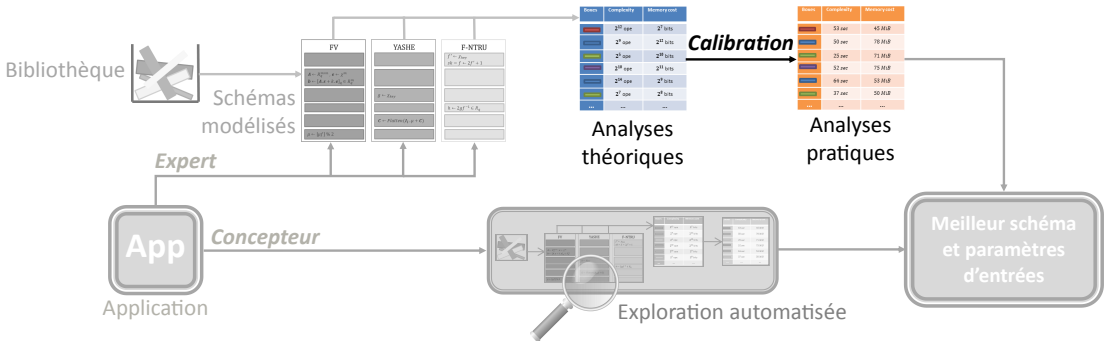


FIGURE 4.1 – Différentes étapes de PAnTHErS avec l'étape de calibration mise en avant.

Ce chapitre rappelle les limites de l'approche précédente et propose une méthode, appelée *calibration*, répondant aux problématiques citées ci-dessus. Cette calibration est positionnée et mise en avant sur la Figure 4.1 qui présente un récapitulatif des étapes de l'outil final PAnTHErS. Dans ce chapitre, la calibration est détaillée et appliquée sur les résultats théoriques du chapitre précédent.

4.1 Limites de l'approche précédente

Une des limitations du chiffrement homomorphe est la difficulté de faire une estimation de la complexité et du coût mémoire des schémas et par conséquent de les comparer entre eux. Avant de pouvoir réaliser des analyses sur les schémas, ils sont d'abord modélisés dans un format commun. Le chapitre 3 présente une méthodologie de modélisation puis une méthode d'analyse de schémas de chiffrement homomorphe. Cette section rappelle brièvement en quoi consistent ces méthodes et expose les limitations des résultats retournés par la méthode d'analyse.

Pour être basés sur un même modèle, les schémas de chiffrement homomorphe sont tout d'abord modélisés sous la forme d'enchaînements d'algorithmes. Trois types d'algorithmes existent : les Atomiques (en général, un Atomique représente une opération), les Spécifiques (représentant des séquences d'instructions, *c-à-d* des séquences d'Atomiques et/ou de Spécifiques, identiques dans plusieurs schémas) et les Homomorphes basiques (correspondant aux cinq fonctions d'un schéma *c-à-d KeyGen, Encrypt, Decrypt, Add* et *Mult*). En modélisant un schéma, les algorithmes créés sont stockés dans une bibliothèque pour pouvoir plus tard être re-utilisés dans d'autres modélisations. Ainsi, plus la bibliothèque est grande, plus il est possible de réutiliser des séquences algorithmiques, ce qui permet d'accélérer la modélisation de nouveaux schémas.

Les schémas ainsi modélisés vont pouvoir être analysés sur une base commune. Un des points faibles des schémas actuels est qu'ils souffrent d'une grande complexité et de coût mémoire important. Il est donc essentiel de pouvoir comparer ces schémas entre eux sur une même base. C'est pourquoi les toutes premières méthodes d'analyses évaluent : la complexité algorithmique et la consommation mémoire. Les résultats de ces analyses

permettront une étude comparative afin de trouver le schéma ayant la complexité la plus faible et/ou le coût mémoire la plus basse.

L'analyse d'un schéma se déroule de la façon suivante. Chaque algorithme A de la bibliothèque, *c-à-d* Atomique, Spécifique et Homomorphe basique, est lié à deux modules d'évaluation : complexité algorithmique et utilisation mémoire. Chaque module possède un tableau de suivi interne. Le module d'analyse de complexité va se charger de mettre à jour son tableau pour suivre l'évolution de la complexité en y ajoutant le nombre d'opérations effectuées dans A. Quant au module d'analyse mémoire, il ajoute et met à jour son tableau pour suivre l'évolution de la consommation mémoire : il y ajoute ou modifie des informations concernant les variables créées et/ou modifiées par A. Les deux modules peuvent ainsi suivre l'évolution de la complexité algorithmique et de l'utilisation mémoire du schéma. En fin d'analyse, les modules de complexité algorithmique et de consommation mémoire retournent respectivement : le nombre de multiplications effectuées lors de l'exécution et le nombre d'entiers de 32 bits stockés simultanément pendant l'exécution du schéma.

Grâce à ces méthodes, un schéma de chiffrement homomorphe peut être analysé sans nécessiter son exécution concrète. Les résultats exposés dans le chapitre 3, présentés sous forme de graphes, montrent l'évolution de la complexité et du coût mémoire en fonction de la variation d'un paramètre du schéma étudié. Ces résultats indiquent l'influence qu'ont les paramètres d'entrée sur le déroulé du schéma analysé. De plus, il est également possible d'observer la complexité algorithmique et la consommation mémoire après chaque appel des algorithmes présents dans le schéma. Ceci permet d'identifier les opérations coûteuses (*p. ex.* les multiplications de polynômes) ainsi que le stockage mémoire des variables temporaires et des sorties (*p. ex.* clés, textes clairs et chiffrés) d'un schéma. Ainsi, ces informations peuvent guider plus finement les concepteurs de l'implémentation du schéma lors du choix d'opérations à optimiser.

Les analyses théoriques mettent en avant l'influence des paramètres d'entrée sur un schéma sans toutefois refléter une exécution concrète de ce schéma sur une cible donnée. Par conséquent, il est difficile d'estimer le temps d'exécution d'un schéma à partir des analyses théoriques de la complexité. Enfin, le caractère *pire cas* des analyses théoriques ne permet pas forcément une représentation satisfaisante des influences concrètes des paramètres lors d'une utilisation pratique des schémas. Par exemple, les analyses pire cas peuvent indiquer qu'un schéma S_1 a une complexité moins importante qu'un schéma S_2 alors que, lors de leur exécution, il est possible que S_2 s'exécute plus rapidement que S_1. Ce cas de figure est présenté plus en détails dans la sous-section 4.3.1. Cette situation n'est pas satisfaisante car l'objectif est de pouvoir comparer plusieurs schémas entre eux. Pour cela, nous avons ajouté une étude permettant d'estimer les temps d'exécution des schémas à partir de nos analyses théoriques : cette étude s'appelle la calibration.

Dans les sections suivantes, une méthode de calibration est présentée. Cette méthode permet de passer d'analyses théoriques à des résultats pratiques en se basant sur quelques exécutions concrètes. Ainsi, la complexité algorithmique (resp. coût mémoire) pire cas est alors calibrée en temps d'exécution (resp. mébioctet). La calibration peut être faite en se basant sur p exécutions : nous parlons donc de p-calibration.

4.2 Présentation de la p-calibration

La p-calibration a pour objectif, à partir d'analyses théoriques et de résultats concrets de p exécutions, de convertir les résultats théoriques en estimations concrètes. Autrement dit, à partir de résultats concrets : temps d'exécution en secondes et consommations mémoire

en mébioctets (Mio), les analyses théoriques au pire cas sont calibrées en secondes (pour la complexité) et en Mio (pour le coût mémoire).

Les résultats concrets pris en référence sont obtenus grâce à l'exécution du schéma analysé. En effet, en exécutant concrètement un schéma, il est possible d'obtenir son temps d'exécution pour certains jeux de paramètres pour une cible fixée. Les exécutions étant effectuées en langage Python, le module Python appelé *memory_profiler* [Ped17] est utilisé afin de récupérer la consommation mémoire du schéma en Mio. Lors de l'exécution d'un algorithme, *memory_profiler* indique la consommation mémoire à chaque ligne de l'algorithme. Les coûts mémoire sont exprimés en mébioctets. Un mébioctet équivaut à 1024^2 octets en comparaison avec un mégaoctet qui équivaut à 1000^2 octets. Le module ne retourne pas seulement la quantité de stockage mémoire utilisée pour les paramètres créés mais aussi le coût mémoire du programme dans son ensemble. En effet, les résultats de *memory_profiler* vont comprendre toutes les instances d'objets créées, les méthodes utilisées par l'algorithme, etc... Ces résultats pratiques seront donc logiquement supérieurs aux estimations théoriques pire cas qui ne comptabilisent que les valeurs des variables créés.

En résumé, les analyses théoriques de la complexité algorithmique représentent le nombre d'opérations pire cas effectuées dans un schéma ou une application. Ce nombre d'opérations pire cas est calibré en un temps d'exécutions en secondes dans un cas moyen (*c-à-d* une utilisation quotidienne). De plus, les analyses théoriques de la consommation mémoire correspondent à la consommation mémoire maximale (pire cas) des paramètres crées pendant l'exécution d'un schéma ou d'une application. Cette proportion de paramètres est calibrée, pour un cas moyen, en une consommation mémoire totale (en Mio) regroupant les paramètres créés et la mémoire requise pour l'exécution du programme.

4.2.1 Notations et cadre expérimental

Dans ce chapitre, nous utilisons les notations suivantes :

• $CompTheo$ qui correspond à la complexité algorithmique théorique exprimée en nombre de multiplications.

• $MemTheo$ qui est la consommation mémoire théorique exprimée en nombre maximum d'entiers de 32 bits stockés.

• $CompPrac$ qui représente le temps d'une exécution en secondes.

• $MemPrac$ qui désigne la consommation mémoire retournée par *memory_profiler* en Mio.

Dans ce chapitre, les expérimentations, telles que les exécutions, sont réalisées avec la configuration logicielle et matérielle présentée dans la Table 4.1. Les temps d'exécution sont calculés grâce à la fonction *time()* de la bibliothèque *time* de Python.

TABLE 4.1 – Configuration matérielle et logicielle de l'environnement d'exécution pour les expérimentations.

	Machine hôte	**Machine Virtuelle**
Processeur	Intel Core i5-4310M @ 2.70 GHz	-
Nombre de cœurs	8	1
Mémoire vive	8 Go / DDR3 PC3-12800 @ 800 MHz	3 Go
Environnement de virtualisation	-	VirtualBox 5.2.8 r121009
Système d'exploitation	Windows 8.1 Entreprise 64-bit	Debian 8.4 64-bit
Environnement logiciel	-	Sage 7.6 Python 2.7.13

4.2.2 Définition de la p-calibration

La méthode de p-calibration vise à convertir un ensemble de résultats théoriques afin qu'ils soient représentatifs de valeurs obtenues suite à une exécution concrète. Les résultats théoriques initiaux sont exprimés, d'un côté, en nombre de multiplications pour la complexité algorithmique et, d'un autre côté, en nombre d'entiers de 32 bits stockés pour la consommation mémoire.

Aussi, il est important que les résultats théoriques convertis soient au plus proche des résultats concrets. Il conviendra donc de choisir une méthode d'adaptation qui minimise au maximum l'écart entre les valeurs concrètes et les valeurs théoriques converties. C'est pourquoi une attention particulière sera apportée sur l'erreur d'approximation des données théoriques converties.

La méthode de p-calibration s'effectue en plusieurs étapes. Tout d'abord, une sélection d'un nombre p de jeux de paramètres est effectuée. Ces jeux de paramètres sont sélectionnés méthodiquement afin de maximiser la couverture de l'espace de valeurs des paramètres, tout en sélectionnant, quand c'est possible, le minimum et le maximum de chaque intervalle.

Une exécution concrète du schéma ou de l'application est effectuée pour chacun de ces p jeux de paramètres. Les résultats obtenus vont ainsi définir p points de référence, qui traduisent chacun le temps d'exécution ($CompPrac$) et la consommation mémoire ($MemPrac$) réelle du schéma ou de l'application, pour ces jeux de paramètres choisis.

Ensuite, les p valeurs $CompPrac_i$ (resp. $MemPrac_i$) obtenues pour les points de référence sont associées avec les p valeurs théoriques $CompTheo_i$ (resp. $MemTheo_i$) obtenues lors de l'analyse théorique (présentée dans le chapitre 3).

Puis, l'objectif est de trouver deux fonctions de conversion f_{comp} et f_{mem} telles que :

$$f_{comp}(CompTheo) = CompPrac \quad \text{et} \quad f_{mem}(MemTheo) = MemPrac. \tag{4.1}$$

Nous avons choisi la forme suivante pour les fonctions de conversion f_{comp} et f_{mem} :

$$\forall\ CompTheo,\quad f_{comp}(CompTheo) = \sum_{i=0}^{p-1} CompTheo^i \times y_i \tag{4.2}$$

$$\forall\ MemTheo,\quad f_{mem}(MemTheo) = \sum_{i=0}^{p-1} MemTheo^i \times y'_i. \tag{4.3}$$

Il suffit ensuite de déterminer les valeurs des p variables y_i et des p variables y'_i, introduits dans les équations 4.2 et 4.3.

Pour f_{comp}, à partir des p points de complexité ($CompTheo_i$, $CompPrac_i$) et pour $p > 1$, la résolution du système 4.4 permet de trouver les valeurs y_i :

$$\begin{cases} \sum_{i=0}^{p-1} CompTheo_1^i \times y_i &= CompPrac_1 \\ \sum_{i=0}^{p-1} CompTheo_2^i \times y_i &= CompPrac_2 \\ \vdots & \\ \sum_{i=0}^{p-1} CompTheo_p^i \times y_i &= CompPrac_p \end{cases} \tag{4.4}$$

Le système 4.4 est également valable pour trouver les valeurs y'_i et, par conséquent, f_{mem}, en remplaçant $CompTheo$ par $MemTheo$ et $CompPrac$ par $MemPrac$.

Une fois que les paramètres y_i et y'_i sont déterminés, les résultats issus de l'analyse théorique peuvent être convertis grâce aux fonctions f_{comp} et f_{mem} décrites dans les équations 4.2 et 4.3. En d'autres termes, la calibration essaye de trouver un polynôme passant par les points de référence en suivant au mieux la courbe des analyses théoriques. Soit p le nombre de points de référence, $p-1$ correspond au degré du polynôme à trouver.

Deux cas particuliers sont à considérer. Dans le cas particulier $p = 1$, les équations à résoudre pour la complexité et le coût mémoire sont les suivantes :

$$\begin{array}{rcrcl} (f_{comp}(CompTheo) & =) & CompTheo \times y & = & CompPrac \\ (f_{mem}(MemTheo) & =) & MemTheo \times y' & = & MemPrac \end{array} \tag{4.5}$$

Un autre cas particulier se présente également lorsque tous les $CompTheo_i$ (ou $MemTheo_i$) sont identiques. Dans ce cas, une moyenne Moy entre les $CompPrac_i$ (ou $MemPrac_i$) est réalisée. Ensuite, la calibration se ramène à une 1-calibration avec $CompPrac = Moy$ (ou $MemPrac = Moy$).

Exigences

Un utilisateur doit pouvoir se fier aux résultats fournis par la p-calibration. C'est pourquoi un taux d'erreur est calculé afin d'estimer l'écart entre les valeurs estimées $CompTheo_{p-cal}$ et les résultats concrets $CompPrac$. Ce taux d'erreur est calculé de la façon suivante :

$$\frac{|CompPrac - CompTheo_{p-cal}|}{CompPrac} \times 100. \tag{4.6}$$

Afin de garantir la fiabilité des résultats théoriques calibrés obtenus, nous fixons les exigences suivantes :

- La moyenne des taux d'erreur ne doit pas dépasser 10 %.
- Au moins 95 % des résultats calibrés doivent avoir un taux d'erreur inférieur à 20 %.

Limitation

La calibration n'a d'intérêt que pour l'évaluation de l'impact de la variation d'un paramètre sur un schéma. Elle permet d'adapter un ensemble de résultats théoriques en se basant sur quelques résultats concrets. Elle permet ainsi d'obtenir un ensemble de valeurs théoriques converties en des résultats concrets, sans toutefois avoir eu à exécuter concrètement le schéma pour chaque jeu de paramètres étudié.

En revanche, si l'objectif d'un utilisateur est d'obtenir des résultats pour un jeu de paramètres unique, il est possible d'obtenir les valeurs de consommation mémoire et de temps d'exécution en exécutant directement le schéma avec le jeu de paramètre fourni. Cependant, ces résultats étant obtenus directement par la simple exécution, la calibration n'aura donc pas d'intérêt pour ce cas de figure.

Modularité

Si un utilisateur souhaite étudier de manière détaillée la consommation en temps et en mémoire des différents algorithmes qui composent un schéma ou une application, il est possible d'exploiter la p-calibration pour effectuer un profilage plus précis. Pour l'évolution d'un paramètre d'entrée donné sur une plage de valeurs (*p. ex.*, le module q), l'utilisateur pourra ainsi obtenir une évolution détaillée, avec des résultats de consommation (temps et mémoire) détaillés individuellement pour chaque algorithme du schéma. Il peut ainsi

mettre en avant, par exemple, les algorithmes les plus impactés par la variation du paramètre d'entrée choisi.

Pour ce faire, l'utilisateur applique la méthode d'analyse théorique sur chaque algorithme de l'application pour toute variation de q (ou autre paramètre d'entrée variant). L'utilisateur exécute ensuite p fois l'application en récupérant les temps d'exécution et les consommations mémoire pour chaque algorithme appelé et il applique la p-calibration pour chaque algorithme de l'application restant. Le nombre de p-calibrations est égal au nombre d'algorithmes présents dans l'application.

TABLE 4.2 – Temps d'exécutions concrets à relever pour la 2-calibration de la complexité de l'application *cinqHB* détaillée.

Type d'algorithme	**Algorithme à analyser**	**Temps d'exécution concret 1**	**Temps d'exécution concret 2**
H. basique	$sk, pk = KeyGen()$	t_{KeyGen_1}	t_{KeyGen_2}
H. basique	$C_m = Encrypt(m, pk)$	$t_{Encrypt_1}$	$t_{Encrypt_2}$
H. basique	$C_m = Add(C_m, C_m)$	t_{Add_1}	t_{Add_2}
H. basique	$C_m = Mult(C_m, C_m)$	t_{Mult_1}	t_{Mult_2}
H. basique	$D_{C_m} = Decrypt(C_m, sk)$	$t_{Decrypt_1}$	$t_{Decrypt_2}$
Application	$D_{C_m} = cinqHB(m)$	$t_1 = t_{KeyGen_1} + t_{Encrypt_1}$ $+t_{Add_1} + t_{Mult_1} + t_{Decrypt_1}$	$t_2 = t_{KeyGen_2} + t_{Encrypt_2}$ $+t_{Add_2} + t_{Mult_2} + t_{Decrypt_2}$

La Table 4.2 donne un exemple de temps d'exécutions concrets à relever pour effectuer une 2-calibration sur les analyses théoriques de complexité d'une application *cinqHB* et des algorithmes qui la composent (voir Algorithme 17 dans la sous-section 3.2.6). Dans l'exemple de la Table 4.2, les lignes 2 à 6 correspondent à la décomposition de l'application ligne 7 au niveau Homomorphe basique.

Dans le cadre d'une 2-calibration des analyses théoriques de l'application (ligne 7 de la Table 4.2), l'application est exécutée pour deux points de référence p_1 et p_2 afin de récupérer les temps d'exécution concrets t_1 et t_2. Par la suite, les analyses théoriques de l'application sont calibrées en analyses concrètes (*c-à-d* en temps d'exécution).

Pour l'analyse de l'application détaillée (lignes 2 à 6 de la Table 4.2), l'application est exécutée pour les points de référence p_1 et p_2 en récupérant les temps d'exécutions concrets de chaque Homomorphe basique appelé dans l'application *c-à-d* en récupérant t_{KeyGen_1}, t_{KeyGen_2}, $t_{Encrypt_1}$, ..., $t_{Decrypt_2}$. Puis, une 2-calibration est effectuée pour chaque Homomorphe basique appelé, soit pour cet exemple, cinq 2-calibrations sont effectuées.

Dans l'exemple présenté dans la Table 4.2, l'application est détaillée à haut niveau (Homomorphe basique). Il est également possible d'appliquer la même procédure pour des niveaux de granularité plus ou moins fins en calibrant l'application respectivement à bas niveau (Atomique) ou à niveau intermédiaire (Spécifique).

Afin pour pouvoir utiliser la p-calibration, l'utilisateur va devoir choisir le nombre de points p à exécuter pour calibrer les analyses théoriques. Ce choix va jouer un rôle sur la qualité de la calibration. La sous-section suivante présente des résultats pour plusieurs p en citant les avantages et inconvénients de ceux-ci.

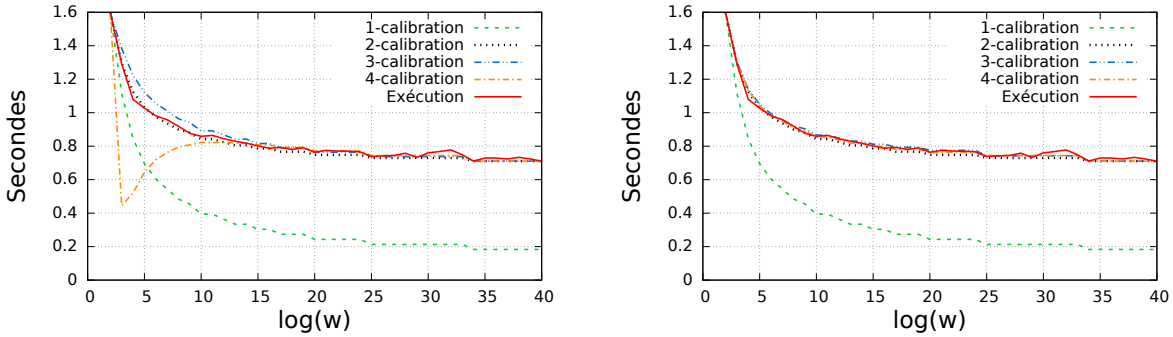
4.2.3 Choix de p et p-calibration alternative

La p-calibration requiert p exécutions afin d'ajuster les analyses théoriques dans le pire cas en analyses concrètes. Plus p est grand, plus le nombre d'exécutions et, par conséquent, le temps d'exécution de la calibration sont importants.

Dans le chapitre 3, l'un des objectifs fixés était de créer des méthodes d'analyse rapides. L'intérêt des méthodes créées est d'obtenir des analyses de schémas rapides et de ne pas avoir à exécuter les schémas pour chaque variation de paramètre.

La méthode de p-calibration va donc allonger le temps d'exécution de la phase d'analyse globale. Le temps d'analyse global inclut les analyses théoriques, les temps d'exécution des p points de référence et la calibration.

L'enjeu est donc de choisir un p assez petit pour que l'exécution de la calibration soit assez faible tout en proposant un taux d'erreur faible sur l'estimation réalisée.



(a) Complexité calibrée (calibration originale) pour l'application *cinqHB* avec le schéma FV en fonction de w.

(b) Complexité calibrée (calibration alternative) pour l'application *cinqHB* avec le schéma FV en fonction de w.

FIGURE 4.2 – Comparaison entre les temps d'exécution concrets de l'application *cinqHB* avec le schéma FV et ses analyses de complexité p-calibrées avec p allant de 1 à 4. À gauche, les résultats sont obtenus avec la p-calibration originale et à droite, avec la p-calibration alternative. Le paramètre $\log(w)$ est varié de 2 à 40 par pas de 1. Les valeurs fixées sont : $\log(q)$ = 100, $n = 2^{12}$ et $t = 2$.

La Figure 4.2a montre les résultats de la p-calibration obtenus pour le schéma FV pour $p = 1, 2, 3$ et 4. Pour obtenir ces résultats, le paramètre $log(w)$ varie de 2 à 40 par pas de 1. La courbe *Exécution* a été réalisée en exécutant FV pour chaque $log(w)$. Les quatre autres courbes illustrent la calibration des valeurs théoriques grâce à l'exécution concrète de certains points, *c-à-d* certaines valeurs de $\log(w)$. Lors de chaque calibration, nous avons fait en sorte que les p valeurs de $\log(w)$ choisies pour l'exécution concrètes soient équitablement réparties, en commençant par sélectionner le minimum et le maximum du paramètre varié $\log(w)$. Pour la 1-calibration, nous avons donc choisi d'exécuter FV avec $\log(w) = 2$, la valeur minimale de $\log(w)$. La 2-calibration a été réalisée en exécutant le schéma avec $\log(w) = 2$ et 40, soit les valeurs minimale et maximale de $\log(w)$. Pour la 3-calibration, nous avons pris $\log(w) = 2$, 21 et 40. Enfin, la 4-calibration a été faite en exécutant FV avec les valeurs $\log(w) = 2, 14, 27$ et 40.

Les courbes pour le schéma FV de la Figure 4.2a montrent l'impact de p sur l'erreur de prédiction au regard des exécutions réelles. L'intérêt de cette figure est de trouver un p :

- limitant l'erreur entre la courbe *Exécution* et la courbe p-calibrée
- assez petit pour avoir un nombre restreint d'exécutions à effectuer.

Avec $p = 1$, l'erreur entre la courbe 1-calibrée et *Exécution* grandit de plus en plus : la différence entre les deux courbes passe de 0.33 seconde pour $\log(w) = 5$ à 0.52 secondes pour $\log(w) = 40$. Autrement dit, le taux d'erreur entre les deux courbes passe de 47.9 % pour $\log(w) = 5$ à 288.1 % pour $\log(w) = 40$. À partir de $p = 2$, les courbes p-calibrées ont une meilleure estimation et se retrouvent proches de la courbe *Exécution*. Néanmoins, la courbe 4-calibrée fait un écart important quand $\log(w) < 5$. En effet, en forçant la courbe à être un

polynôme de degré 4 en passant par des points précis, des "pics" peuvent apparaître sans correspondre à la courbe des analyses théoriques. Ces "pics" faussent alors les résultats, ce qui est confirmé par les taux d'erreur donnés dans la Table 4.3.

TABLE 4.3 – Impact de p sur le taux d'erreur et le temps de la calibration originale pour le schéma FV en faisant varier $\log(w)$ de 2 à 40 par pas de 1.

p	**Pourcentage d'erreur**		**Temps d'exécution de la calibration**
	Moyen	**Maximal**	
1	189.4 %	300.5 %	1.60 seconde
2	2.2 %	6.6 %	2.31 secondes
3	2.7 %	11.7 %	3.10 secondes
4	12.6 %	190.3 %	3.87 secondes

La Table 4.3 donne des valeurs plus précises concernant les différences entre les courbes dessinées sur la Figure 4.2a. Un pourcentage d'erreur a été calculé entre les exécutions concrètes ($CompPrac$) et les analyses calibrées ($CompTheo_{p-cal}$). Pour rappel, le calcul du pourcentage d'erreur utilisé est le suivant :

$$\frac{|CompPrac - CompTheo_{p-cal}|}{CompPrac} \times 100. \tag{4.7}$$

La Table 4.3 donne les taux d'erreur moyen et maximal, et précise également le temps requis pour obtenir les courbes de la Figure 4.2a *c-à-d* le temps d'exécution de la calibration. Ces temps sont dus en grande majorité aux exécutions requises pour calibrer toutes les analyses théoriques.

Sur la Figure 4.2a, il est visible que la courbe des valeurs théoriques 2-calibrées est la plus proche des valeurs de la courbe *Exécution*. La Table 4.3 le confirme en montrant que l'erreur maximal constatée entre les deux courbes est de 6.6 % : la 2-calibration possède ainsi le taux d'erreur maximal le plus faible. De plus, la courbe 2-calibrée obtient également le meilleur taux d'erreur moyen (2.2 %), comparé aux autres résultats p-calibrés. La courbe 1-calibrée possède un taux d'erreur maximal de 300.5 % et un taux moyen de 189.4 %. Ces pourcentages d'erreur sont trop important et montrent qu'il n'est pas envisageable de choisir $p = 1$. Le pourcentage d'erreur moyen (resp . maximum) entre $p = 1$ et $p = 2$ diminue de 187 % (resp. 293 %) alors que le temps de calibration augmente seulement de moins d'une seconde. Puis, pour $p > 2$, le taux d'erreur remonte : 11 % au maximum et 2.7 % en moyenne pour la 3-calibration et 190.1 % au maximum et 12.6 % en moyenne pour la 4-calibration. Les taux d'erreur de la 4-calibration ne remplissent pas les exigences fixées dans la sous-section 4.2.2.

Remarquant que la 2-calibration semble donner des résultats plus intéressants, que ce soit en taux d'erreur ou en temps d'exécution de la calibration, la p-calibration a été révisée dans le but de fournir de meilleurs résultats lorsque $p > 2$.

p-calibration alternative

Soit $p_1, p_2, ..., p_n$ des points de référence tels que $p_i = (CompTheo_i, CompPrac_i)$. Les p_i sont organisés de telle sorte que p_1 soit généré par le paramètre ayant la valeur la plus faible et p_n par le paramètre ayant la valeur la plus grande. La p-calibration alternative consiste à effectuer des 2-calibrations entre p_i et p_{i+1} pour i allant de 1 à $n-1$. Autrement dit, pour chaque couple (p_i, p_{i+1}), la p-calibration alternative va trouver deux variables y_{i_1} et y_{i_2} telles que :

$$\begin{cases} CompTheo_i \times y_{i_1} + y_{i_2} &= CompPrac_i \\ CompTheo_{i+1} \times y_{i_1} + y_{i_2} &= CompPrac_{i+1} \end{cases} \tag{4.8}$$

Ensuite, toutes les analyses théoriques $CompTheo$ comprises entre $CompTheo_i$ et $CompTheo_{i+1}$ sont calibrées avec y_{i_1} et y_{i_2}.

Les cas particuliers de la p-calibration présentée dans la sous-section précédente 4.2.2 restent valables pour la p-calibration alternative.

TABLE 4.4 – Impact de p sur le taux d'erreur et le temps de la calibration alternative pour le schéma FV en faisant varier $\log(w)$ de 2 à 40 par pas de 1.

p	**Pourcentage d'erreur**		**Temps d'exécution de la calibration alternative**
	Moyen	**Maximal**	
1	189.4 %	300.5 %	1.60 seconde
2	2.2 %	6.6 %	2.31 secondes
3	1.4 %	5.4 %	3.10 secondes
4	1.3 %	4.8 %	3.87 secondes

Les résultats de la calibration alternative sont visibles sur la Figure 4.2b et dans la Table 4.4. Avec cette version alternative, le taux d'erreur baisse à mesure que p augmente, donnant ainsi de meilleurs résultats que la p-calibration de base. En effet, la Table 4.4 montre que le taux maximal (resp. moyen) baisse de 300.5 % à 4.8 % (resp. 189.4 % à 1.3 % pour un p allant de 1 à 4. Le temps d'exécution entre les deux types de calibrations sont identiques car ce temps repose principalement sur le temps d'exécution des points de référence requis.

La p-calibration alternative donne des résultats plus satisfaisant que la p-calibration originale. En effet, les pourcentages d'erreur moyens comme maximaux sont plus bas pour la p-calibration alternative (*p. ex.* pour $p = 4$, le taux d'erreur est de 190 % pour la p-calibration originale et de 4.8 % pour la p-calibration alternative). Donnant de meilleurs résultats, la p-calibration alternative est utilisée dans la suite de ce manuscrit. À partir de maintenant, lorsque que nous parlons de calibration, il s'agira de la calibration alternative. Par conséquent, une 3-calibration implique donc en réalité deux 2-calibrations. Par exemple, soit p_1, p_2 et p_3 des points exécutés tels que $p_i = (CompTheo_i, CompPrac_i)$ obtenus respectivement avec $q = 100$, $q = 200$ et $q = 300$. Une 3-calibration consiste à effectuer une première 2-calibration entre p_1 et p_2 et une seconde 2-calibration entre p_2 et p_3. Le point p_2 est utilisée dans les deux 2-calibrations. Par conséquent, les temps d'exécution des points pour une 3-calibration originale et une 3-calibration alternative sont identiques.

Avec la p-calibration alternative, plus p est choisi grand, plus le taux d'erreur est faible. En effet, d'après le graphe de la Figure 4.2b, la différence entre les courbes calibrées et la courbe *Exécution* est de plus en plus faible lorsque p augmente. Cependant, augmenter p rend le temps de la calibration plus long comme indiqué dans la Table 4.4.

En résumé, le choix de p est un compromis entre le taux d'erreur et le temps d'exécution de la calibration. D'un côté, une p-calibration entraîne p fois l'exécution d'un schéma ou d'une application. D'un autre côte, plus p est grand et plus la calibration est précise. Un p trop bas *p. ex.* $p = 1$, entraîne une mauvaise calibration (plus de 180 % d'erreur en moyenne). La valeur de p qui semble la plus raisonnable en matière de pourcentage d'erreur et de temps d'exécution est $p = 2$. En effet, il fournit un taux d'erreur de 2.2 % en moyenne et de 6.6 % au maximum, ce qui satisfait les exigences fixées (sous-section 4.2.2) et convient donc pour une analyse pré-implémentation.

4.3 Applications sur les analyses théoriques

Cette section met en application la p-calibration (alternative) sur les résultats théoriques présentés dans le chapitre 3 *c-à-d* les résultats pour les schémas FV, YASHE et F-NTRU pour des variations de q, w (pour les 3 schémas) et t (pour FV et YASHE). Les résultats exposés dans la sous-section 3.2.6 du chapitre 3 sont repris afin d'être p-calibrés avec $p = 2, 3$ et 4. Ensuite, une évaluation de la p-calibration est réalisée en comparant les différents pourcentages d'erreur générés à partir du nombre p choisi. L'exécution des points de référence nécessaires à la p-calibration a été effectuée en Python avec Sage [SAA16]. La configuration logicielle et matérielle des expérimentations est entièrement détaillée dans la Table 4.1 de la sous-section 4.2.1.

Les résultats de cette section ont été obtenus en effectuant l'analyse de l'application *cinqHB* décrite dans l'Algorithme 17 du chapitre 3. L'application est composée des cinq algorithmes homomorphes basiques appelés une fois chacun [1].

4.3.1 Résultats des valeurs théoriques calibrées

TABLE 4.5 – Valeurs des abscisses choisies pour exécuter les points, en fonction des variations et du nombre de points p.

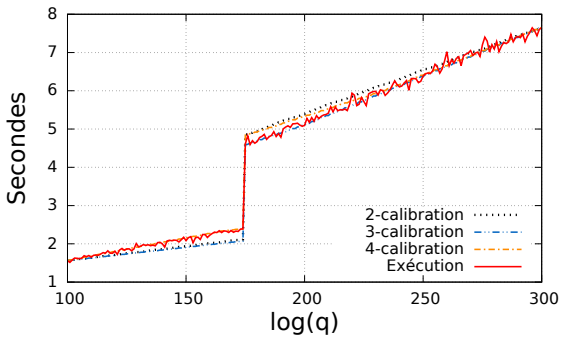
Variations \ p	**2**	**3**	**4**
De 100 à 300	100 / 300	100 / 200 / 300	100 / 166 / 233 / 300
De 2 à 40	2 / 40	2 / 21 / 40	2 / 14 / 27 / 40
De 90 à 140	90 / 140	90 / 115 / 140	90 / 106 / 127 / 140

TABLE 4.6 – Rappel : Variations des paramètres choisis par schéma pour valider la méthode d'analyse.

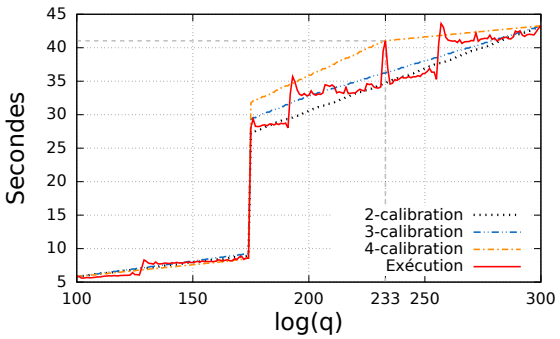
Schéma	**Paramètre variant**			**Paramètres fixes**		
	Nom	**Min**	**Max**	$\log_2(q)$	$log_2(w)$	t
FV et YASHE	$\log_2(q)$	100	300	-	2	2
	$\log_2(w)$	2	40	100	-	2
	t	2	40	100	2	-
F-NTRU	$\log_2(q)$	90	140	-	2	-
	$\log_2(w)$	2	40	90	-	-

Chaque graphe des Figures 4.3 et 4.4 représente la variation d'un paramètre pour un schéma donné. Les schémas étudiés sont FV [FV12], YASHE [BLLN13] et F-NTRU [DS16] avec la variation des paramètres d'entrée q (module des chiffrés) et w (entier de base pour décomposition) pour les 3 schémas et t (module des clairs) pour FV et YASHE. Sur chacune des Figures 4.3 et 4.4, quatre courbes représentent les estimations p-calibrées avec $p = 1, 2, 3$ et 4. La courbe *Exécution* a été obtenue en exécutant le schéma pour chaque valeur du paramètre d'entrée varié. La comparaison de ces cinq courbes permet de valider la méthode de calibration et de la confronter à la méthode d'analyse théorique. La sous-section suivante donne plus précisément les taux d'erreur entre la courbe *Exécution* et chaque courbe p-calibrée.

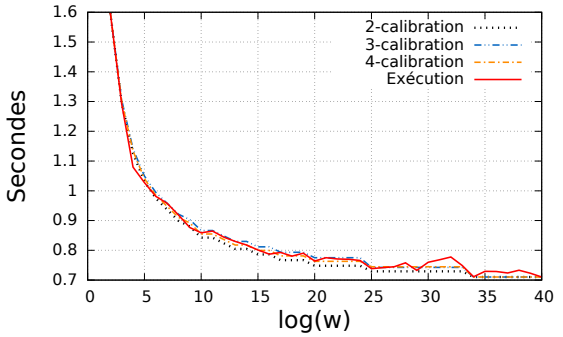
1. La calibration peut toutefois être appliquée sur toute application composée d'Homomorphe basique, Spécifiques et/ou Atomiques.



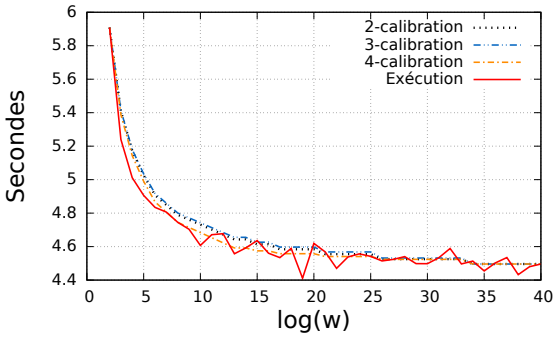
(a) Complexité calibrée pour le schéma FV en fonction de q.



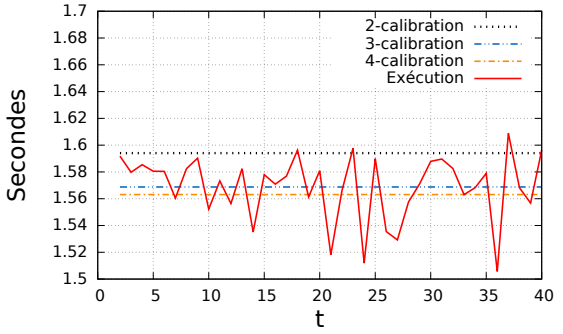
(b) Complexité calibrée pour le schéma YASHE en fonction de q.



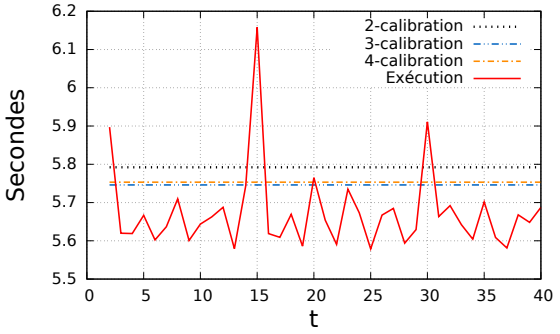
(c) Complexité calibrée pour le schéma FV en fonction de w.



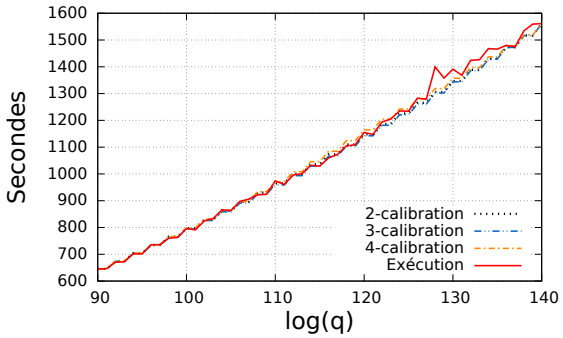
(d) Complexité calibrée pour le schéma YASHE en fonction de w.



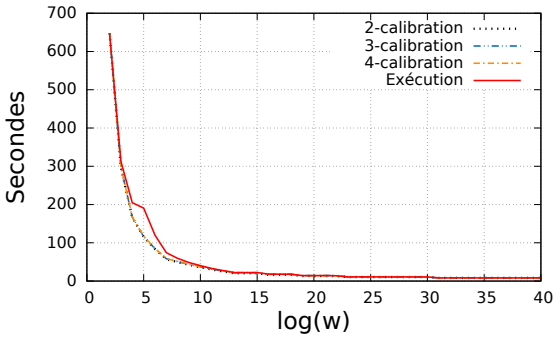
(e) Complexité calibrée pour le schéma FV en fonction de t.



(f) Complexité calibrée pour le schéma YASHE en fonction de t.

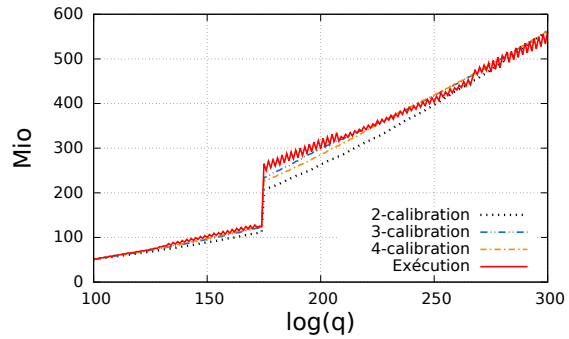


(g) Complexité calibrée pour le schéma F-NTRU en fonction de q.

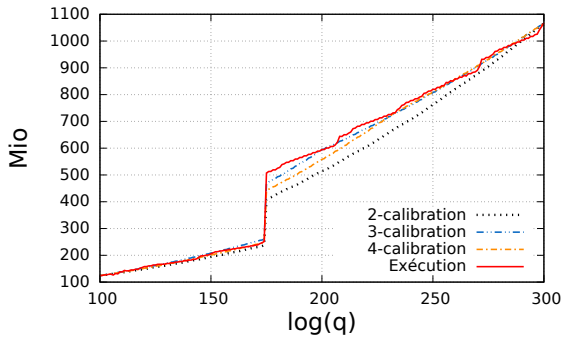


(h) Complexité calibrée pour le schéma F-NTRU en fonction de w.

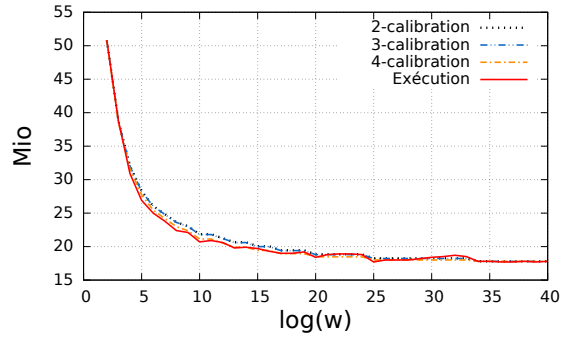
FIGURE 4.3 – Résultats d'analyse de la complexité calibrée exprimée en secondes.



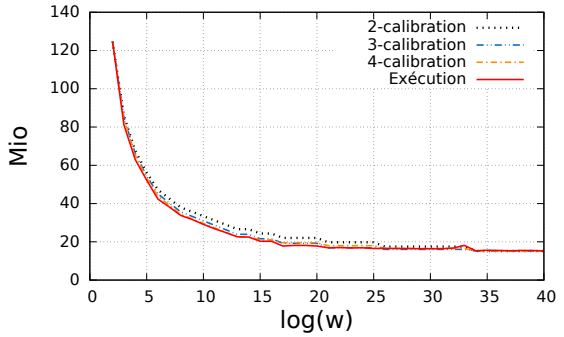
(a) Coût mémoire calibré pour le schéma FV en fonction de q.



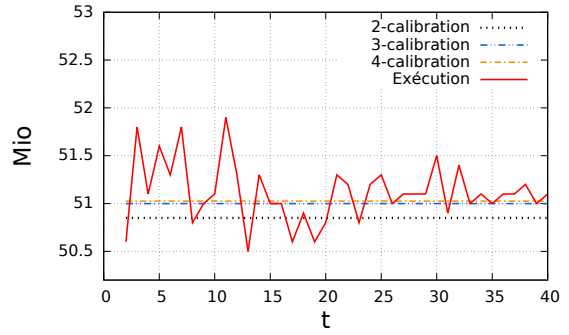
(b) Coût mémoire calibré pour le schéma YASHE en fonction de q.



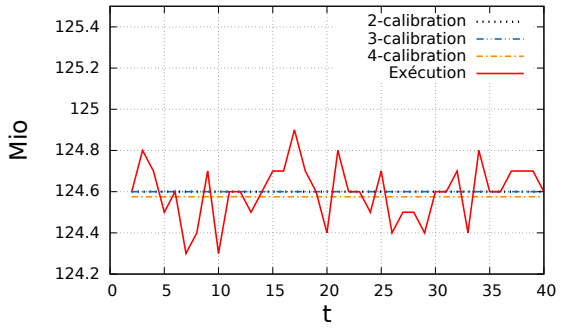
(c) Coût mémoire calibré pour le schéma FV en fonction de w.



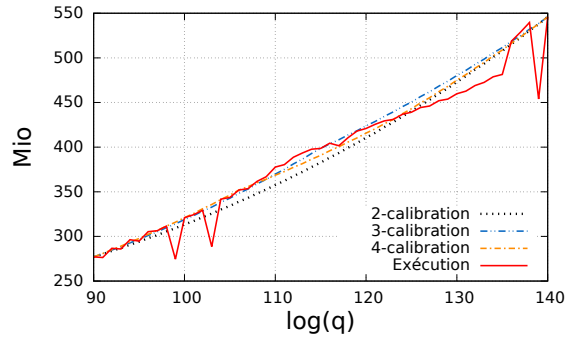
(d) Coût mémoire calibré pour le schéma YASHE en fonction de w.



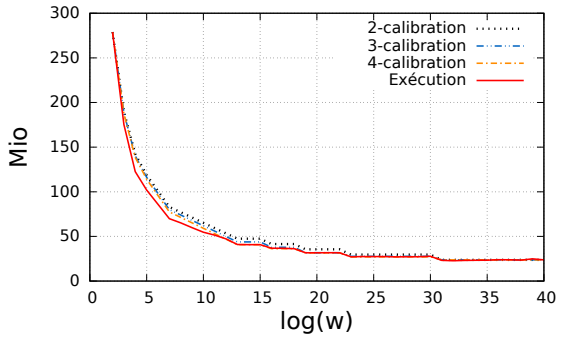
(e) Coût mémoire calibré pour le schéma FV en fonction de t.



(f) Coût mémoire calibré pour le schéma YASHE en fonction de t.



(g) Coût mémoire calibré pour le schéma F-NTRU en fonction de q.



(h) Coût mémoire calibré pour le schéma F-NTRU en fonction de w.

FIGURE 4.4 – Résultats d'analyse du coût mémoire calibré exprimé en Mio.

Les variations des paramètres des schémas pour les analyses de ce chapitre correspondent aux mêmes variations que dans le chapitre précédent et sont rappelées dans la Table 4.6 [2]. Les abscisses choisies pour l'exécution des schémas sont notées dans la Table 4.5. Par exemple, pour la 3-calibration d'un schéma S avec la variation d'un paramètre $param$ allant de 90 à 140, nous avons exécuté le schéma S avec $param = 90$, $param = 115$ et $param = 140$. Comme pour la sous-section précédente, les points de référence ont été choisis de telle manière à ce qu'ils soient équidistants.

De manière générale, l'étude des Figures 4.3 et 4.4 montre que les différentes courbes sur chaque graphe sont proches : en moyenne, 10 % d'erreur séparent les courbes p-calibrées et les courbes *Exécution*. À première vue, les exécutions lors de la variation des t pour FV et YASHE semblent irrégulières : l'axe des ordonnées montrent que ces variations sont finalement minimes (< 1.5 Mio et < 0.9 seconde). Par exemple, pour les variations de la courbe *Exécution* représentant la complexité du schéma FV lorsque t varie, un écart absolu maximal de 0.1 seconde est visible entre l'exécution concrète et les résultats théoriques, soit un écart maximal de 6.3 %. L'écart maximal constaté est compatible avec les exigences fixées dans la sous-section 4.2.2.

Plus précisément, des pics ou écarts sont visibles sur certaines courbes *Exécution*. Ces écarts sont difficilement estimables et peuvent être variables d'une exécution à une autre car dûs aux variations du système sur lequel sont réalisées les exécutions. Cependant, si les abscisses de ces pics sont pris en tant que points de référence pour la calibration, alors ils peuvent augmenter l'erreur entre la courbe *Exécution* et la courbe calibrée.

Par exemple, un écart est visible sur la courbe de la complexité de YASHE en fonction de q (Figure 4.3b). Le point d'abscisse 233 a été pris en référence pour la calibration mais ce point représente une irrégularité de la courbe *Exécution*. Par conséquent, la courbe de la 4-calibration s'écarte des temps d'exécution moyens de la courbe *Exécution*. Le taux d'erreur entre les deux courbes monte alors jusqu'à 16 % autour de $\log(q) = 233$.

Dans l'objectif de réduire ces pics et d'obtenir de meilleurs calibrations, deux possibilités s'offrent à l'utilisateur. La première solution consiste à exécuter un grand nombre de fois les points de référence et prendre une moyenne des temps d'exécution. La seconde consiste à réaliser plusieurs calibrations en prenant des points de références différents.

Analyse de la complexité

L'étude de la complexité théorique du chapitre 3 a montré que les schémas FV et YASHE ont des complexités proches *c-à-d* leurs complexités ont le même ordre de grandeur (10^9 pour $\log(q)$ et 10^8 pour t et $\log(w)$). La Figure 3.6 du chapitre 3 indique que, pour toute variation de paramètres d'entrée, FV a une complexité algorithmique supérieure à YASHE. La calibration donne un tout autre classement pour leurs exécutions concrètes. En effet, les courbes pour les schémas FV et YASHE de la Figure 4.3 indiquent que FV a des temps d'exécution jusqu'à 5 fois plus faible que ceux de YASHE. Ces résultats nous permettent de conclure, qu'en pratique, la quantité de calculs réalisés dans le cas moyen dans le schéma FV est largement inférieure à la quantité de calculs pire cas. Cette différence de quantité d'opérations entre le cas moyen et le pire cas est plus marquée pour le schéma FV que pour le schéma YASHE. Ceci implique que le schéma FV retourne des temps d'exécution plus faibles et donc plus intéressants que YASHE. Cette constatation montre également l'intérêt d'effectuer une calibration pour être plus représentatif d'une utilisation au quotidien pour une application donnée.

Les résultats des analyses théoriques dans le pire cas permettent d'obtenir une idée générale de l'évolution des résultats en fonction pour la variation d'un paramètre, cependant,

2. Ce tableau est identique à la Table 3.6 du chapitre 3.

il est nécessaire de calibrer ces résultats si un utilisateur souhaite les comparer avec ceux d'un schéma différent.

Analyse du coût mémoire

L'étude du coût mémoire théorique, représenté sur la Figure 3.7 du chapitre 3, montre des résultats identiques à celui des graphes de la Figure 4.4. En effet, que les analyses soient théoriques ou calibrées, le classement reste identique : FV semble le moins coûteux en ressources mémoire, YASHE arrive en seconde position et enfin, de F-NTRU est le plus coûteux. De plus, sur la Figure 3.7c du chapitre 3, les courbes théoriques pour les schémas FV et YASHE se coupent lors de la variation du paramètre w : le même phénomène peut être constaté, après calibration des résultats théoriques, en superposant les courbes de coût mémoire calibré de FV et YASHE en fonction de w *c-à-d* en superposant les Figures 4.4c et 4.4d. De plus, le chapitre précédent révèle une importante différence entre F-NTRU et les deux autres schémas : la complexité et le coût mémoire de F-NTRU ont des ordres de grandeurs supérieures à FV et YASHE (10^{10} pour F-NTRU contre 10^9 pour FV et YASHE avec les variations de $\log(q)$). Ce constat est toujours vrai après avoir effectué la calibration.

Dans le cadre d'une comparaison des schémas FV, YASHE et F-NTRU, les analyses théoriques semblent suffisantes afin de déterminer le schéma le moins consommateur de mémoire. Ceci n'est pas le cas pour la complexité où la calibration est requise (voir paragraphe précédent). Néanmoins, pour une analyse du coût mémoire plus précise, la calibration est nécessaire. En effet, même si l'allure générale des courbes est conservée après calibration, il peut subsister un biais dans les résultats : par exemple, les courbes des schémas FV et YASHE se coupent pour $\log(w) = 11$ pour les analyses théoriques alors qu'elles s'intersectent à $\log(w) = 26$ pour les analyses calibrées.

Afin de quantifier plus précisément les écarts constatés entre les différentes courbes des Figures 4.3 et 4.4, la sous-section suivante présente les taux d'erreur moyens et maximaux entre chaque courbe p-calibrées et la courbe *Exécution* associée.

4.3.2 Validation et pourcentage d'erreur

La sous-section précédente expose les résultats obtenus après une p-calibration. Ces résultats sont illustrés sous forme de graphes. Dans cette sous-section, les taux d'erreur visibles entre les courbes p-calibrées et les courbes *Exécution* sont donnés. Cette sous-section compare également les temps d'exécution requis pour effectuer les calibrations avec les temps nécessaires pour exécuter tous les points d'une courbe *Exécution*.

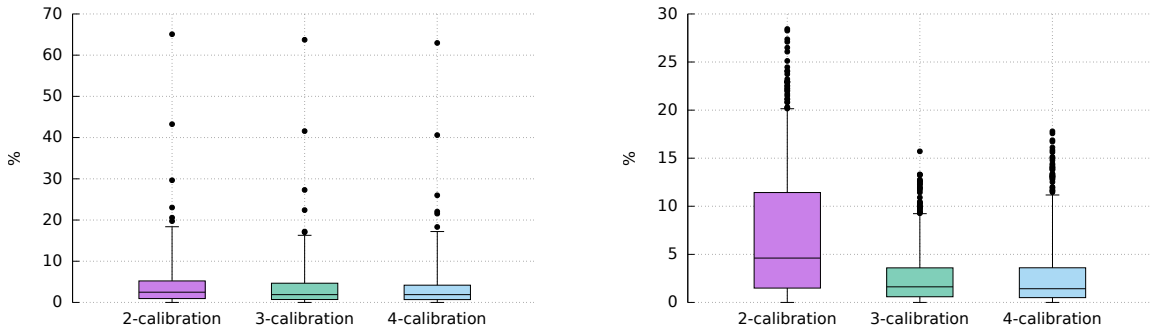
La Table 4.7 présente les taux d'erreur moyens et maximaux entre les courbes p-calibrées (pour $p = 1, 2, 3$ et 4) et la courbe *Exécution* associée au schéma et à la variation de paramètre. Par exemple, la courbe de 3-calibration de FV lorsque q varie (Figure 4.3a) a en moyenne 4.1 % d'erreur avec la courbe *Exécution* du même graphe. Le pourcentage d'erreur maximal atteint entre ces mêmes courbes est de 17 %.

En général, d'après la Table 4.7, le pourcentage d'erreur moyen comme maximal décroît lorsque p augmente. Le plus grand taux d'erreur moyen constaté (complexité calibrée et coût mémoire calibré confondus) est de 9.7 % pour les 2-calibrations et descend à 7.6 % pour les 4-calibrations. Le taux d'erreur maximal le plus important est de 65 % pour le schéma F-NTRU : cette erreur est visible sur le graphe de la Figure 4.3h pour $\log(w) = 5$. Ce taux d'erreur important est dû à une irrégularité dans l'exécution du schéma : $\log(w) = 5$ et $\log(q) = 90$ impliquent un temps d'exécution imprévisible qui ne suit pas la tendance de la complexité algorithmique pire cas. En étudiant de plus près F-NTRU, l'écart est dû aux décompositions en base 5 présentes dans l'algorithme *WordDecomp*. Dans le cas $\log(w) = 5$,

TABLE 4.7 – Taux d'erreur moyens et maximaux entre les courbes p-calibrées et leur courbe *Exécution* associée.

Schéma		**FV**			**YASHE**			**F-NTRU**	
		q	w	t	q	w	t	q	w
Erreur moyenne	$p=2$	4.6	2.2	1.6	4.4	1.0	2.6	1.0	8.7
des résultats	$p=3$	4.1	1.4	1.2	4.6	1.1	1.9	1.1	6.1
complexité (%)	$p=4$	1.8	1.3	1.3	7.6	0.8	2.0	1.0	5.4
Erreur moyenne	$p=2$	9.7	2.0	0.6	7.9	9.5	0.1	3.2	8.7
des résultats	$p=3$	3.6	1.9	0.5	2.1	3.8	0.1	2.4	4.3
coût mémoire (%)	$p=4$	4.0	1.2	0.5	3.1	3.0	0.1	2.3	2.7
Erreur maximale	$p=2$	14.8	6.6	5.5	20.6	3.8	6.3	7.2	65.1
des résultats	$p=3$	17.0	5.4	4.0	16.3	4.0	7.2	7.5	63.7
complexité (%)	$p=4$	7.6	4.8	3.7	21.5	3.2	7.0	6.2	63.0
Erreur maximale	$p=2$	28.4	5.3	2.1	24.1	19.3	0.2	15.6	16.2
des résultats	$p=3$	13.3	5.1	1.8	8.5	12.8	0.2	15.7	12.6
coût mémoire (%)	$p=4$	17.8	3.9	1.7	15.1	9.7	0.3	15.6	11.5

ces décompositions entraînent une augmentation du temps d'exécution pouvant aller jusqu'à 16 secondes par algorithme *WordDecomp* appelé. En écartant les variations de w de F-NTRU, les pourcentages d'erreur maximaux ne dépassent pas 22 % pour la complexité et 20 % pour le coût mémoire.



(a) Répartition des taux d'erreur des analyses de complexité (99 centiles).

(b) Répartition des taux d'erreur des analyses de coût mémoire (95 centiles).

FIGURE 4.5 – Boîtes de Tukey des pourcentages d'erreur des p-calibration pour $p = 2, 3$ et 4, tout schéma confondu.

La Figure 4.5 illustre avec des boîtes de Tukey [Tuk77], la répartition des pourcentages d'erreur en fonction de p. Chaque boîte est composée de :

• un rectangle représentant le premier et le troisième quartile, la médiane coupe ce rectangle en deux,

• deux segments correspondant aux 1$^{\text{er}}$ et 99$^{\text{ème}}$ centiles pour la complexité et aux 1$^{\text{er}}$ et 95$^{\text{ème}}$ centiles pour le coût mémoire,

• des marqueurs sous forme de points désignent les valeurs comprises dans le 1$^{\text{er}}$ centile et entre le 99$^{\text{ème}}$ (ou 95$^{\text{ème}}$) et 100$^{\text{ème}}$ centile.

D'après les exigences fixées en matière d'erreur maximale acceptée pour la p-calibration (voir sous-section 4.2.2), 95 % des taux d'erreur doivent être inférieurs à 20 %. La Figure 4.5a montre que 99 % des résultats ont un pourcentage d'erreur en dessous de 20 % pour

la complexité. Dans le cas du coût mémoire (Figure 4.5b), 95 % des taux d'erreur sont inférieurs à 20.1 % (> 20 % fixés) : ce pourcentage reste toutefois acceptable malgré les exigences. D'après la Figure 4.5, 75 % des résultats ont un taux d'erreur inférieur à 5.2 % (resp. 11.4 %) pour la complexité (resp. le coût mémoire).

Sur les boîtes de Tukey de la consommation mémoire (Figure 4.5b) et la complexité (Figure 4.5a), il est notable que les quartiles et la médiane sont moins importants plus p augmente. Ce phénomène n'est cependant que très peu visible pour le 99$^{\text{ème}}$ centile (resp. 95$^{\text{ème}}$ centile) pour la complexité (resp. le coût mémoire) : la 99$^{\text{ème}}$ (resp. 95$^{\text{ème}}$) centile passe de 18.2 % (resp. 20.1 %) pour la 2-calibration à 16.2 % (resp. 12.61 %) pour la 3-calibration et remonte à 16.9 % (resp. 15.4 %) pour la 4-calibration. Autrement dit, les taux d'erreur entre les estimations de la complexité et les exécutions concrètes ne baissent que légèrement avec une calibration plus fine *c-à-d* avec un p plus grand et donc un temps de calcul plus important.

TABLE 4.8 – Temps d'exécution des courbes de complexité p-calibrées et taux d'accélération par rapport au temps d'exécution de la courbe *Exécution*.

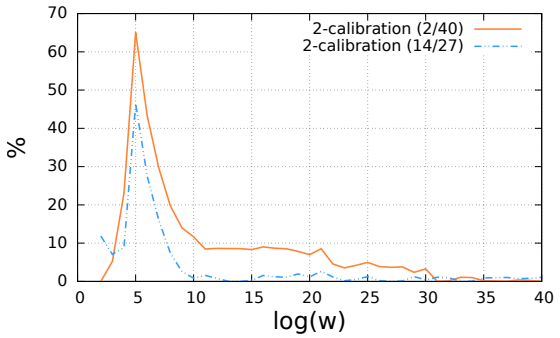
Schéma		**FV**			**YASHE**			**F-NTRU**	
		q	w	t	q	w	t	q	w
Temps analyse complète (sec)	$p=2$	17.44	4.07	5.10	81.63	13.21	14.28	2179.8	646.6
	$p=3$	21.63	4.93	7.80	112.4	17.60	19.02	3195.8	663.1
	$p=4$	25.83	5.82	8.25	130.4	21.14	25.61	4312.3	691.5
Temps *Exécution* (sec)		918.9	32.6	61.20	5114.4	180.9	221.3	54637.3	2104.7
Accélération	$p=2$	52.7	8.0	12.0	62.7	13.7	15.5	25.1	3.3
	$p=3$	42.5	6.6	7.8	45.5	10.3	11.6	17.1	3.2
	$p=4$	35.6	5.6	7.4	39.2	8.6	8.6	12.7	3.0

La Table 4.8 donne les temps d'exécution nécessaires pour la calibration de chaque courbe présente sur les graphes de complexité de la Figure 4.3. Le temps de la calibration comprend : l'analyse théorique, l'exécution des p points de référence et la calibration du reste des points. En réalité, la calibration du reste des points est négligeable en temps (< 0.05 seconde). Les facteurs d'accélération de la Table 4.8 correspondent au ratio entre le temps *Exécution* et les temps des analyses complètes. Autrement dit, un facteur d'accélération a d'une courbe C montre que les valeurs de C ont été estimées a fois plus vite que si toutes les valeurs de C avaient été exécutées concrètement.

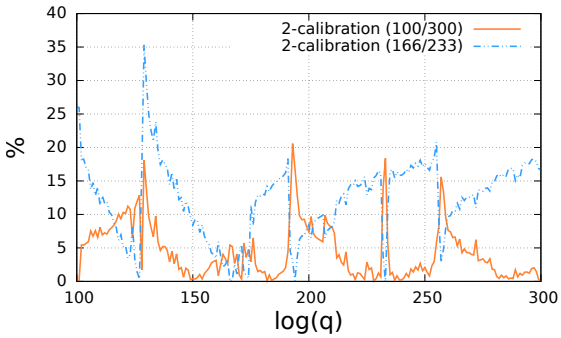
Les temps d'exécution nécessaires à l'obtention des points de calibration pour déterminer le coût mémoire sur les graphes de la Figure 4.4 ne sont pas détaillés : ces résultats sont relativement identiques aux résultats présentés dans la Table 4.8. En effet, seul le temps de l'analyse théorique diffère entre les temps pour la complexité et ceux du coût mémoire. Les temps d'analyse de la complexité sont toutefois proches de ceux du coût mémoire. En effectuant une moyenne des accélérations, la 2-calibration engendre une accélération de 24.1 pour la complexité et de 21.8 pour le coût mémoire. La moyenne des accélérations de la 4-calibration est de 15.1 pour la complexité et de 13.9 pour le coût mémoire.

Les facteurs d'accélérations permettent de rendre compte du gain de temps entre l'analyse et la calibration des schémas et leurs exécutions concrètes. Ces facteurs diffèrent en fonction :

• de l'intervalle de variation des paramètres d'entrée. Par exemple, en considérant une 2-calibration $Calib_1$ pour la variation d'un paramètre allant de 10 à 15 par pas de 1 et une 2-calibration $Calib_2$ pour une variation allant de 10 à 1000 par pas de 1, le taux d'accélération sera plus important pour la calibration $Calib_2$. En effet, le nombre de jeux de paramètres à



(a) Évolution du taux d'erreur entre la courbe *Exécution* et les courbes 2-calibrées de F-NTRU en fonction de w.



(b) Évolution du taux d'erreur entre la courbe *Exécution* et les courbes 2-calibrées de YASHE en fonction de q.

FIGURE 4.6 – Évolution du taux d'erreur des estimations de complexité de F-NTRU et YASHE (en fonction de resp. w et q) 2-calibrées en prenant des points de référence différents, précisés entre parenthèses dans les légendes.

exécuter pour obtenir le temps *Exécution* de la $Calib_2$ est 165 fois plus important que celui pour la $Calib_1$.

• du temps d'exécution des schémas. En effet, d'après la Table 4.8, les facteurs d'accélérations diffèrent d'un schéma à l'autre. L'accélération peut aller jusqu'à 62.7 pour YASHE contre seulement 3.0 pour F-NTRU.

• du nombre p choisi. Par exemple, l'accélération est moins important si p est choisi grand : ceci est dû au temps requis pour l'exécution des p points de référence. La courbe illustrant la complexité estimée de F-NTRU en fonction de w (sur la Figure 4.3h) a le plus petit facteur d'accélération d'après la Table 4.8. En effet, l'un des points de référence exécuté pour cette courbe a le temps d'exécution le plus important. Puis, la courbe décroît très rapidement pour passer de 646.6 secondes à 8.3 secondes. En choisissant d'autres points de référence, le rendement aurait pu être plus important. Cependant, un autre choix de points de référence a des conséquences sur les taux d'erreur, comme expliqué ci-dessous avec deux exemples illustrés par la Figure 4.6.

Pour rappel, les résultats obtenus après la calibration ont été calculés à partir d'une sélection de points de référence (détaillée dans la Table 4.5). Le choix de ces points de références peut différer d'un analyse à l'autre. En fonction des points de référence de départ choisis, les pourcentages d'erreur et les temps d'exécution peuvent être différents : ils peuvent être plus avantageux comme moins satisfaisants.

• des points de référence exécutés et l'impact du paramètre varié sur les temps d'exécution de l'application. Deux exemples sont donnés ci-dessous.

La Figure 4.6 est composée de deux graphes dont chacun illustre les pourcentages d'erreur obtenus après deux 2-calibrations d'un schéma en fonction d'un paramètre fixé. Ces deux 2-calibrations ont été calculées à partir de deux points de référence différents. Les points de référence sélectionnés sont précisés entre parenthèses dans les légendes des graphes.

Afin d'illustrer ce phénomène, le graphe de la Figure 4.6a, pour F-NTRU, montre que le pourcentage d'erreur est plus faible en prenant comme points de référence $\log(w) = 14$ et 27. Le taux d'erreur maximal descend de 65.1 % à 46.5 % et le taux d'erreur moyen de 8.7 % à 3.9 %. De plus, la courbe 2-calibration (14/27) est calculée en seulement 35.9 secondes comparée à la courbe 2-calibration (2/40) réalisée en 646.6 secondes. Autrement dit, la courbe 2-calibration (14/27) offre une accélération de 58.6 contre seulement 3.3 pour la

courbe 2-calibration (2/40). Pour le schéma YASHE, sur la Figure 4.6b, la 2-calibration obtenue avec les points $\log(q) = 100$ et 300 engendre moins d'erreur (4.5 % en moyenne) que la 2-calibration calculée avec $\log(q) = 166$ et 233 (12.4 % d'erreur en moyenne). Toutefois, les deux courbes ont un temps d'exécution quasiment identiques (seulement 0.4 seconde de différence entre les deux temps).

Comparés aux analyses théoriques, les résultats retournés par la p-calibration permettent d'obtenir des estimations de temps d'exécution (secondes) et de coût mémoire (Mio) des schémas de chiffrement homomorphe. De manière générale, en prenant un grand p, la p-calibration demande un temps d'exécution plus long. Cependant, elle rend des résultats plus fins et donc plus proches des temps d'exécution concrets et des consommations mémoire concrètes.

La Table 4.7 regroupant des taux d'erreur obtenus et la Figure 4.6 illustrant des boîtes de Tukey, présentant les répartitions des taux d'erreur, montrent que 99 % des points calibrés pour la complexité et 95 % des points calibrés pour le coût mémoire ont un taux d'erreur de respectivement 20 % et 20.1 %. De plus, 75 % des points calibrés ont un taux d'erreur de 11.4 % au maximum pour le coût mémoire et 5.2 % pour la complexité.

Les p-calibrations sont réalisées plus rapidement que si chaque jeu de paramètre avait dû être exécuté indépendamment. En effet, en moyenne, la 2-calibration offre une accélération de 24.1 pour la complexité et 21.8 pour le coût mémoire. Cette accélération est moins importante avec un p plus grand. Celle-ci est, pour la complexité (resp. consommation mémoire), en moyenne, de 18.1 (resp. 16.6) pour une 3-calibration et de 15.1 (resp. 13.9) pour une 4-calibration.

Enfin, la p-calibration se base sur des exécutions de points de référence. Ces points de référence sont choisis par l'utilisateur. Pour une même calibration, le choix des points de référence peut avoir un impact sur les résultats finaux. Bien que que les résultats soient distincts, les pourcentages d'erreur engendrés restent toujours à 20 % au maximum. Les temps de calibration, quant à eux, différent également en fonction des points de référence choisis. L'accélération peut donc être variable en fonction des choix établis précédemment. Dans le cas de la variation de w pour F-NTRU, l'accélération pour la courbe 2-calibration (2/40) est de 3.3 et passe à 58.6 pour la courbe 2-calibration (14/27).

4.4 Conclusion

L'approche du chapitre précédent consiste à analyser théoriquement des schémas de chiffrement homomorphe. Bien qu'intéressante, cette méthode ne donne aucune indication concernant l'exécution concrète du schéma pour une implémentation donnée. La p-calibration décrite dans ce chapitre montre la possibilité de faire une estimation du temps d'exécution d'un schéma visant, par exemple, une implémentation Python sur un processeur Intel Core i5 cadencé à 2.70 GHz. Rendant les analyses théoriques plus exploitables, la p-calibration permet d'obtenir une analyse concrète des schémas de chiffrement homomorphe à un certain pourcentage d'erreur près.

Les résultats notables de ce chapitre sont les suivants :

• En testant la p-calibration pour différents valeurs de p, nous constatons que le taux d'erreur diminue lorsque p augmente (voir Table 4.4 et Figure 4.5). Cependant, une augmentation du nombre d'exécutions concrètes p implique un coût plus important en temps de calibration (voir Table 4.4) et réduit le facteur d'accélération (voir Table 4.8). C'est pourquoi, dans un souci de compromis temps/efficacité, nous avons sélectionné la plus petite valeur de p pour laquelle nos exigences sont satisfaites (voir sous-section 4.2.2), soit $p = 2$.

• D'après les expérimentations effectuées (section 4.3), nous avons remarqué que, d'un côté, les analyses théoriques du coût mémoire peuvent permettre une comparaison directe des schémas FV, YASHE et F-NTRU. Néanmoins, afin d'avoir des résultats d'analyse plus précis, il est préférable d'utiliser la calibration. D'un autre côté, la calibration est indispensable dans l'optique d'une comparaison des temps d'exécution entre plusieurs schémas.

• Lors des applications concrètes (section 4.3), les points de référence ont été choisis de telle sorte qu'ils soient équidistants et en s'assurant de prendre en premier le minimum et le maximum du paramètre varié. Néanmoins, un autre choix de points de référence peut affecter les taux d'erreur générés mais également le temps d'exécution de la calibration. Une perspective d'évolution de la p-calibration serait de déterminer un ensemble de points de référence optimal *c-à-d* minimisant au mieux le taux d'erreur et le temps d'exécution.

Les méthodes de modélisation et d'analyse peuvent dorénavant être automatisées afin d'effectuer des analyses complètes de schémas et d'applications utilisant du chiffrement homomorphe. Cependant, même automatisées, ces méthodes sont utilisables seulement par un expert en chiffrement homomorphe. En effet, une connaissance des schémas est requise pour savoir faire varier leurs paramètres d'entrée afin de respecter, entre autres, une bonne sécurité et une profondeur multiplicative adaptée à l'application ou au besoin de l'utilisateur. Pour palier cela, le chapitre suivant présente une méthode d'exploration automatisée permettant à tout utilisateur, expert ou non, de déterminer le schéma et les paramètres d'entrée les plus adaptée pour une application. Cette exploration prend seulement en entrée l'application à analyser et ne requiert pas l'intervention de l'utilisateur ensuite. Autrement dit, l'utilisateur n'a pas besoin de choisir les variations de paramètres d'entrée et donc de connaître les schémas qui seront utilisés.

De plus, la calibration ne permet pas d'obtenir l'évolution du temps d'exécution et du coût mémoire pour chaque algorithme utilisé dans une application donnée pour un jeu de paramètres d'entrée fixé. Autrement dit, il n'est pas possible d'estimer les temps d'exécution et la consommation mémoire de chaque algorithme homomorphe basique, spécifique et/ou atomique qui compose l'application ou le schéma en fixant un jeu de paramètres d'entrée. Toutefois, il est possible pour un utilisateur d'obtenir une évolution détaillée de la consommation mémoire ou du temps d'exécution requis pour chaque algorithme d'une application ou d'un schéma, en considérant plusieurs jeux de paramètres. Il est nécessaire d'exécuter p fois le schéma ou l'application. Ensuite, une p-calibration est réalisée pour chaque algorithme appelé dans le schéma ou l'application (voir sous-section 4.2.2).

Le choix des p points de référence de la p-calibration influence les résultats obtenus. En effet, en prenant des points de référence différents, les résultats p-calibrés diffèrent. Cependant, ceux-ci sont proches et ne dépassent pas les 20 % d'erreur maximal calculés dans les expérimentations. Une étude pourrait être réalisée afin de choisir les points permettant d'obtenir des courbes p-calibrées optimales.

Chapitre 5

Exploration de paramètres et conception de l'outil PAnTHErS

Sommaire

L'outil PAnTHErS pour *Prototyping and Analysis Tool for Homomorphic Encryption Schemes* [1] permet de modéliser des schémas de chiffrement homomorphe sous une forme qui rend possible d'évaluer l'impact de la variation de paramètres d'entrée sur les performances lors de l'exécution *c-à-d* sur les temps d'exécution et les consommations mémoire (voir Chapitres 3 et 4).

Cependant, il est nécessaire de bien connaitre le schéma modélisé pour pouvoir choisir des intervalles de valeurs convenables pour faire varier ses paramètres d'entrée, par exemple, des valeurs de paramètres qui garantissent une profondeur multiplicative suffisante pour la bonne exécution d'une application. Cette limitation peut être un frein pour l'usage de l'outil par un utilisateur novice. C'est pour répondre à ce besoin qu'un module d'exploration automatisée a été mis au point.

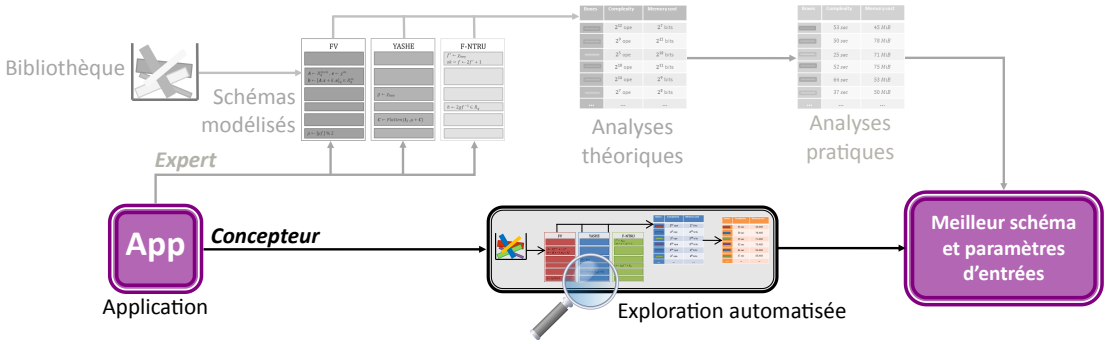


FIGURE 5.1 – Différentes étapes de PAnTHErS avec la méthode d'exploration automatisée mise en avant.

Ce chapitre présente une méthode d'exploration, mise en avant sur la Figure 5.1 représentant l'outil final, qui permettra à un utilisateur d'analyser des schémas de chiffrement homomorphe, sans nécessiter de connaissances préalables dans le domaine du chiffrement homomorphe. Suite à cela, le chapitre expose l'implémentation logicielle réalisée pour l'exécution de la modélisation, des analyses et de l'exploration des schémas de chiffrement homomorphe. En d'autres termes, le chapitre décrit les exigences et usages auxquels le programme final doit répondre. Ce programme est appelé PAnTHErS. De plus, après l'intégration d'applications dans PAnTHErS, ce chapitre est illustré de leurs résultats d'exploration.

5.1 Exploration complète des paramètres et des schémas

Les deux chapitres précédents (Chapitres 3 et 4) ont démontré la faisabilité d'une analyse de schéma de chiffrement homomorphe. Pour cela, une phase de modélisation est nécessaire avant d'analyser concrètement un schéma ou une application. Le chapitre 3 détaille en particulier comment un expert peut effectuer des analyses théoriques sur la complexité algorithmique et sur la consommation mémoire de ces schémas tandis que le chapitre 4 montre comment calibrer ses analyses théoriques afin d'obtenir des résultats concrets exprimés en secondes pour les temps d'exécution et en mébioctets pour la consommation mémoire.

Cette section présente une méthode d'exploration des paramètres d'entrée de schéma pour une application donnée. Cette exploration permet aux novices en cryptographie comme aux experts en chiffrement homomorphe de trouver le schéma et ses paramètres d'entrée

1. Outil de prototypage et d'analyse pour les schémas de chiffrement homomorphe, en français.

convenant le mieux pour une application donnée. En d'autres termes, l'exploration permet de sélectionner automatiquement le schéma et les paramètres d'entrée impliquant le plus petit temps d'exécution et/ou la plus faible consommation mémoire pour une application fixée en amont.

5.1.1 Limitations de la p-calibration

Le chapitre 4 décrit une méthode de calibration appliquée aux résultats théoriques du chapitre 3. Les résultats des analyses théoriques donnent une idée de l'évolution de la complexité algorithmique et de la consommation mémoire pour un schéma donné et la variation d'un paramètre d'entrée.

La p-calibration présentée dans le chapitre 4, permet de convertir la complexité algorithmique en temps d'exécution, et d'effectuer une évaluation de la consommation mémoire en mébioctets (Mio). Comme détaillé dans la section 4.2, la p-calibration se base sur l'exécution concrète de p jeux de paramètres appelés points de référence. Une p-calibration nécessitera donc p exécutions concrètes du schéma avec les points de référence choisis.

Cependant, il est important de bien sélectionner les intervalles de valeurs pour l'ensemble des paramètres d'entrée du schéma. En effet, certaines valeurs de paramètres n'auront pas de sens pour une application donnée. Par exemple, les paramètres doivent permettre d'engendrer une profondeur multiplicative utile pour l'application dans laquelle sera utilisée le schéma de chiffrement homomorphe.

Une bonne connaissance des schémas est donc indispensable pour savoir quelles valeurs choisir pour les paramètres pour les analyses théoriques et p-calibrées. Autrement dit, l'utilisation des modules d'analyse théorique et de calibration ne peut être faite que par des experts en chiffrement homomorphe.

C'est pourquoi une méthode d'exploration automatisée a été créée dans l'optique que tout utilisateur, même non expert en chiffrement homomorphe, puisse profiter des avantages des analyses de schémas. En s'appuyant sur les résultats de l'exploration, un concepteur saura alors quel schéma et quels paramètres d'entrée utiliser pour intégrer du chiffrement homomorphe dans ses applications. Cette méthode d'exploration est présentée en détail dans la sous-section suivante.

5.1.2 Présentation de la méthode d'exploration

La méthode d'exploration consiste à utiliser les méthodes de modélisation, d'analyse et de calibration, présentées dans les chapitres précédents, afin de déterminer, à partir d'une application et d'une configuration donnée, le schéma et ses paramètres d'entrée impliquant le plus faible coût en temps d'exécution et en utilisation mémoire, ou bien un compromis de ces métriques. L'exploration est décomposée en cinq étapes détaillées ci-dessous.

Étape 1 : *Sélection de l'application*
Cette étape vise à sélectionner une application et une configuration à analyser. L'application choisie possède une profondeur multiplicative d. Cette profondeur est une valeur clé, indispensable pour la suite de la méthode d'exploration. L'utilisateur peut choisir un intervalle de valeurs pour la profondeur multiplicative. Le minimum de cet intervalle doit alors correspondre à la profondeur d de l'application.

Par la suite, les étapes 2, 3 et 4 sont automatiquement déroulées pour chaque schéma modélisé et disponible dans la bibliothèque *c-à-d* pour FV [FV12], YASHE [BLLN13] et F-NTRU [DS16].

Étape 2 : *Identification des jeux de paramètres*

La profondeur multiplicative impliquée par schéma peut être déterminée à partir de ses paramètres d'entrée. Cependant, le calcul nécessaire pour déterminer cette profondeur multiplicative est différent pour chaque schéma. Il dépend (notamment) du bruit généré par le schéma. Les auteurs de schémas donnent dans leur article les bornes maximales du bruit produit après le chiffrement, l'addition homomorphe et la multiplication homomorphe. Ces bornes permettent de calculer, au final, la profondeur multiplicative du schéma à partir d'un jeu de paramètres fixé. Ce calcul a été intégré dans la méthode d'exploration afin de pouvoir calculer les profondeurs multiplicatives des schémas en fonction des jeux de paramètres d'entrée qui leurs sont fournis.

Cette étape 2 de l'exploration consiste à trouver tous les jeux possibles de paramètres d'entrée du schéma, noté S, impliquant la profondeur multiplicative de l'application choisie à l'étape 1. Pour cela, la profondeur multiplicative de chaque jeu de paramètres d'entrée est calculée. Les jeux de paramètres conservés sont ceux qui impliquent une profondeur multiplicative égale à celle de l'application choisie. Ces jeux sont stockés dans une liste notée L.

Dans le cas où un intervalle de profondeur est choisi à l'étape 1, tous les jeux de paramètres impliquant une profondeur multiplicative comprise dans cet intervalle sont conservés.

Étape 3 : *Analyses théoriques*

Tous les jeux de paramètres stockés dans la liste L, créée à l'étape 2, sont analysés théoriquement (section 3.2) avec l'algorithme d'analyse théorique du schéma S.

Étape 4 : *Calibration*

Les analyses théoriques sont ensuite calibrées avec la méthode de la p-calibration du chapitre 4. La p-calibration de l'exploration se déroule en deux parties : la première consiste en l'exécution de points de référence et la seconde en la conversion des analyses théoriques en données pratiques (temps d'exécution en secondes et coût mémoire en mébioctets).

Dans cette étape, des 2-calibrations sont effectuées. En effet, le chapitre 4 a montré que 2 est la plus petite valeur de p satisfaisant les exigences exprimées (voir sous-section 4.2.2). De plus, le choix d'une valeur la plus petite possible pour p limite le nombre d'exécutions nécessaires de points de référence et favorise donc la rapidité d'exécution de cette étape.

Étape 4.1 : *Exécution des points de référence*

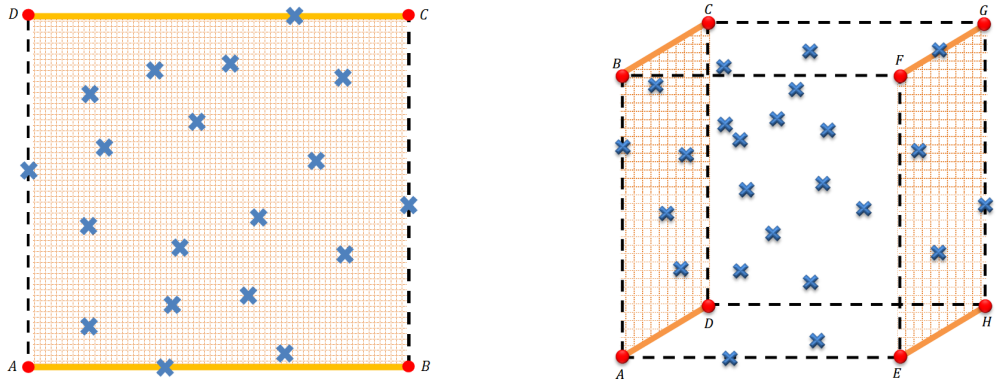
Cas général n paramètres variables : Soit $\mathcal{J}_2$ l'ensemble des jeux de paramètres $j_i = (p_1, p_2, ..., p_n)$ choisis à l'étape 2. Soit, $\forall k \in \{1, ..., n\}$, $min(p_k) = min\{p_k \in \mathcal{J}_2\}$ et $max(p_k) = max\{p_k \in \mathcal{J}_2\}$ les minimum et maximum de chaque paramètre utilisé dans les $j_i \in \mathcal{J}_2$. Les jeux de paramètres exécutés correspondent à tous les jeux étant une combinaison de $min(p_k)$ et/ou $max(p_k)$. Par conséquent, l'application sera exécutée avec 2^n jeux de paramètres j_{exec} différents. Les jeux j_{exec} ne sont pas forcément contenus dans $\mathcal{J}_2$. Dans la suite, l'ensemble $\mathcal{J}_{exec}$ désigne l'ensemble des jeux j_{exec}.

Application à F-NTRU ($n = 2$) : Les paramètres q (module des textes chiffrés) et w (base de décomposition de mots) sont variables dans F-NTRU. Tous les $j_i \in \mathcal{J}_2$ sont donc de la forme $(p_1, p_2) = (q, w)$. Les minimum et maximum de chaque paramètre sont trouvés : $min(q), max(q), min(w)$ et $max(w)$. En tout, l'application est exécutée avec $2^n = 2^2 = 4$ jeux de paramètres j_{exec}, listés dans l'équation 5.1. Les j_{exec} ne sont pas nécessairement tous contenus dans $\mathcal{J}_2$.

$$\begin{aligned} A &= j_{exec_1} = (min(q), min(w)) \\ B &= j_{exec_2} = (max(q), min(w)) \\ C &= j_{exec_3} = (max(q), max(w)) \\ D &= j_{exec_4} = (min(q), max(w)) \end{aligned} \tag{5.1}$$

Application à FV et YASHE ($n = 3$) **:** En plus de q et w, le paramètre t (module des textes clairs) est variable dans les schémas FV et YASHE. Tous les $j_i \in \mathcal{J}_2$ sont donc de la forme $(p_1, p_2, p_3) = (q, w, t)$. L'application est exécutée avec $2^n = 2^3 = 8$ jeux de paramètres j_{exec} différents. Ces paramètres sont listés dans l'équation 5.2.

$$\begin{array}{ll} A = j_{exec_1} = (min(q), min(w), min(t)) & E = j_{exec_5} = (min(q), min(w), max(t)) \\ B = j_{exec_2} = (min(q), max(w), min(t)) & F = j_{exec_6} = (min(q), max(w), max(t)) \\ C = j_{exec_3} = (max(q), max(w), min(t)) & G = j_{exec_7} = (max(q), max(w), max(t)) \\ D = j_{exec_4} = (max(q), min(w), min(t)) & H = j_{exec_8} = (max(q), min(w), max(t)) \end{array} \tag{5.2}$$



(a) Représentation pour la variation de 2 paramètres pour F-NTRU.

(b) Représentation pour la variation de 3 paramètres pour FV ou YASHE.

FIGURE 5.2 – Représentation schématique des jeux de paramètres dans $\mathcal{J}_2$, représentés par des croix bleues, et de leur calibration. Les points rouges représentent les jeux de paramètres dans $\mathcal{J}_{exec}$.

Étape 4.2 : *Conversion des analyses théoriques*

Dès que les exécutions des jeux de paramètres dans $\mathcal{J}_{exec}$ ont été réalisées, les analyses théoriques liées aux jeux de paramètres dans $\mathcal{J}_2$ sont alors calibrées. La démarche adoptée pour la calibration est représentée schématiquement dans la Figure 5.2. Pour rappel, dans cette phase d'exploration, seules des 2-calibrations sont effectuées.

La Figure 5.2a (resp. Figure 5.2b) représente le cas $n = 2$ (resp. $n = 3$). Dans les deux figures, les croix bleues font référence aux jeux de paramètres dans $\mathcal{J}_2$ et les points rouges aux jeux de paramètres dans $\mathcal{J}_{exec}$. Les coordonnées de chaque point rouge sont notées sur les Figures 5.2a et 5.2b. Les lignes pleines et pointillées symbolisent les limitations de l'ensemble $\mathcal{J}_2$. L'objectif, ici, est de calibrer les croix bleues, compris entre les délimitations, à l'aide de leur analyse théorique et de l'exécution concrète de l'application avec le schéma pour chaque point rouge.

Cas $n = 2$ **:** Sur la Figure 5.2a *c-à-d* dans le cas où $(p_1, p_2) = (q, w)$, l'application est exécutée avec le schéma pour chacun des 4 jeux de paramètres représentés par les points rouges. Deux premières calibrations sont alors réalisées entre :

- le point $A = (min(q), min(w))$ et le point $B = (max(q), min(w))$,
- le point $D = (min(q), max(w))$ et le point $C = (max(q), max(w))$.

Ces premières calibrations sont présentes dans la Figure 5.2a sous la forme de deux traits pleins orange (2 côtés opposés du carré : AB et CD). Lors de ces deux calibrations, w est fixé et q varie. Afin de pouvoir réaliser ces calibrations, les analyses théoriques des jeux de paramètres compris entre A et B et entre D et C doivent être calculées en amont.

Enfin, à partir des exécutions et des analyses calibrées citées ci-dessus, toutes les analyses théoriques des croix bleues *c-à-d* les jeux de paramètres dans $\mathcal{J}_2$ peuvent

être calibrées. Chaque analyse théorique d'un jeu de paramètre (q_i, w_i) est calibrée en prenant comme points de référence $(q_i, min(w))$ et $(q_i, max(w))$. Ces deux derniers points sont associés soit au résultat concret dû à une exécution soit à des analyses calibrées. Schématiquement, sur la Figure 5.2a, quand les analyses liées aux points rouges et ceux sur les lignes pleines orange sont calibrées, alors les analyses théoriques de tous les points de la surface $ABCD$ (fond quadrillé) peuvent être calibrées.

Cas $n = 3$: Sur la Figure 5.2b *c-à-d* dans le cas où $(p_1, p_2, p_3) = (q, w, t)$, l'application est exécutée avec le schéma en prenant chacun des huit points rouges comme points de référence. En fixant le paramètre $p_3 = min(t)$ (resp. $p_3 = max(t)$) et en variant seulement q et w, les analyses théoriques des points de la face $ABCD$ du cube (resp. la face opposée $EFGH$) représentée dans la Figure 5.2b peuvent être calibrées comme expliqué pour le cas $n = 2$. Pour cela, les analyses théoriques des points de ces deux faces doivent être calculées en amont. Á cet instant, toutes les analyses des points de la forme (q_i, w_i, t_i) avec $t_i = min(t)$ ou $t_i = max(t)$ sont calibrées. Enfin, chaque analyse théorique de jeux de paramètres de la forme (q_i, w_i, t_i) avec $t_i \neq min(t)$ et $t_i \neq max(t)$ peut être calibrée en prenant comme points de référence $(q_i, w_i, min(t))$ et $(q_i, w_i, max(t))$.

Étape 5 : *Tri et sélection du meilleur schéma et de ses paramètres d'entrée*

Les étapes précédentes ont permis de récupérer les analyses calibrées de l'application pour tous les schémas présents dans la bibliothèque pour chaque jeu de paramètres impliquant une des profondeurs multiplicatives choisies à l'étape 1 (soit une seule profondeur, équivalente à celle de l'application, soit une profondeur parmi l'intervalle de valeurs choisi). L'objectif de cette étape est de déterminer le schéma et les jeux de paramètres engendrant les meilleurs résultats en matière de temps d'exécution et/ou en consommation mémoire pour l'application.

L'utilisateur peut choisir en amont s'il préfère privilégier les schémas et jeux de paramètres impliquant le temps d'exécution (resp. le coût mémoire) le plus faible. Dans ce cas, les paramètres sont triés en fonction des complexités (resp. consommations mémoire) calibrées associées et le jeu de paramètres dont l'analyse calibrée à fait ressortir la durée (resp. consommation mémoire) la plus basse est retournés à l'utilisateur.

Si l'utilisateur n'a pas forcément de préférence entre la complexité et le coût mémoire, alors l'optimum de Pareto [Par64] est utilisé, tel que décrit dans la définition 17.

Définition 17 (Front de Pareto du contexte de la méthode d'exploration) *Soit $\mathcal{J}_2$ l'ensemble des jeux de paramètres et $\mathcal{J}_P$ l'ensemble des points sur le front de Pareto. Chaque $j \in \mathcal{J}_2$ (resp. $j_P \in \mathcal{J}_P$) engendre un temps d'exécution t_j (resp. t_P) et un coût mémoire m_j (resp. m_P). Alors $\forall j \in \mathcal{J}_2$ tel que $j \notin \mathcal{J}_P$, il existe $j_P \in \mathcal{J}_P$ tel que $t_P \leq t_j$ et $m_P \leq m_j$.*

La Figure 5.3 donne un exemple du front de Pareto. Chaque jeu de paramètres, représenté par un carré ou un triangle, a été positionné en fonction du temps d'exécution et de la consommation mémoire qu'il engendre. D'après la définition 17, les jeux de paramètres étant sur le front de Pareto sont ceux illustrés par un triangle vert sur la Figure 5.3. Graphiquement, il est possible de vérifier qu'un jeu de paramètres est sur le front de Pareto ou non. Par exemple, le jeu de paramètres A l'est car aucun jeu de paramètres n'est présent sur la surface entre $(0, 0)$, $(t_A, 0)$, (t_A, m_A) et $(0, m_A)$. En revanche, sur la Figure 5.3, un carré bleu est présent sur la surface $(0, 0)$, $(t_B, 0)$, (t_B, m_B) et $(0, m_B)$; autrement dit, le jeu de paramètres B n'est pas sur le front de Pareto.

Le tri de cette étape 5 consiste trouver les jeux de paramètres présents sur le front de Pareto. Ces jeux de paramètres sont ensuite proposés à l'utilisateur qui pourra choisir un des jeux selon qu'il souhaite favoriser un meilleur temps d'exécution ou un coût mémoire plus faible.

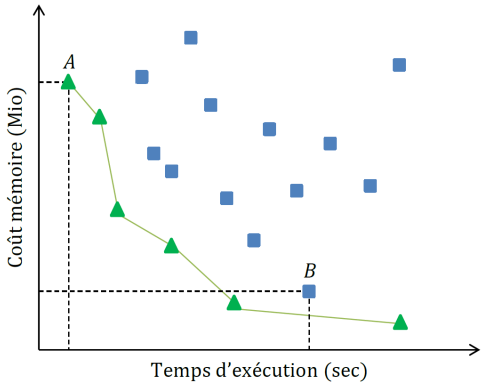


FIGURE 5.3 – Représentation graphique d'un exemple d'application de l'optimum de Pareto à un ensemble de résultats. Chaque triangle et carré représente un jeu de paramètres en fonction des temps d'exécution et de la consommation mémoire qu'il engendre. Les triangles verts sont les jeux de paramètres sur le front de Pareto.

Exigences

Dans l'exploration, nous appliquons une calibration en prenant des points de référence issus d'une première calibration. Autrement dit, d'après les exigences posées dans le chapitre 4, les points de références issus de la première calibration possèdent un taux d'erreur moyen de 10 %. Et 5 % de ces points de référence affichent un taux d'erreur supérieur à 20 %. En effectuant la seconde calibration à partir de ces valeurs, le biais déjà présent va se trouver accentué dans les nouveaux résultats calibrés.

Concrètement, en propageant successivement les taux d'erreur précédents (*c-à-d* resp. 10 % et 20 %) sur des valeurs v_1 et v_2, ces dernières peuvent présenter un écart absolu final de :

- 10 % de $(v_1 + 10$ % de $v_1) = \dfrac{10}{100}\left(\dfrac{110}{100} \times v_1\right) = \dfrac{121}{100} \times v_1 = 21$ % de v_1.
- 20 % de $(v_2 + 20$ % de $v_2) = \dfrac{20}{100}\left(\dfrac{120}{100} \times v_2\right) = \dfrac{144}{100} \times v_2 = 44$ % de v_2.

Nous avons arrondi ces deux pourcentages aux 5 % supérieurs afin de poser les exigences pour les taux d'erreur des résultats calibrés de l'exploration. Les exigences posées sont les suivantes :

- La moyenne des taux d'erreur ne doit pas dépasser 25 %.
- Au moins 95 % des résultats calibrés doivent avoir un taux d'erreur inférieur à 45 %.

Conclusion

Ces cinq étapes de la méthode d'exploration ne demandent pas l'intervention de l'utilisateur. Ce dernier renseigne seulement l'application qu'il souhaite analyser (étape 1). L'exploration analyse alors toutes les combinaisons de schémas et de jeux de paramètres adaptés pour l'application choisie afin de trouver celles impliquant temps d'exécutions et coûts mémoire les plus intéressants. L'optimum de Pareto est utilisé pour déterminer les meilleurs jeux de paramètres parmi tous ceux analysés. Ces jeux de paramètres et leurs estimations sont ensuite présentés à l'utilisateur qui pourra sélectionner le jeu dont l'estimation lui semble la plus satisfaisante.

Perspectives

Comme présenté dans les métriques secondaires (sous-section 3.2.5) du chapitre 3, le calcul de l'expansion de bruit pourrait être rajouté dans les analyses théoriques. Ainsi, le calcul de la profondeur multiplicative d'un schéma serait alors calculable dès lors que ce schéma est modélisé. Par conséquent, l'étape 2 pourrait utiliser ce calcul afin d'évaluer la profondeur engendrée par les jeux de paramètres choisis.

5.1.3 Application de l'exploration sur les schémas

Cette sous-section présente les résultats de la méthode d'exploration, décrite ci-dessus, pour l'algorithme *cinqHB* (Algorithme 17), décrit dans le chapitre 3. Cet algorithme représente la succession des cinq algorithmes homomorphes basiques d'un schéma. L'exploration a été appliquée en utilisant les schémas FV [FV12], YASHE [BLLN13] et F-NTRU [DS16]. La configuration logicielle et matérielle de l'exploration est récapitulée dans la Table 5.1. Pour chaque exploration, nous avons alloué 2 cœurs de calcul ainsi que 128 Go de mémoire vive. Enfin, le logiciel Sage 8.0 a été lancé dans un conteneur Docker.

TABLE 5.1 – Configuration matérielle et logicielle de l'environnement d'exécution de l'exploration.

Processeurs	8 × Intel Xeon E7-8890 v4 @ 2.20 GHz
Nombre de cœurs	8 × 24
Mémoire vive	192 × 32 Go / DDR4
Système d'exploitation	Red Hat Entreprise Linux 7.5 64-bit
Environnement logiciel	Sage 8.0 Python 2.7.10

TABLE 5.2 – Bornes minimales et maximales atteintes sur l'ensemble des jeux de paramètres d'entrée de chaque schéma, trouvées après exploration de l'application *cinqHB*.

Schéma	$\log(q)$		$\log(w)$		t	
	Min	**Max**	**Min**	**Max**	**Min**	**Max**
FV	43	148	2	99	2	49
YASHE	45	162	2	99	2	49
F-NTRU	18	124	2	99	-	-

L'étape 2 de l'exploration consiste à trouver tous les jeux de paramètres qui impliquent la profondeur de l'application choisie en étape 1. Bien qu'une exploration totale des paramètres soit possible, nous avons fixé des bornes maximales pour les paramètres d'entrée des schémas. Pour rappel, deux paramètres d'entrée sont variés pour FV, YASHE et F-NTRU : il s'agit de q et w. De plus, le paramètre t est varié également pour FV et YASHE. Nous avons fixé une borne de 2^{500} (pour FV et YASHE) et 2^{150} (pour F-NTRU) pour q, 2^{100} pour w et 50 pour t. Dans la Table 5.2, les bornes atteintes pour chaque schéma durant l'exploration sont récapitulées.

Toutefois, les bornes définies pour les paramètres d'entrée n'empêchent pas une exploration conséquente. En effet, la Table 5.3 indique dans la seconde ligne le nombre de jeux de paramètres explorés pour chaque schéma. Cumulés, ces nombres de jeux de paramètres donnent un total de 186082 analyses pour l'application *cinqHB*. La Table 5.3 renseigne également le temps nécessaire pour trouver les jeux de paramètres. Cependant, une fois

TABLE 5.3 – Informations sur l'exploration réalisée pour chaque schéma pour l'application *cinqHB* : le nombre de jeux de paramètres d'entrée explorés, le temps d'exécution de l'exploration pour chaque schéma (temps pour trouver les jeux de paramètres - étape 2 - et temps de l'analyse de ces jeux de paramètres - étape 3,4,5), le temps total des exécutions concrètes et le facteur d'accélération (ratio entre temps d'analyse et temps d'exécution concret).

Schéma	Nombre de jeux	Temps pour trouver les jeux	Durée d'exploration des paramètres	Temps total des exécutions	Accélération
FV	96456	1 h 36 min 25 sec	9 h 33 min 57 sec	16 h 56 min 25 sec	1.8
YASHE	87400	3 h 37 min 50 sec	9 h 38 min 55 sec	127 h 56 min 35 sec	13.3
F-NTRU	2226	0 h 5 min 47 sec	0 h 32 min 54 sec	2 h 14 min 26 sec	4.1
TOTAL	186082	5 h 20 min 2 sec	19 h 45 min 46 sec	147 h 7 min 26 sec	7.4

que les jeux de paramètres ont été calculés pour une profondeur multiplicative fixée, ceux-ci sont stockés et ré-utilisables. Autrement dit, pour un schéma modélisé dans la bibliothèque, la recherche de jeux de paramètres d'entrée impliquant une certaine profondeur multiplicative d est réalisée une et une seule fois.

Les temps d'exécution des explorations de schémas sont indiqués dans la Table 5.3 (colonne 4). Chacun de ces temps va dépendre du nombre de jeux de paramètres analysés, du nombre d'algorithmes à évaluer dans le schéma, du temps requis pour l'exécution des jeux de paramètres et des jeux supplémentaires analysés requis (points de référence) pour effectuer la calibration (voir étape 4 de la sous-section 5.1.2).

Aussi, pour évaluer la pertinence des résultats obtenus, tous les jeux de paramètres ont été exécutés concrètement. Le temps total d'exécution requis pour ces exécutions concrètes est noté dans la Table 5.3 (colonne 5).

Le ratio entre le temps total des exécutions concrètes et du temps d'exécution de l'exploration donne le taux d'accélération de l'exploration (Table 5.3, colonne 6). Ce taux d'accélération est différent pour chaque schéma. Il dépend du nombre de jeux de paramètre à évaluer, du schéma et de l'application choisie. L'exploration de FV est seulement 1.8 fois plus rapide qu'une exécution concrète alors que l'exploration de YASHE est 13.3 fois plus rapide. Pour une exploration des trois schémas, le facteur d'accélération est de 7.4.

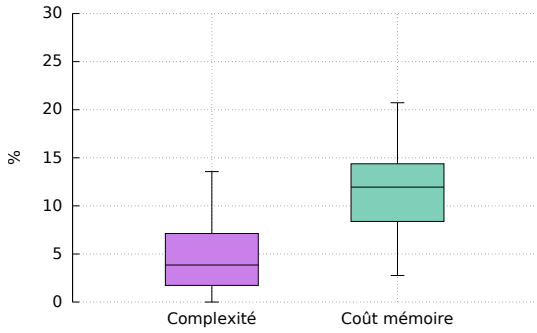
En l'occurrence, le faible taux d'accélération de FV est dû à la rapidité d'exécution concrète des jeux de paramètres. Le temps nécessaire à l'exécution concrète de l'ensemble des jeux de paramètre est trop proche du temps nécessaire pour effectuer les analyses de schémas, ce qui implique un faible taux d'accélération.

Étude des taux d'erreur

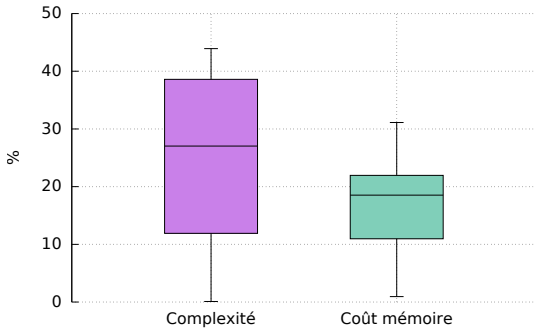
La Figure 5.4 illustre la répartition des taux d'erreur entre les résultats issus des analyses théoriques calibrées et ceux issus des analyses concrètes (obtenus suite à une exécution concrète). La Figure 5.4 présente les résultats pour les trois schémas et les deux analyses : complexité et coût mémoire.

Pour rappel, dans la Figure 5.4, comme pour le chapitre 4, le rectangle comprend les taux d'erreur compris entre le 1er quartile et le 3ème quartile. La médiane de ces taux est dessinée par un segment coupant le rectangle en deux parties. Les segments, à l'extérieur du rectangle correspondent au 1er au 99ème centile (pour FV et F-NTRU) et 1er au 96ème centile (pour YASHE).

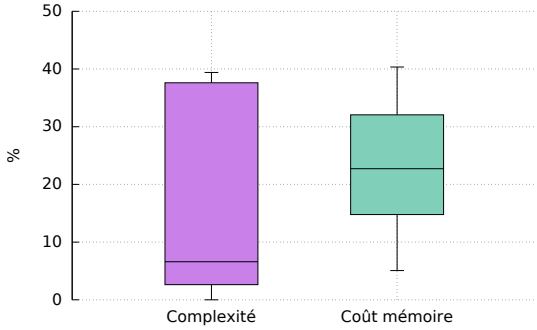
Par rapport aux exigences posées, au moins 95 % des taux d'erreur sont inférieurs à 45 % pour les trois schémas. De plus, 50 % des taux d'erreur pour la complexité de FV et YASHE sont inférieurs à 7 %. Les taux d'erreur moyens sont récapitulés dans la Table 5.4.



(a) FV (99 centiles).



(b) F-NTRU (99 centiles).



(c) YASHE (96 centiles).

FIGURE 5.4 – Boîte de Tukey des taux d'erreur obtenus pour les résultats de l'exploration comparés aux résultats des exécutions concrètes, pour chaque schéma et chaque analyse théorique calibrée (complexité et coût mémoire).

D'après ces taux, les exigences sont donc respectées à 0.4 % près en moyenne pour YASHE (complexité).

D'après la Table 5.4, le taux d'erreur maximal atteint pour le schéma YASHE est de 573.6 % pour la complexité. Plus généralement, tous les jeux de paramètres ayant un $\log(q)$ compris entre 49 et 62 inclus, soit 2 % des jeux de paramètres explorés pour YASHE, présentent un taux d'erreur supérieur à 100 %. La moyenne des taux d'erreur pour la complexité de ces jeux de paramètres est de 396.4 %. Pour les 98 % des jeux de paramètres restant, la moyenne de leur taux d'erreur pour la complexité est de 17.7 %. Ces forts écarts peuvent être expliqués par un phénomène de propagation d'erreur présenté ci-après.

Pour rappel, pour le schéma YASHE, l'étape 4 de la calibration dans l'exploration commence par une 2-calibration entre $\log(q) = min(\log(q))$ et $\log(q) = max(\log(q))$ avec $\log(w) = min(\log(w))$ et $t = min(t)$ *c-à-d* entre les points A et D de la Figure 5.2b dans la sous-section 5.1.2. Les taux d'erreur générés par cette première calibration sont illustrés dans la Figure 5.5.

Dans la Figure 5.5, pour $\log(q)$ allant de 49 à 62, le taux d'erreur des valeurs 2-calibrées est supérieur à 100 %. Ces valeurs 2-calibrées ont été prises en référence lors de l'exploration. Autrement dit, dès la première calibration, un important taux d'erreur est généré. Les calibrations suivantes prennent comme références des points présentant déjà un taux d'erreur supérieur à 100 %. Ainsi, ce taux d'erreur est propagé sur les calibrations suivantes pour tous les points ayant un $\log(q)$ compris entre 49 et 62.

Une solution pour contrer ce problème est d'effectuer une calibration plus fine pour réduire les taux d'erreur. Pour cela, un ou plusieurs points de référence supplémentaires peuvent être ajoutés afin de réaliser une p-calibration ($p > 2$) entre A et D. L'idéal est

TABLE 5.4 – Taux d'erreur moyens et maximaux atteint par les paramètres explorés pour FV, YASHE et F-NTRU.

Schémas	**Complexité**		**Coût mémoire**	
	Moyen	Maximal	Moyen	Maximal
FV	4.8 %	50.1 %	11.4 %	34.8 %
YASHE	25.4 %	573.6 %	22.3 %	43.1 %
F-NTRU	23.7 %	43.9 %	17.0 %	31.1 %

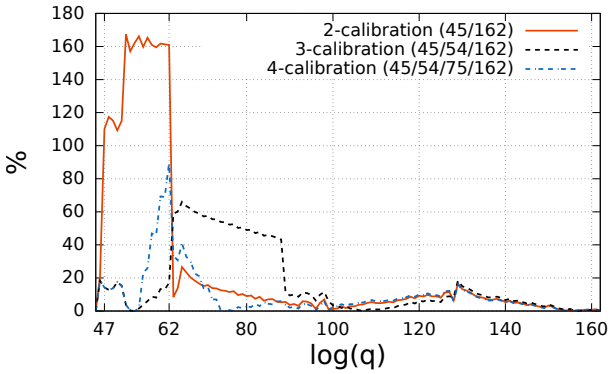


FIGURE 5.5 – Taux d'erreur des valeurs p-calibrées avec $p = 2$, 3 et 4 du schéma YASHE pour $\log(q)$ allant de 45 à 162. Les $\log(q)$ pris en référence sont notés entre parenthèses dans la légende de la figure.

de prendre, parmi les nouveaux points de référence, un point ayant pour valeur un $\log(q)$ compris entre 49 et 62.

La courbe 3-calibrée de la Figure 5.5 prend effectivement comme référence $\log(q) = 54 \in [49, 62]$ en plus de $\log(q) = 45$ et 162. Graphiquement, une baisse des taux d'erreur est visible entre les courbes 2-calibrée et 3-calibrée ainsi qu'entre les courbes 3-calibrée et 4-calibrée. Plus concrètement, la moyenne des taux d'erreur pour les valeurs p-calibrées sur la Figure 5.5 est de 26.0 % pour $p = 2$, 16.6 % pour $p = 3$ et 9.8 % pour $p = 4$. Ces chiffres montrent bien une réduction des taux d'erreur plus p est grand *c-à-d* plus la calibration est fine.

Les exigences posées sur les taux d'erreur sont respectées dans l'ensemble. Seul le schéma YASHE présente un taux d'erreur moyen global de 25.4 % pour la complexité. Ce taux d'erreur moyen est légèrement supérieur aux 25 % exigés : en effet, 2 % des jeux de paramètres présentent des forts taux d'erreur (396.4 % en moyenne). L'étude de l'exploration de ce schéma a montré qu'une solution pour contrer ce problème est réalisable : cette solution consiste en l'exécution d'une p-calibration ($p > 2$) au lieu d'une 2-calibration sur les premières calibrations réalisées (entre A et D, entre B et C, entre E et H et entre F et G). Par exemple, la moyenne des taux d'erreur générés après la première 2-calibration (entre A et D) pour YASHE est de 26.0 %. Cette moyenne peut baisser jusqu'à 9.8 % en effectuant une 4-calibration.

Étude des jeux de paramètres optimaux

D'un côté, les résultats de l'exploration pour chacun des schémas sont détaillés dans la Table 5.5. Cette table présente tous les jeux de paramètres trouvés grâce au front de Pareto pour chaque schéma (indépendamment des autres). Pour chaque paire (schéma,

TABLE 5.5 – Résultats retournés par la méthode d'exploration de l'application *cinqHB* pour les schémas FV, YASHE et F-NTRU.

Schéma	$\log(q)$	$\log(w)$	t	**Temps (sec)**			**Coût mémoire (Mio)**		
				Estim.	**Exéc.**	**Erreur**	**Estim.**	**Exéc.**	**Erreur**
FV	46	23	3	0.3846	0.3691	4.1898 %	7.9356	8	0.8050 %
	46	23	2	0.3633	0.3922	7.3580 %	7.9846	8.2	2.6269 %
	43	15	2	0.3672	0.3962	7.3204 %	7.9698	8	0.3776 %
YASHE	46	10	2	0.1398	0.1268	10.2999 %	2.7922	2.5	11.6876 %
	75	38	3	1.7660	1.7349	1.7920 %	2.7673	4.5	38.5052 %
F-NTRU	30	16	–	0.6420	0.7539	14.8516 %	7.7350	8.5	8.9996 %
	32	17	–	0.6407	0.7194	10.9436 %	7.7396	8.5	8.9453 %
	34	18	–	0.6396	0.7143	10.4638 %	7.7485	8.5	8.8409 %
	36	19	–	0.6386	0.7910	19.2638 %	7.7614	8.5	8.6898 %
	38	20	–	0.6378	0.7440	14.2697 %	7.7780	8.5	8.4942 %
	40	21	–	0.6372	0.7071	9.8893 %	7.7982	8.5	8.2562 %
	42	22	–	0.6366	0.6991	8.9480 %	7.8220	8.5	7.9769 %
	44	23	–	0.6360	0.7020	9.3977 %	7.8491	8.5	7.6574 %
	46	24	–	0.6356	0.6752	5.8689 %	7.8797	8.5	7.2981 %

jeu de paramètres d'entrée), l'analyse théorique calibrée est confrontée aux résultats issus d'exécutions concrètes. Ainsi, un taux d'erreur a pu être calculé.

D'un autre côté, tous les jeux de paramètres obtenus à l'étape 2 de l'exploration ont été exécutés afin de récupérer les temps d'exécutions et les coût mémoires concrets. Suite à ces exécutions, nous avons récupéré et listé sur la Table 5.6 les résultats du front de Pareto appliqué sur les résultats concrets. Sur la Table 5.6, les lignes écrites en rouge correspondent aux jeux de paramètres présents également dans la Table 5.5. En d'autres termes, les jeux de paramètres optimaux en rouge ont été obtenus par le module d'exploration des paramètres.

Les deux Tables 5.6 et 5.5 listent des jeux de paramètres différents. Par conséquent, afin de mieux se rendre compte de l'efficacité de l'exploration, chaque jeu de paramètre exploré est représenté sur un graphe dans la Figure 5.6.

La Figure 5.6 représente la répartition des analyses de jeux de paramètres explorés (croix bleue). Chaque jeu de paramètres est placé sur le graphe en fonction de son temps d'exécution concret (axe des abscisses) et de son coût mémoire réel (axe des ordonnées). Sur la figure, les carrés rouges correspondent aux paramètres ou points optimaux *c-à-d* sur le front de Pareto des valeurs concrètes. Les ronds verts représentent les points sélectionnés par la méthode d'exploration, *c-à-d* les points listés dans la Table 5.5, placés sur le graphe avec leur temps d'exécution et coût mémoire concrets (colonne 6 et 9 de la Table 5.5).

Un jeu de paramètres ou point optimal correspond à un jeu de paramètres induisant une consommation mémoire et/ou un temps d'exécution faible, ou encore un compromis entre les deux. En d'autres termes, les points optimaux se situeront dans le coin bas-gauche de leur graphe sur la Figure 5.6. Les jeux de paramètres obtenus par l'exploration sont bien situés dans le coin bas-gauche sur les graphes de la Figure 5.6 et s'approchent donc des points optimaux. Pour le schéma YASHE, un unique point a été sélectionné par la méthode de l'exploration. Ce point correspond à l'unique point optimal, identifié sur le front de Pareto des résultats d'exécutions concrètes. Pour les deux autres schémas (FV et F-NTRU), nous avons évalué la distance entre les points sélectionnés par l'exploration et les points optimaux des exécutions afin d'évaluer leur proximité.

TABLE 5.6 – Résultats retournées après l'exécution complète de tous les jeux de paramètres de l'application *cinqHB* pour les schémas FV, YASHE et F-NTRU. En rouge, les jeux de paramètres présents également dans la Table 5.5.

Schéma	$\log(q)$	$\log(w)$	t	**Temps (sec)**	**Coût mémoire (Mio)**
FV	43	18	2	0.3516	8.3
	45	17	3	0.3540	8
	45	15	4	0.3549	7.9
	46	24	2	0.3510	8.5
	46	21	3	0.3540	8
YASHE	46	10	2	0.12678	2.5
F-NTRU	18	10	–	0.7116	7.9
	20	11	–	0.7033	7.9
	20	12	–	0.6994	7.9
	41	21	–	0.6872	8.5
	43	22	–	0.6784	8.5
	46	24	–	0.6752	8.5

Le critère de détermination des points optimaux $\mathcal{P}$ est leur présence sur le front de Pareto. Les points choisis en seconde position sont les points se trouvant sur le deuxième front de Pareto *c-à-d* les points sur le front de Pareto de l'ensemble des jeux de paramètres exécutés $\mathcal{J}_2^{exec}$ moins les points du premier front de Pareto $\mathcal{P}$. Ainsi de suite, nous pouvons définir le $n^{\text{ème}}$ front de Pareto détaillé ci-dessous.

Définition 18 ($n^{\text{ème}}$ front de Pareto) *Soit $\mathcal{P}_1$ l'ensemble des points se trouvant sur le front de Pareto de $\mathcal{J}_2^{exec}$. Notons $\mathcal{P}_1$ = $FrontPareto(\mathcal{J}_2^{exec})$ avec $FrontPareto(E)$ retournant l'ensemble des points sur le front de Pareto de l'ensemble E. L'ensemble des points se trouvant sur le $n^{ème}$ front de Pareto est :*

$$\mathcal{P}_n = FrontPareto\left(\mathcal{J}_2^{exec} \setminus \bigcup_{i=1}^{n-1} \mathcal{P}_i\right). \tag{5.3}$$

À partir de l'indice de front de Pareto des points sélectionnés par l'exploration, nous avons calculé une position de classement comme défini ci-dessous.

Définition 19 (Classement d'un jeu de paramètres) *Soit j un jeu de paramètres présent dans $\mathcal{P}_n$ l'ensemble des points du $n^{ème}$ front de Pareto. La position de classement de j est :*

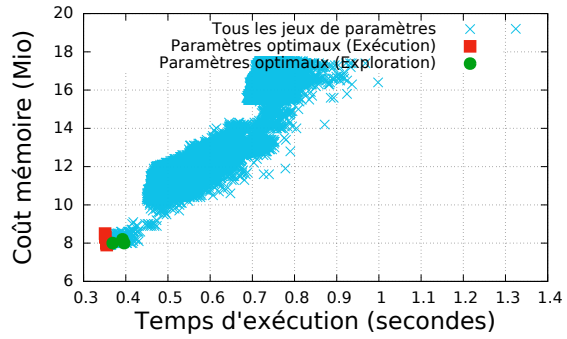
$$Card\left(\bigcup_{i=1}^{n-1} \mathcal{P}_i\right) + 1, \tag{5.4}$$

avec $Card(E)$ le cardinal de l'ensemble E.

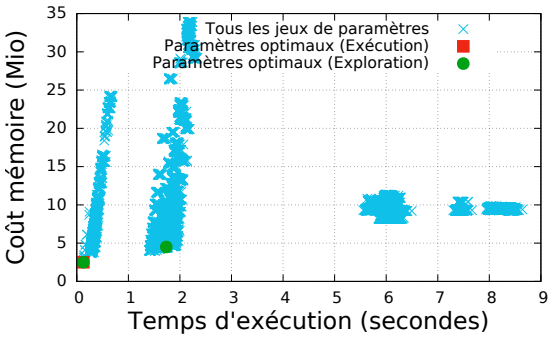
Cette position de classement c permet de dire qu'un jeu de paramètres j est le $c^{\text{ème}}$ point choisi sur un total de $Card(\mathcal{J}_2^{exec})$ points. Autrement dit, le jeu j est situé à p % dans le classement total des points avec :

$$p = \frac{c-1}{Card(\mathcal{J}_2^{exec})} \times 100 \tag{5.5}$$

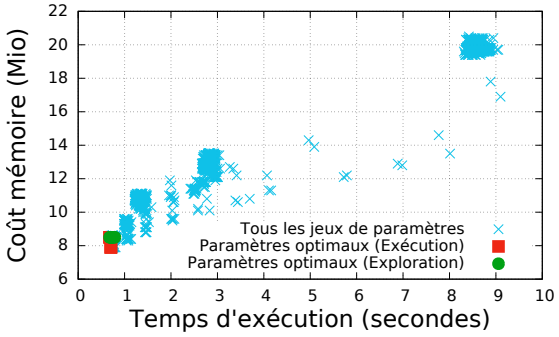
La Table 5.7 liste le classement en terme de position et de pourcentage dans le classement des points obtenus par exécution concrète des trois schémas. Les points sélectionnés



(a) FV.



(b) YASHE.



(c) F-NTRU.

FIGURE 5.6 – Identification des points retournés par l'exploration. Les croix bleues représentent les résultats d'analyse obtenus suite à l'exécution concrète des jeux de paramètres. Les carrés rouges correspondent aux points optimaux (*c-à-d* identifiés sur le front de Pareto) des exécutions (croix bleues). Les ronds verts représentent les résultats sélectionnés lors de la phase de l'exploration.

pour FV et YASHE se situent dans le premier centile du classement final. Plus de la moitié des points sélectionnés pour F-NTRU (5 points sur 9 au total) se situent à moins de 5 % du classement. Le point sélectionné pour F-NTRU le plus éloigné dans le classement ($(\log(q), \log(w)) = (36, 19)$) est situé à 14.0611 % *c-à-d* qu'il se situe environ à un septième des 2226 points explorés de F-NTRU.

En plus du classement des points, la Table 5.7 donne la différence maximale visible entre chaque temps d'exécution (resp. coût mémoire) des points sélectionnés par l'exploration et le meilleur temps d'exécution concrète (resp. coût mémoire). Le point ayant la plus grande différence de temps, *c-à-d* le point $(\log(q), \log(w)) = (36, 19)$ de F-NTRU, est distant de moins de 0.12 seconde du temps optimal et de 0.6 Mio du coût mémoire optimal. Ces chiffres correspondent à un écart de 17.1 % du temps optimal et de 7.6 % du coût mémoire optimal.

Conclusion

L'exploration a permis de mettre en avant 14 jeux de paramètres pour trois schémas (FV, YASHE et F-NTRU). Au total, sans compter le temps nécessaire pour trouver les 186082 jeux de paramètres à explorer, l'exploration a été effectuée 7.4 fois plus vite que le temps requis pour exécuter intégralement l'application pour chaque jeu de paramètres à évaluer.

TABLE 5.7 – Classement des points sélectionnés par l'exploration et écart de ces points avec le résultat optimal obtenu par exécution concrète.

Schéma	$\log(q)$	$\log(w)$	t	**Classement**		**Écart avec le point optimal**	
				Position	**En %**	**Temps (sec)**	**Coût mémoire (Mio)**
FV	46	23	3	43/96456	0.0435	0.0181	0.1
	46	23	2	140/96456	0.1441	0.0412	0.3
	43	15	2	18/96456	0.0176	0.0452	0.1
YASHE	46	10	2	1/87400	0.0000	0.0	0.0
	75	38	3	867/87400	0.9908	1.6081	2.0
F-NTRU	30	16	–	209/2226	9.3441	0.0787	0.6
	32	17	–	131/2226	5.8401	0.0442	0.6
	34	18	–	82/2226	3.6388	0.0391	0.6
	36	19	–	314/2226	14.0611	0.1158	0.6
	38	20	–	292/2226	13.0728	0.0688	0.6
	40	21	–	18/2226	0.7637	0.0319	0.6
	42	22	–	7/2226	0.2695	0.0239	0.6
	44	23	–	28/2226	1.2129	0.0268	0.6
	46	24	–	1/2226	0	0.0	0.6

En exécutant concrètement l'application pour l'ensemble de ses jeux de paramètres, un classement des résultats a pu être établi en s'aidant du front de Pareto. Parmi les résultats de l'exploration, représentés par des points, 10 sur 14 se situent dans les 5 premiers centiles du classement final. Le résultat le moins bien classé (pour F-NTRU) se situe dans le 15$^{\text{ème}}$ centile du classement des résultats pour ce schéma.

Les exigences fixées dans la sous-section 5.1.2 sont garanties pour les schémas FV et F-NTRU. En revanche, la moyenne des taux d'erreur engendrée par les points du schéma YASHE (25.4 %) dépasse les 25 % fixés. En étudiant de plus près l'exploration du schéma, nous avons remarqué que les premières calibrations engendrent des taux d'erreur supérieurs à 100 %. Ces taux d'erreur initiaux peuvent être améliorés en appliquant la perspective décrite ci-dessous.

Perspectives

Une perspective d'amélioration de l'exploration serait d'affiner l'étape de la calibration en ajoutant des points de références (à exécuter). Actuellement, le nombre de points de référence pour les 2-calibrations est limité à 4 pour un schéma possédant deux paramètres variables et à 8 pour un schéma ayant trois paramètres variables. Ce nombre de points de référence pourrait évoluer soit :

- en fonction du nombre total de jeux de paramètres à explorer.
- en fonction de la grandeur des intervalles des paramètres variés.

La Table 4.7 du chapitre 4 met en avant des taux d'erreur moyens pour des p-calibrations avec $p = 2$, 3 et 4. Cette table permet de constater que les taux d'erreur pour $p = 3$ sont plus faibles que pour $p = 2$. Par exemple, pour le schéma FV, le taux d'erreur moyen des résultats coûts mémoire de q est de 9.7 % pour $p = 2$ et de 3.6 % pour $p = 3$.

En disposant de plus de points de référence, les p-calibrations effectuées dans le module d'exploration ($p \geq 2$) seraient plus fines et par conséquent induiraient un taux d'erreur plus faible. En effet, la propagation d'erreur d'estimation, présentée dans la sous-section 5.1.2, serait moins importante lors des calibrations successives. La propagation d'erreur sur deux 2-calibrations est estimé à 21 % (d'après le calcul effectué lors des exigences, dans la section 5.1.2) en partant du fait qu'une 2-calibration génère un taux d'erreur de 10 %. En prenant

comme référence le taux d'erreur moyen de l'exemple précédent pour une 3-calibration, la propagation d'erreur sur deux 3-calibrations est de 7.3 %. Ce taux d'erreur est trouvé en appliquant deux fois une 3-calibration, générant chacune 3.6 % d'erreur, sur une valeur v :

- 3.6 % de $(v + 3.6$ % de $v) = \frac{3.6}{100}\left(\frac{103.6}{100} \times v\right) = \frac{107.3}{100} \times v = 7.3$ % de v.

En revanche, plus p est choisi grand, plus le nombre de points de référence à exécuter sera important. Par conséquent, la durée de l'exploration sera plus longue.

Dans cette section, l'exploration est détaillée et appliquée sur *cinqHB*, une application composée de seulement cinq Homomorphes basiques. La section suivante utilise l'exploration sur trois applications regroupant n Homomorphes basiques avec $n = 22$, 138 et 640.

5.2 Intégration d'applications et résultats

La section précédente présente le fonctionnement de la méthode d'exploration automatisée. L'exploration permet à un novice en chiffrement homomorphe de pouvoir identifier le schéma le plus intéressant (temps d'exécution et/ou coût mémoire le plus faible) pour une application donnée.

Utilisée avec l'application *cinqHB* dans la section précédente, l'exploration peut être effectuée sur tout type d'application. Afin de montrer les capacités de l'exploration, trois autres applications ont été explorées, chacune avec un nombre d'opérations et une profondeur multiplicative différents.

Cette section introduit tout d'abord les applications avant de présenter les résultats obtenus après leur exploration.

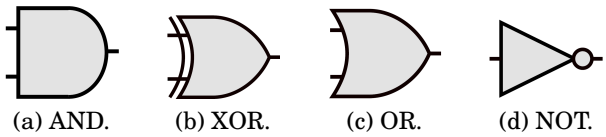
5.2.1 Définition des applications

Quatre applications ont été créées afin d'appliquer la méthode d'exploration sur plusieurs cas d'études. Les applications possèdent un nombre d'opérations et une profondeur multiplicative différents. Les caractéristiques détaillées de ces quatre applications, intégrées dans PAnTHErS, sont présentées dans la Table 5.8. Cette Table récapitule le nombre d'Homomorphes basiques effectuées et la profondeur multiplicative de chaque application.

TABLE 5.8 – Détail des applications (nombre d'Homomorphes basiques appelés et profondeur multiplicative)

Applications	*cinqHB*	*Pédagogique*	*Croissant*	*Médicale*
Nb de *KeyGen*	1	1	1	1
Nb de *Encrypt*	1	3	35	235
Nb de *Add*	1	5	80	294
Nb de *Mult*	1	12	14	106
Nb de *Decrypt*	1	1	8	4
Nb total d'Homomorphes basiques	5	22	138	640
Profondeur	1	10	6	12

Chaque application est présentée plus en détail ci-après. Une définition de chaque application est donnée ainsi qu'une représentation de leur circuit booléen respectif. Pour cela, les notations suivantes sont valables pour chacune de ces représentations. Chaque *input* (entrée), en vert, représente un message chiffré avec l'Homomorphe basique *Encrypt*.



(a) AND. (b) XOR. (c) OR. (d) NOT.

FIGURE 5.7 – Représentation graphique des opérations booléennes AND, XOR, OR et NOT.

Les opérations AND (resp. XOR) correspondent aux multiplications (resp. additions) homomorphes. Les opérations NOT et OR correspondent aux opérations homomorphes suivantes :

$$\begin{array}{rcl} NOT(a) & \Leftrightarrow & Add(a, Encrypt(1)) \\ OR(a,b) & \Leftrightarrow & Add(Add(a,b), Mult(a,b)) \end{array} \tag{5.6}$$

avec a et b des messages chiffrés.

La Figure 5.7 illustre les opérations booléennes utilisées pour les représentations graphiques des applications en circuits booléens.

Les *output* (sorties) sont ensuite déchiffrées avec l'Homomorphe basique *Decrypt*.

cinqHB

L'application *cinqHB*, décrite dans la section 3.2.6 du chapitre 3, a été explorée dans la section précédente. Les résultats de son exploration y sont détaillés.

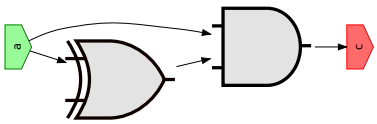


FIGURE 5.8 – Représentation de l'application *cinqHB* sous forme de circuit booléen.

Le circuit booléen de l'application est représenté dans la Figure 5.8. L'application correspond a un enchaînement des cinq Homomorphes basiques d'un schéma de chiffrement homomorphe.

Pédagogique

L'application *Pédagogique* représente un nombre arbitraire d'opérations homomorphes. Elle ne remplit aucune fonction utile pour un utilisateur. Elle a été conçue uniquement dans le but de pouvoir évaluer une application possédant une profondeur multiplicative plus importante que celle de *cinqHB*. L'application pédagogique possède ainsi une profondeur multiplicative de 10 contre seulement 1 pour *cinqHB*.

Le circuit booléen de l'application *Pédagogique* est illustré dans la Figure 5.9. Cette application prend en entrée trois messages clairs sous la forme de polynômes. Les 3 messages m_1, m_2 et m_3 sont chiffrés en c_1, c_2 et c_3. Ensuite, les opérations suivantes sont effectuées :

$$c = c_1^3 \cdot c_2^2 \cdot c_3^2 \cdot S \cdot T \cdot (S + c_3) \cdot (S + c_2 + c_3) \tag{5.7}$$

avec $S = c_1 \cdot T + c_1 + c_2$ et $T = c_1 \cdot c_2 + c_3$. Le résultat c est alors déchiffré.

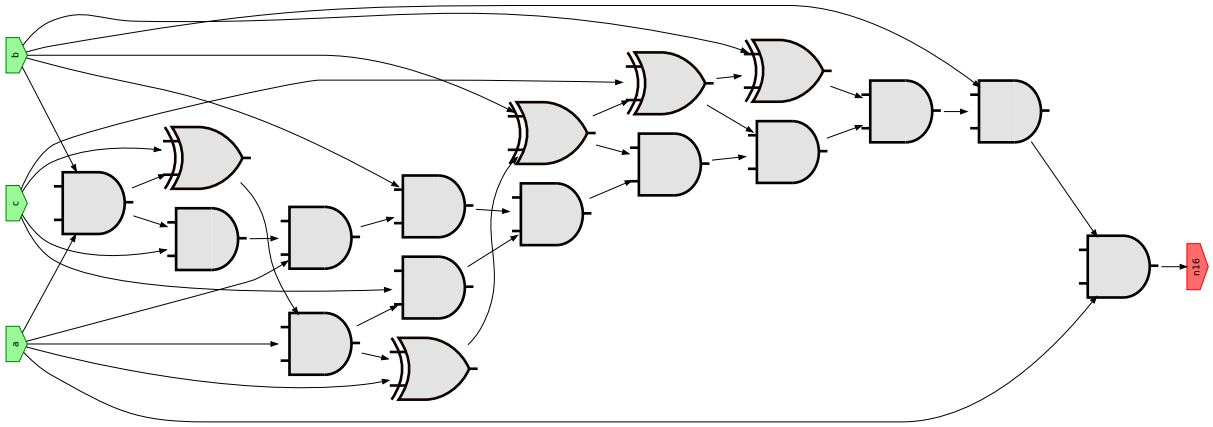


FIGURE 5.9 – Représentation de l'application *Pédagogique* sous forme de circuit booléen.

Croissant

L'application *Croissant* permet de savoir combien de dizaines de kcal une personne va consommer en mangeant n croissants, en considérant le paramètre d'entrée n codé sur 8 bits. Cette application correspond à une fonctionnalité de Samsung Health présentée dans la section 2.1.4 du chapitre 2. En effet, Samsung Health propose à ses utilisateurs de connaître le nombre de calories accumulées dans la journée en fonction des aliments qu'ils ont consommés. Cependant, telle qu'est conçue l'application, les données sont traitées en clair et Samsung a connaissance des aliments consommés par l'utilisateur. L'application *Croissant* proposée ici est plus simple, par rapport à l'application de Samsung Health, car elle est seulement applicable sur des croissants.

Le traitement effectué par l'application *Croissant* se veut simple. Sachant que la consommation d'un croissant équivaut à 231 kcal, une simple multiplication est nécessaire pour obtenir la quantité finale équivalente. Aussi, pour simplifier les traitements, l'application arrondit les quantités à la dizaine de kcal. Un croissant sera donc ici équivalent à 23 dizaines de kcal. L'application finale se résume donc à prendre en entrée une quantité n de croissants, puis à multiplier cette quantité par 23.

Les schémas FV [FV12] et YASHE [BLLN13] ont une variable t correspondant au module des textes clairs (chaque texte clair est réduit modulo t). Ce paramètre est d'ailleurs variable dans l'exploration des schémas. Cependant, le résultat final de l'application *Croissant* doit être un entier supérieur à 23. Par conséquent, si t est choisi petit (*p. ex.* $t = 2$), l'application ne déchiffrera pas la bonne valeur car le résultat final sera tronqué modulo t. D'un autre côté, si t est choisi grand (> 23), il doit être pris suffisamment grand pour répondre correctement à un utilisateur. Par exemple, si un utilisateur mange 5 croissants, l'application doit pouvoir retourner la valeur $5 \times 23 = 115$ et donc le schéma devra utiliser une valeur de $t > 115$. Par conséquent, pour garantir un bon fonctionnement, la valeur t choisie va restreindre la valeur maximale du paramètre n saisi par un utilisateur. C'est pourquoi il est important de choisir judicieusement cette valeur t.

Une méthode pour contrer cette limitation est d'effectuer les opérations homomorphes sur les bits des entrées de l'application. En agissant directement sur les bits, il n'y a pas restriction sur le paramètre d'entrée. En revanche, cela implique que le paramètre t devra toujours être égal à 2 car chaque message de sorti sera un bit (soit 0, soit 1).

Afin d'appliquer cette méthode, l'application *Croissant* doit être transformée en un circuit booléen. L'outil Cingulata [Tea17] permet de convertir une application codée à haut niveau en C++, en un circuit booléen, composé exclusivement de portes OR, XOR, AND

et NOT. Le circuit ainsi converti est sauvegardé dans un fichier texte au format BLIF [2] [Ber92]. Nous avons donc codé l'application *Croissant* en C++ tel que décrit dans l'Algorithme 18.

D'autres outils existent et peuvent être utilisés afin de réaliser la conversion d'une application à haut niveau (langage C/C++) en un circuit booléen dont, par exemple, l'outil Madeo [LPVG03].

Algorithme 18 Code source C++ de l'application Croissant

```
1: Integer8 nb_croissant, c;          //création de deux variables de type
2:                                    //Integer8 (entiers sur 8 bits)
3: cin>>nb_croissant;                 //lecture du nombre de croissants
4: c = nb_croissant*Integer8(23);     //calcul de la quantité de kcal consommés
5:                                    //(en dizaines de kcal)
6: cout<<c;                           //sortie du résultat
7: FINALIZE_CIRCUIT(blif_name);       //circuit BLIF mis dans le fichier blif_name
```

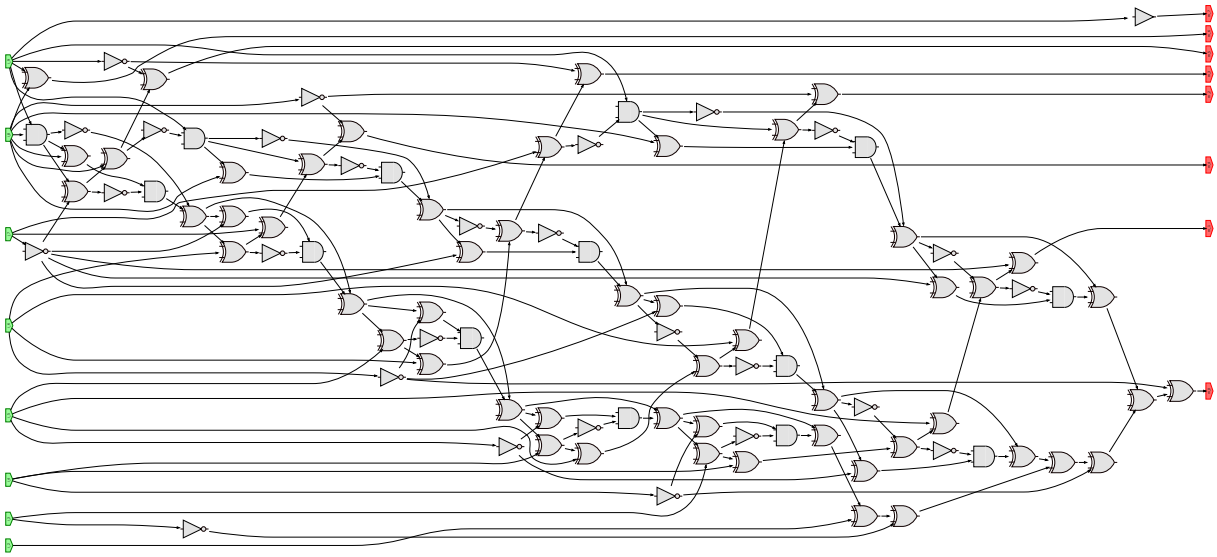


FIGURE 5.10 – Représentation de l'application *Croissant* sous forme de circuit booléen.

L'outil Cingulata fourni le circuit booléen de l'application *Croissant* à partir de sa description en langage C++. Cingulata retourne une version optimisée et non-optimisée du circuit booléen, tous les deux au format BLIF. Nous avons récupéré la version optimisée du circuit booléen au format BLIF afin de le convertir en un format plus compréhensible et utilisable dans PAnTHErS. L'équation 5.8 présente la démarche employée pour convertir chaque opération booléenne. Les opérations au format BLIF, présentées à gauche de l'accolade, sont converties vers l'instruction équivalente située à droite de l'accolade.

$$
\begin{array}{l|l}
\left.\begin{array}{l}.names\ a\ b\ c\\ 11\ \ 1\end{array}\right\} \Leftrightarrow c = AND(a,b) & \left.\begin{array}{l}.names\ a\ b\ c\\ 10\ \ 1\\ 01\ \ 1\end{array}\right\} \Leftrightarrow c = XOR(a,b) \\
\left.\begin{array}{l}.names\ a\ b\ c\\ 10\ \ 1\\ 01\ \ 1\\ 11\ \ 1\end{array}\right\} \Leftrightarrow c = OR(a,b) & \left.\begin{array}{l}.names\ a\ b\\ 0\ \ 1\end{array}\right\} \Leftrightarrow c = NOT(a,b)
\end{array}
\tag{5.8}
$$

2. Berkeley Logic Interchange Format.

L'enchaînement optimisé des opérations OR, XOR, NOT et AND de l'application *Croissant*, obtenu après conversion, est illustré dans la Figure 5.10. Dans PAnTHErS, les opérations booléennes sont redéfinies afin qu'elles fassent référence aux Homomorphes basiques des schémas.

Médicale

L'application *Médicale* correspond à l'application présentée dans [CNS+16]. Cette application vise à calculer un facteur de risque de maladie cardiaque à partir des informations médicales d'un patient. Le facteur calculé est un chiffre compris entre 0 (risque faible) et 9 (risque élevé). Le pseudo-code de l'application *Médicale* est détaillé dans l'Algorithme 19.

Algorithme 19 Pseudo-code de l'application *Médicale* (décrite dans [CNS+16] et disponible dans [Tea17])

Entrées : sexe, age, antecedents, fumeur, diabetique, cholesterolHDL, hypertension, poids, consommationAlcool, tempsSport, hauteur
Sorties : facteurRisque (compris entre 0 et 9)

```
1:  facteurRisque = 0
2:  Si sexe == HOMME ET age > 50 :
3:          facteurRisque += 1
4:  Si sexe == FEMME ET age > 60 :
5:          facteurRisque += 1
6:  Si antecedents dans la famille :
7:          facteurRisque += 1
8:  Si fumeur :
9:          facteurRisque += 1
10: Si diabetique :
11:         facteurRisque += 1
12: Si hypertension :
13:         facteurRisque += 1
14: Si cholesterolHDL < 40 :
15:         facteurRisque += 1
16: Si tempsSport par jour < 30 min :
17:         facteurRisque += 1
18: Si poids - 10 > taille :
19:         facteurRisque += 1
20: Si sexe == HOMME ET consommationAlcool > 3 verres/jour :
21:         facteurRisque += 1
22: Si sexe == FEMME ET consommationAlcool > 2 verres/jour :
23:         facteurRisque += 1
```

De la même manière que pour l'application *Croissant*, l'application *Médicale* a été transformée en un circuit booléen par l'outil *Cingulata* [Tea17]. Le circuit booléen final est représenté dans la Figure 5.11. Par la suite, pour l'exploration de cette application, le paramètre t sera fixé à 2 comme pour l'application *Croissant*.

Conclusion

Afin de valider rapidement la méthode de l'exploration, nous avons créé les applications *cinqHB* et *Pédagogique* qui possèdent un faible nombre d'opérations homomorphes (< 22).

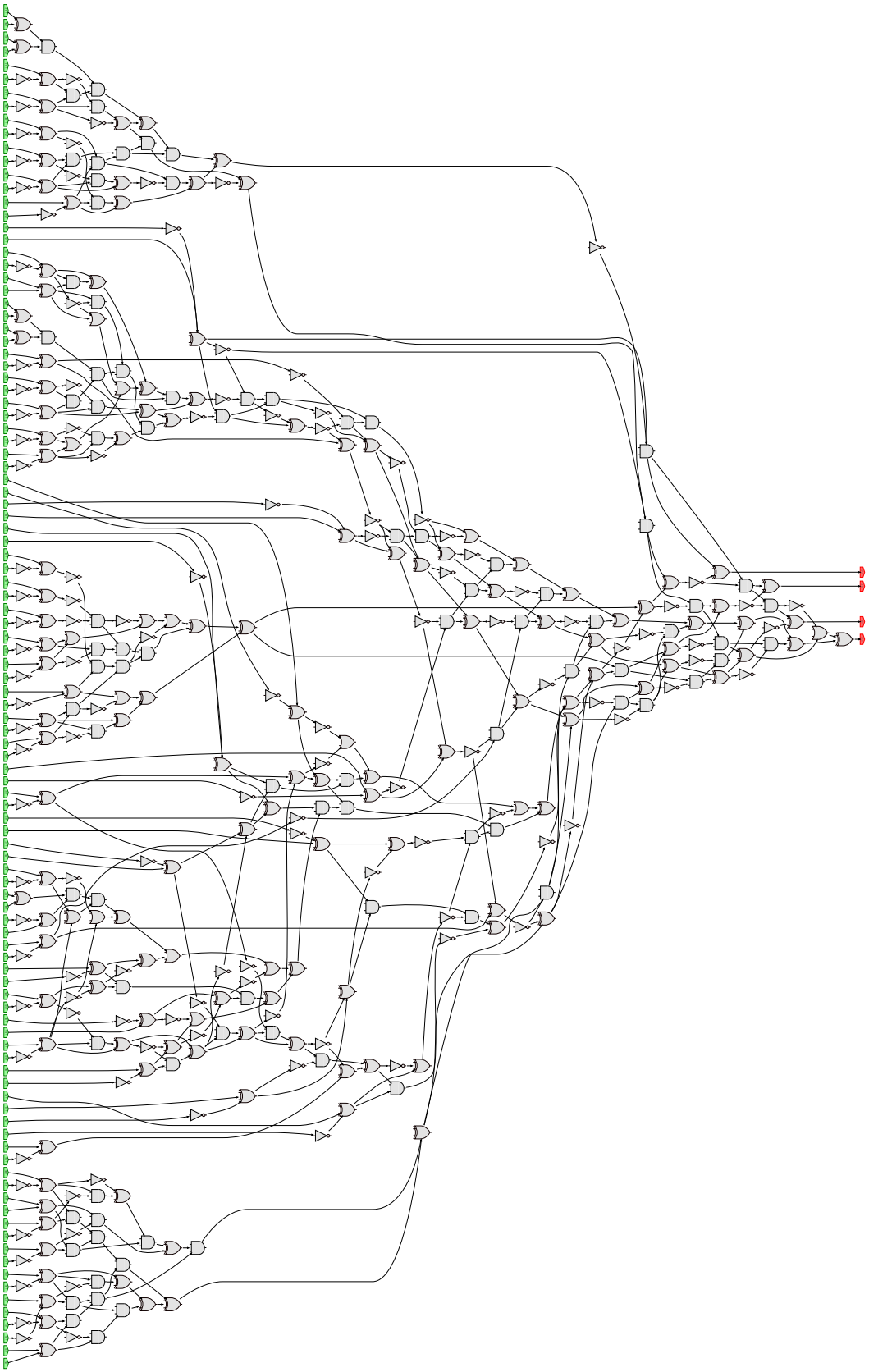


FIGURE 5.11 – Représentation de l'application *Médicale* sous forme de circuit booléen.

Toutefois, toute application est représentable sous la forme d'un circuit booléen composé de AND, XOR, OR et NOT. La conversion d'une application (codée en C++) vers un circuit booléen est réalisable avec l'utilisation de l'outil Cingulata [Tea17]. Les opérations booléennes du circuit peuvent ensuite être interprétées dans le domaine homomorphe comme des opérations (ou enchaînement d'opérations) Homomorphes basiques.

Dans le cas d'une application convertie en circuit booléen, les schémas utilisés pour celle-ci devront prendre une valeur t (module de texte clair) égale à 2. Par exemple, pour les applications *Croissant* et *Médicale*, le paramètre t sera toujours égal à 2 pour les schémas FV et YASHE.

Limitations

Le nombre d'Homomorphes basiques explose lorsque qu'une application est transformée en un circuit booléen. Par exemple, l'application *Croissant*, écrite en C++, correspond à une simple multiplication entre entiers de 8 bits. En homomorphe, cette application représente un enchaînement de 138 Homomorphes basiques. Si nous avions pris des entiers codés sur 16 bits pour cette application, celle-ci aurait dû effectuer 262 Homomorphes basiques, soit quasiment le double.

Cependant, l'utilisation d'entiers sur 8 bits réduit fortement le nombre de valeurs possibles pour le paramètre d'entrée n (le nombre de croissant). Comme il est codé sur 8 bits, le résultat de l'application *Croissant* ne pourra pas dépasser 256 dizaines de kcal, ce qui équivaut à 11.13 croissants. Cette application ne pourra donc pas prendre en entrée une valeur n supérieure à 11, au risque de renvoyer un résultat erroné.

5.2.2 Résultats de l'exploration des applications

Cette section présente les résultats de la méthode d'exploration de trois schémas (FV [FV12], YASHE [BLLN13] et F-NTRU [DS16]) sur les applications *Pédagogique*, *Croissant* et *Médicale*. La même configuration logicielle et matérielle détaillée dans la Table 5.1 a été utilisée pour les explorations et exécutions des applications. Les plages d'exploration des paramètres pour chaque schéma de chaque application sont précisées dans la Table 5.9.

TABLE 5.9 – Bornes minimales et maximales atteintes sur l'ensemble des jeux de paramètres d'entrée de chaque schéma, trouvées après exploration de chaque application.

Application	**Schéma**	$\log(q)$		$\log(w)$		t	
		Min	**Max**	**Min**	**Max**	**Min**	**Max**
Pédagogique	FV	297	418	2	99	2	14
	YASHE	308	443	2	99	2	5
	F-NTRU	102	149	2	7	-	-
Croissant	FV	189	287	2	99	2	2
	YASHE	196	308	2	99	2	2
	F-NTRU	61	149	2	19	-	-
Médicale	FV	376	476	2	99	2	2
	YASHE	389	499	2	99	2	2
	F-NTRU	121	149	2	4	-	-

La Table 5.10 renseigne le nombre de paramètres explorés, le temps d'exécution de l'exploration et le temps requis pour effectuer toutes les exécutions concrètes pour chaque application. Un facteur d'accélération est précisé pour chaque application. Ce facteur

TABLE 5.10 – Informations sur l'exploration réalisée pour chaque application : le nombre de jeux de paramètres d'entrée explorés, le temps d'exécution de l'exploration pour chaque application (temps pour trouver les jeux de paramètres et temps passé pour l'analyse de ces jeux de paramètres), le temps total des exécutions concrètes et le facteur d'accélération (ratio entre temps d'analyse et temps d'exécution concret).

Application	Nombre de jeux	Temps pour trouver les jeux	Durée d'exploration des paramètres	Temps total des exécutions	Accélération
Pédagogique	22969	2 h 49 min 37 sec	24 h 29 min 36 sec	1024 h 31 min 17 sec	41.8
Croissant	5684	2 h 27 min 41 sec	29 h 53 min 17 sec	289 h 16 min 2 sec	9.7
Médicale	5808	1 h 32 min 44 sec	319 h 29 min 6 sec	5872 h 9 min 56 sec	18.4
TOTAL	34461	6 h 50 min 2 sec	373 h 51 min 59 sec	7185 h 57 min 15 sec	19.2

représente le ratio entre le temps des exécutions concrètes et le temps d'exécution de l'exploration.

Le facteur d'accélération est compris entre 9.7 (*Croissant*) et 41.8 (*Pédagogique*). L'accélération de toutes les applications cumulées est de 19.2. Une étude par schéma de l'accélération produite des applications montre que F-NTRU (resp. YASHE) présente le plus petit (resp. grand) rendement. Par exemple, le facteur d'accélération de F-NTRU pour l'application *Médicale* est de 5.9 et celui de YASHE pour la même application est de 137.5 (voir Table A.9 dans l'Annexe A). Cette différence est principalement due aux nombres de jeux de paramètres analysés (26 seulement pour F-NTRU contre 2936 pour YASHE) et la durée des exécutions concrètes obligatoires pour la calibration : en moyenne, une exécution de l'application *Médicale* avec le schéma F-NTRU est 53 fois plus lent qu'avec le schéma YASHE.

Étude des taux d'erreur

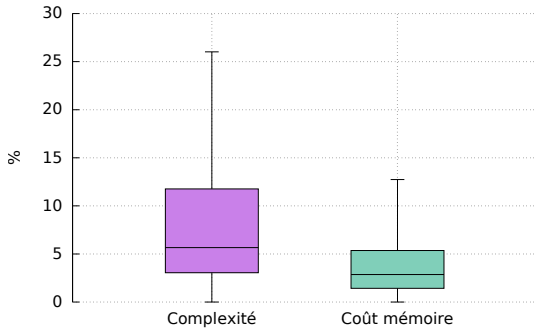
La Figure 5.12 présente la répartition des taux d'erreur constatés entre les analyses calibrées de l'exploration et les analyses concrètes, obtenus par exécution concrète. Les taux d'erreur de la Figure 5.12 sont représentés par exploration d'application *c-à-d* tous schémas confondus.

Plus de 95 % des taux d'erreur, pour chaque application, sont inférieurs à 45 %. De plus, la moyenne des taux d'erreur pour la complexité calibrée (resp. coût mémoire calibré) est égale à :

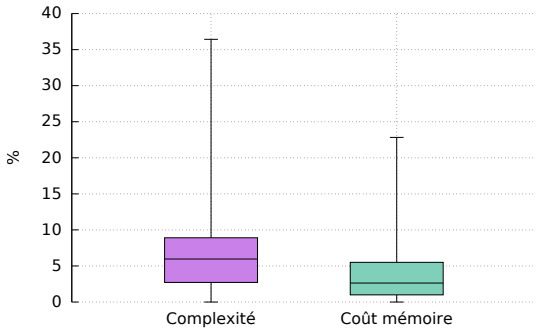
- 7.5 % (resp. 3.8 %) pour l'application *Pédagogique*,
- 7.4 % (resp. 5.1 %) pour l'application *Croissant*,
- 7.5 % (resp. 1.3 %) pour l'application *Médicale*.

Toutes ces moyennes sont bien inférieures à 25 %. Ces résultats sont conformes aux exigences fixées pour l'exploration dans la sous-section 5.1.2.

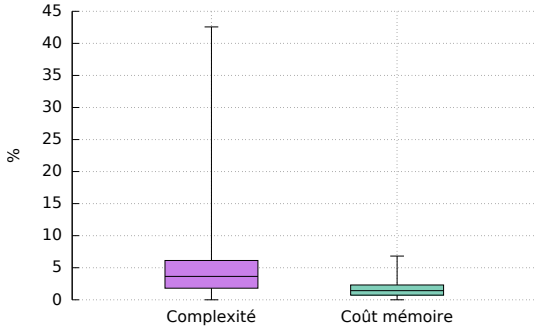
En considérant les taux d'erreur des applications pour chaque schéma distinct, les exigences sont également respectées sauf pour l'application *Croissant* avec le schéma F-NTRU (voir Figure A.3c dans l'Annexe A.2). Plus de la moitié des taux d'erreur engendrés par cette exploration sont supérieurs à 45 % pour la complexité calibrée. Concernant le coût mémoire calibré, seulement 70 % des valeurs sont effectivement inférieures à 45 %. Ces taux d'erreur peuvent être diminués en appliquant la perspective d'évolution citée dans la sous-section 5.1.3.



(a) *Pédagogique* (99 centiles).



(b) *Croissant* (97 centiles).



(c) *Médicale* (99 centiles).

FIGURE 5.12 – Boîte de Tukey des taux d'erreur obtenus pour les résultats de l'exploration comparés aux résultats des exécutions concrètes, pour chaque application et chaque analyse théorique calibrée (complexité et coût mémoire).

Étude des jeux de paramètres optimaux

Les jeux de paramètres obtenus par la méthode de l'exploration des applications sont listés dans la Table 5.11. Les jeux de paramètres sélectionnés par l'exploration sont tous des jeux de paramètres pour le schéma FV. Pour plus de détails, les résultats retournés par l'exploration pour chaque schéma sont listés dans l'Annexe A, dans la section A.1 pour l'application *Pédagogique*, la section A.2 pour l'application *Croissant* et la section A.3 pour l'application *Médicale*.

TABLE 5.11 – Résultats retournés par la méthode d'exploration des trois applications. Chaque jeu de paramètres sélectionné est associé au schéma FV.

Application	$\log(q)$	$\log(w)$	t	**Temps (sec)**			**Coût mémoire (Mio)**		
				Estim.	**Exéc.**	**Erreur**	**Estim.**	**Exéc.**	**Erreur**
Pédagogique	297	23	2	10.4223	10.2558	1.6235 %	99.3713	93.8	5.9395 %
	339	68	2	10.7477	11.5291	6.7776 %	83.9337	80.2	4.6555 %
Croissant	243	81	2	12.9071	13.1468	1.8229 %	39.2560	39.8	1.3668 %
Médicale	393	44	2	167.2526	175.2254	4.5500 %	269.5541	265.5	1.5270 %
	437	88	2	171.0225	167.4467	2.1355 %	237.2856	240	1.1310 %

Les jeux de paramètres optimaux sont obtenus suite aux exécutions concrètes de tous les schémas pour chaque application. Comme pour les jeux de paramètres sélectionnés par l'exploration, ces points optimaux correspondent tous à des jeux de paramètres pour

le schéma FV. Ce schéma offre une plus grande rapidité de calcul et une consommation mémoire moins importante que YASHE et F-NTRU.

TABLE 5.12 – Résultats retournés après l'exécution complète de tous les jeux de paramètres des schémas FV, YASHE et F-NTRU pour les trois applications étudiées. Seuls des jeux de paramètres pour le schéma FV sont retournés.

Application	$\log(q)$	$\log(w)$	t	**Temps (sec)**	**Coût mémoire (Mio)**
Pédagogique	307	35	2	10.3194	83.6
	310	35	2	10.1390	84.1
	312	40	2	10.3946	79.9
	313	38	2	10.2580	83.6
	314	40	2	10.3467	81.1
	315	40	2	10.3775	80.3
	315	43	2	10.3656	80.6
	320	46	2	10.5102	79.4
	328	56	2	11.3746	77
	331	57	2	10.8673	79.3
	337	57	3	10.9461	78.3
	338	63	2	11.0836	77
	342	70	2	11.0078	77.4
Croissant	253	89	2	11.8020	35.5
	265	94	2	11.5139	37.2
	276	99	2	11.6809	36.5
Médicale	412	58	2	150.9541	258.7
	449	84	2	153.4657	245.3
	449	90	2	154.9951	234.6
	451	98	2	154.3875	235.8
	460	99	2	153.5041	237.3

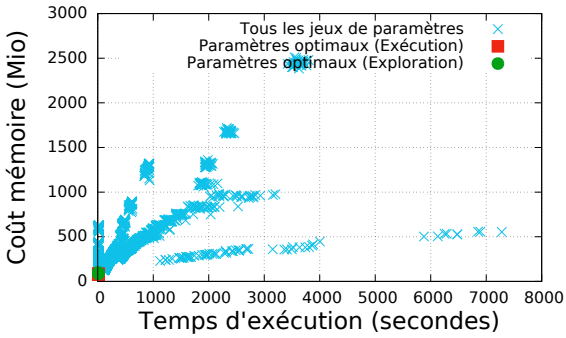
La Figure 5.13 représente la répartition des jeux de paramètres en fonction de leur temps d'exécution et de leur consommation mémoire, obtenus suite à leur exécution concrète. De plus, sur les Figures 5.13a et 5.13b sont placés les points optimaux dans des cercles rouges et les jeux de paramètres sélectionnés par l'exploration dans des carrés verts. Les carrés verts se situent dans le coin bas-gauche des graphes, montrant ainsi que les résultats de l'exploration sont proches des points optimaux. Par la suite, un classement des jeux de paramètres détaille le positionnement effectif des résultats obtenus.

La Table 5.13 donne le classement de chaque résultat obtenu par l'exploration. Le classement des jeux de paramètres est déterminé à partir des résultats d'exécutions concrètes (temps d'exécution et coût mémoire) et de leur indice de front de Pareto (Définition 18). Ce classement est calculé, d'un côté, avec l'équation 5.4 (indice de classement sur la totalité des jeux de paramètres explorés) et, d'un autre côté, avec l'équation 5.5 (indice de classement en pourcentage, *p. ex.* 0 % correspond à la 1ère place du classement et 100 % à la dernière place).

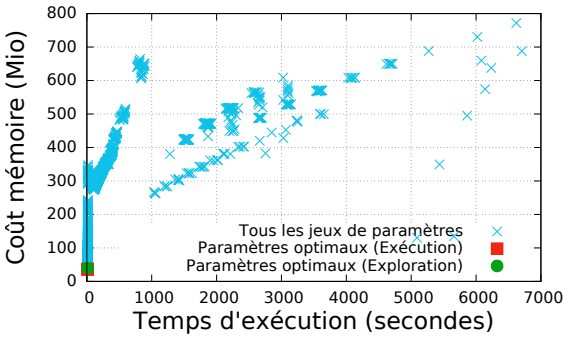
La Table 5.13 renseigne également de l'écart visible entre :

• le temps d'exécution d'un jeu de paramètres et le temps minimal d'exécution constaté pour les points optimaux,

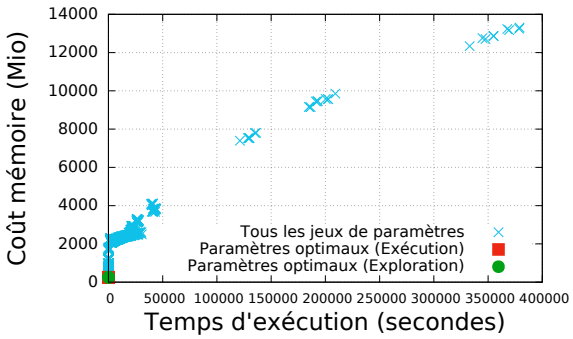
• la consommation mémoire d'un jeu de paramètres et le coût mémoire minimal constaté pour les points optimaux.



(a) *Pédagogique*.



(b) *Croissant*.



(c) *Médicale*.

FIGURE 5.13 – Identification des points retournés par l'exploration. Les croix bleues représentent les résultats d'analyse obtenus suite à l'exécution concrète des jeux de paramètres. Les carrés rouges correspondent aux points optimaux (*c-à-d* identifiés sur le front de Pareto) des exécutions (croix bleues). Les ronds verts représentent les résultats sélectionnés lors de la phase de l'exploration.

Pour les trois applications cumulées, 4 jeux de paramètres sur 5 sont positionnés dans les six premiers centiles de l'ensemble des jeux de paramètres explorés de leur application. Parmi ces jeux de paramètres, 2 sont dans les trois premiers centiles.

L'application *Médicale* possède un jeu de paramètres situé au delà des 20 premiers centiles du classement final. Une meilleure estimation des temps d'exécution et des coûts mémoire aurait permis d'obtenir des résultats d'exploration plus pertinents et ainsi écarter ce résultat trop éloigné de l'optimal. Comme présenté dans les perspectives 5.1.3, il est possible d'améliorer cette estimation en affinant la phase de calibration par l'ajout de points de référence à exécuter.

Conclusion

Parmi les 5 jeux de paramètres sélectionnés par l'exploration (toutes applications confondues), 3 sont proches de l'optimal *c-à-d* situés dans les cinq premiers centiles du classement des jeux de paramètres déterminé à partir des résultats des exécutions concrètes (temps d'exécution et coût mémoire) et de leur indice de front de Pareto (Définition 18). Seulement un jeu de paramètres possède un indice de classement important : il est positionné après les 20 premiers centiles du classement. En affinant la phase de calibration, l'exploration pourrait estimer plus précisément les temps d'exécution et coûts mémoire des schémas pour chaque application, et ainsi proposer des jeux de paramètres situés dans les cinq premiers centiles du classement des jeux de paramètres.

TABLE 5.13 – Classement des points sélectionnés par l'exploration des applications et écart de ces points avec le résultat optimal obtenu par exécution concrète.

Schéma	$\log(q)$	$\log(w)$	t	**Classement**		**Écart avec le point optimal**	
				Position	**En %**	**Temps (sec)**	**Coût mémoire (Mio)**
Pédagogique	297	23	2	119/22969	0.5137	0.1168	16.8
	339	68	2	736/22969	3.2	1.3902	3.2
Croissant	243	81	2	309/5684	5.4187	1.6328	4.3
Médicale	393	44	2	1734/5808	29.8382	24.2713	30.9
	437	88	2	133/5808	2.2727	16.4926	5.4

Dans l'objectif d'explorer d'autres applications et/ou d'autres schémas, ceux-ci doivent d'abord être intégrés dans la bibliothèque de l'outil PAnTHErS. Pour cela, la section suivante présente les différents usages de l'outil ainsi que les exigences établies lors de sa conception.

5.3 Exigences et usage de PAnTHErS

La modélisation, l'analyse théorique, la calibration et l'exploration de paramètres des schémas de chiffrement homomorphe sont montrées réalisables dans les chapitres et section précédentes. Le chapitre 3 détaille la modélisation et, en particulier, les analyses de complexité algorithmique et consommation mémoire. Le chapitre 4 présente la calibration et les résultats de son application sur trois schémas. La section 5.1 de ce chapitre décrit les différentes étapes de l'exploration des paramètres d'entrée de schéma. Ces différentes méthodes ont été automatisées en un logiciel que nous avons baptisé PAnTHErS pour *Prototyping and Analysis Tool for Homomorphic Encryption Schemes*.

Cette section définit les exigences de PAnTHErS. Différents cas d'usage sont alors posés en fonction de l'expertise d'un utilisateur de PAnTHErS. L'un des objectifs est de pouvoir répondre à la demande d'un utilisateur novice en cryptographie jusqu'à celle d'un expert en chiffrement homomorphe.

5.3.1 Exigences

Les schémas modélisés et les méthodes d'analyse théorique, de calibration et d'exploration ont été rassemblés en un outil appelé PAnTHErS, implémenté en Python avec l'utilisation du logiciel Sage [SAA16]. PAnTHErS vise à répondre à la demande de trois types d'utilisateurs :

• Un concepteur voulant utiliser du chiffrement homomorphe dans sa prochaine application ($\mathcal{U}_1$). Cet utilisateur n'est pas forcément expert en cryptographie.

• Un expert en chiffrement homomorphe cherchant à analyser et/ou améliorer son schéma ($\mathcal{U}_2$). En effet, l'analyse de son schéma lui permet de mettre en avant les algorithmes les plus coûteux, que ce soit en temps ou en mémoire. Suite à cela, l'expert pourra alors comparer les performances de son schéma face à d'autres algorithmes homomorphes.

• Un développeur expert en chiffrement homomorphe souhaitant intégrer de nouvelles analyses dans PAnTHErS ($\mathcal{U}_3$).

Afin de répondre à la demande du concepteur $\mathcal{U}_1$, PAnTHErS doit être accessible et doit permettre un usage qui ne requiert pas de compétences particulières. La seule tâche du concepteur $\mathcal{U}_1$ est d'intégrer son application dans l'outil. Pour faciliter l'utilisation de PAnTHErS, une interface graphique a été implémentée avec la bibliothèque TKinter [Shi13].

TABLE 5.14 – Algorithmes atomiques et spécifiques présents dans la bibliothèque de PAnTHErS.

Atomiques	Multiplication	Addition	Division	Soustraction
	Tirage aléatoire	Modulo	Arrondi	Puissance
	Produit scalaire	Inverse	Décomposition binaire	
Spécifiques	*distriLWE*	*doubledistriLWE*	*changeMod*	*addTimes*
	WordDecomp	*WordDecompInv*	*doubleMod*	*pubKey*
	PowersOf	*prodScalMod*	*Flatten*	*randMultMod*
	prodOfAdd	*doubleMultInv*	*modCenterInZero*	

Cette interface doit pouvoir servir non seulement pour le concepteur $\mathcal{U}_1$ mais également pour l'expert en chiffrement $\mathcal{U}_2$ lorsque ce dernier doit effectuer plusieurs analyses de schémas. L'interface a donc été adaptée afin de répondre aux deux demandes. De plus, pour $\mathcal{U}_1$ et $\mathcal{U}_2$, l'intégration d'applications et de schémas dans la bibliothèque de PAnTHErS doit être rapide à prendre en main.

Enfin, aucune nouvelle exigence pour PAnTHErS n'est requise pour le développeur $\mathcal{U}_3$. Celui-ci devra toutefois se familiariser avec l'architecture logicielle de PAnTHErS afin d'y intégrer ses propres modules d'analyse.

5.3.2 Bibliothèque et fonctionnalités de PAnTHErS

Cette sous-section présente tout d'abord la bibliothèque de PAnTHErS [Fer18a] et les fonctionnalités disponibles depuis son interface.

Bibliothèque

Listés dans le chapitre 3, les algorithmes atomiques et spécifiques sont stockés dans la bibliothèque de PAnTHErS. En tout, 11 algorithmes atomiques et 15 algorithmes spécifiques sont intégrés dans le logiciel. Ces algorithmes sont listés dans la Table 5.14.

La section 3.1.2 présente dans la Table 3.1 les schémas qui utilisent les 15 Spécifiques implémentés dans PAnTHErS. Les Spécifiques *modCenterInZero* et *doubledistriLWE* ne sont pas répertoriés dans la Table 3.1 car ils ne sont utilisés qu'une fois dans resp. F-NTRU [DS16] et FV [FV12].

En plus des algorithmes atomiques et spécifiques, les schémas FV [FV12], YASHE [BLLN13] et F-NTRU [DS16] ont été modélisés et intégrés dans PAnTHErS.

Interface graphique et exemples d'utilisation

L'interface graphique permet de simplifier l'utilisation de PAnTHErS, et ainsi rendre l'outil plus accessible pour un utilisateur. Elle permet, d'une part, de faciliter la configuration des analyses et/ou application en proposant la sélection de schémas à évaluer ainsi que la saisie des jeux de paramètres à appliquer. Lors de l'exécution, l'interface fournit des informations concernant la progression des analyses en cours. D'autre part, en fin d'évaluation, elle permet de visualiser des résultats sous forme de graphes.

Bien que sa forme et son style graphique soient rudimentaires, l'interface graphique de PAnTHErS, implémentée avec la bibliothèque TKinter [Shi13], est fonctionnelle et est représentée dans la Figure 5.14. L'utilisation de l'interface se décompose en plusieurs étapes dont les premières sont le choix du type d'analyse (complexité, coût mémoire, exécution, exploration) et le choix de l'application.

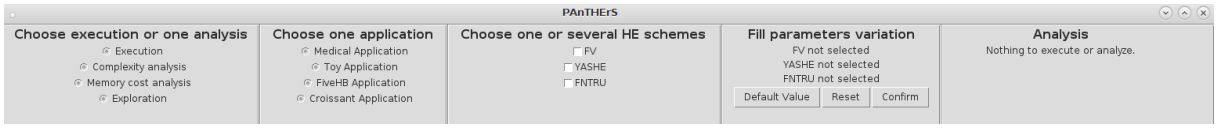


FIGURE 5.14 – Interface graphique de PAnTHErS.

Si l'utilisateur souhaite exécuter un schéma (ou application) ou d'en analyser la complexité ou le coût mémoire, il peut facilement sélectionner le ou les schémas voulus, ainsi que, pour chaque schéma, les variations des paramètres d'entrée. Pour chaque paramètre, l'utilisateur doit renseigner son intervalle de variation (valeur minimale, maximale et pas d'avancement), ainsi que sa valeur par défaut, utilisée la valeur par défaut, utilisée lors de la variation d'un autre paramètre d'entrée. Lorsque les valeurs des paramètres ont été validées, il est possible de vérifier si toutes les combinaisons de valeurs engendrent bien la profondeur multiplicative de l'application choisie.

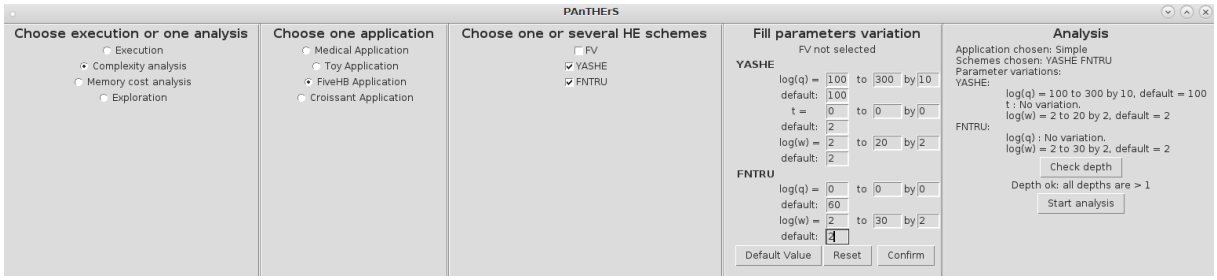


FIGURE 5.15 – Analyse de la complexité de YASHE et F-NTRU configurée via l'interface graphique dePAnTHErS.

La Figure 5.15 présente un exemple d'utilisation de l'interface pour effectuer une analyse de complexité pour l'application *cinqHB* (*fiveHB*, en anglais sur l'interface) avec les schémas YASHE et F-NTRU. Pour cette analyse, l'utilisateur fait varier :

- pour le schéma YASHE, $\log(q)$ de 100 à 300 par pas de 10 en fixant $\log(w) = 2$ et $t = 2$.
- pour le schéma YASHE, $\log(w)$ de 2 à 20 par pas de 2 en fixant $\log(q) = 100$ et $t = 2$.
- pour le schéma F-NTRU, $\log(w)$ de 2 à 30 par pas de 2 en fixant $\log(q) = 60$.

Avec l'utilisation de *Check Depth*, l'interface confirme que toutes les combinaisons de jeux de paramètres engendrent bien la profondeur multiplicative de l'application *c-à-d* 1.

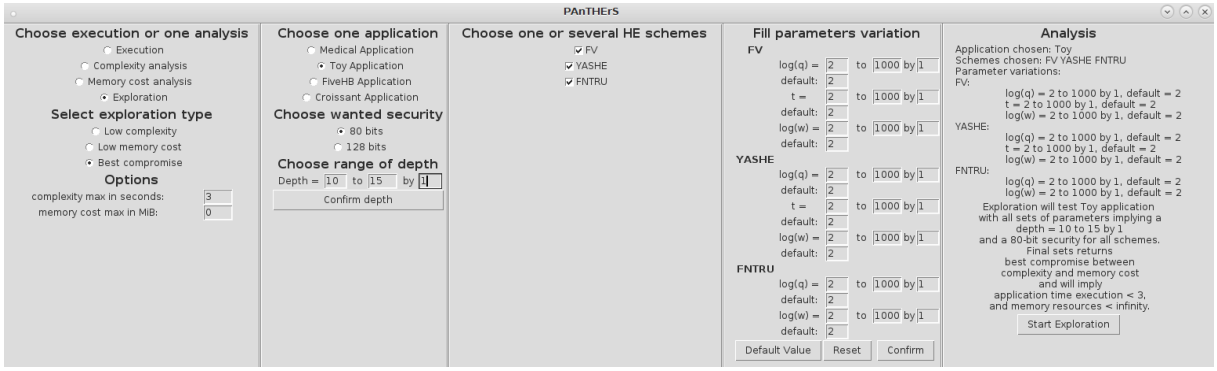


FIGURE 5.16 – Exploration des paramètres des schémas configurée via l'interface graphique de PAnTHErS.

Si l'utilisateur souhaite effectuer une exploration, il peut choisir s'il préfère des résultats favorisant un temps d'exécution le plus faible, une consommation mémoire la plus

faible ou un compromis des deux. De plus, il peut indiquer des bornes maximales acceptables pour les temps d'exécutions c_{max} et les coûts mémoire m_{max}. Autrement dit, l'exploration retiendra seulement les jeux de paramètres qui impliquent un temps d'exécution $< c_{max}$ et/ou une consommation mémoire $< m_{max}$. L'utilisateur peut également choisir un intervalle de profondeurs multiplicatives dont la valeur minimale est a minima celle de l'application. Enfin, l'utilisateur peut choisir d'exiger des paramètres d'entrée impliquant une sécurité de 80 bits ou de 128 bits. Pour différencier les deux types de sécurité, nous avons appliqué les paramètres q (module de messages chiffrés) et n (taille du polynôme quotient) d'après les travaux présentés dans [MBF17]. Une fois que l'utilisateur valide l'ensemble des paramètres, PAnTHErS génère automatiquement l'ensemble des jeux de paramètres à évaluer pour chaque schéma sélectionné.

La Figure 5.16 montre un exemple de configuration de l'exploration via l'interface graphique. Sur cette figure, l'utilisateur a configuré une exploration afin d'obtenir les schémas et leurs jeux de paramètres impliquant un temps d'exécution inférieur à 3 secondes pour l'application *Jouet* (présentée dans la section précédente). Pour ce faire, PAnTHErS va explorer tous les jeux de paramètres impliquant 80 bits de sécurité et une profondeur multiplicative allant de 10 à 15.

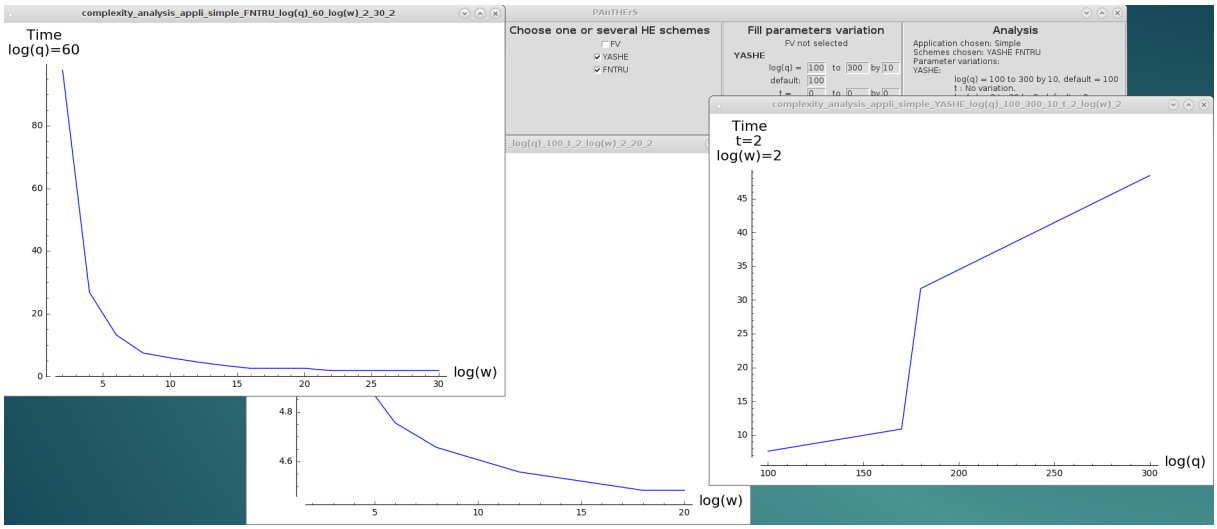


FIGURE 5.17 – Graphes résultant de l'analyse de complexité configurée dans la Figure 5.15.

En fin d'analyse, les résultats sont retournés à l'utilisateur sous forme de graphiques affichables depuis l'interface graphique (voir Figure 5.17). Les résultats sont également retournés sous la forme de différents fichiers textes au format CSV [3]. Un fichier est créé par type de résultat dont :

- un fichier contenant les résultats d'analyse théoriques,
- un fichier contenant les résultats d'analyses calibrées.

Les résultats de l'exploration sont seulement stockés dans des fichiers au format CSV, comprenant :

- un fichier contenant les résultats d'analyse théorique et la calibration de ces analyses (complexité et coût mémoire) pour tous les jeux de paramètres évalués.
- un fichier avec les jeux de paramètres retenus.

3. *Comma-Separated Values*, en anglais. Valeurs séparées par des virgules, en français. Un fichier au format CSV est de type tableur.

5.3.3 Utilisation de PAnTHErS

Cette sous-section présente les différents cas d'usage de PAnTHErS, décrits en fonction des trois profils d'utilisateurs introduits dans la sous-section 5.3.1 *c-à-d* le concepteur d'application $\mathcal{U}_1$, l'expert en chiffrement homomorphe $\mathcal{U}_2$ et le développeur expert $\mathcal{U}_3$.

Concepteur d'applications $\mathcal{U}_1$

Le concepteur et/ou développeur d'applications a pour but d'intégrer du chiffrement homomorphe dans une future application afin d'assurer la sécurité des données client. Pour cela, celui-ci devra d'abord intégrer son application dans PAnTHErS sous la forme d'un enchaînement d'algorithmes homomorphes basiques. Ensuite, le concepteur pourra utiliser la méthode d'exploration à l'aide de l'interface graphique.

Tout d'abord, une première étape est nécessaire pour le concepteur avant d'intégrer son application. Cette dernière doit être convertie sous la forme d'un enchaînement d'additions et de multiplications seulement. En effet, seules les additions et les multiplications peuvent être effectuées en homomorphe. Cette difficulté peut être résolue en utilisant l'outil Cingulata [Tea17]. Cet outil permet de convertir une application, codée en C++, en une suite d'opérations booléennes : AND, XOR, NOT, OR. Les opérations OR et NOT peuvent être ré-écrire comme une suite d'opération AND et XOR [4]. Ces deux opérations correspondront par la suite respectivement à une multiplication homomorphe et une addition homomorphe. Un exemple d'utilisation de Cingulata est donné dans la section précédente. Dans cet exemple, le code C++ de l'application *Croissant* est détaillé dans l'Algorihme 18 et le résultat sous la forme d'une suite d'opérations booléennes est illustré avec la Figure 5.10.

Après avoir converti son application, le concepteur peut l'intégrer dans PAnTHErS. Cette étape n'est pas automatique mais facilement réalisable à l'aide de patrons et de tutoriels [5].

Enfin, le concepteur peut utiliser le mode exploration de l'interface graphique pour rechercher les meilleurs schémas et jeux de paramètres pour son application. PAnTHErS retourne alors les résultats sous la forme de plusieurs fichiers au format CSV dont un contenant le schéma et les paramètres d'entrée les plus intéressants pour l'application choisie ainsi que les résultats d'analyse correspondants.

Expert en chiffrement homomorphe $\mathcal{U}_2$

L'expert en chiffrement homomorphe peut avoir deux objectifs :

1) Analyser son propre schéma pour identifier les algorithmes trop coûteux en temps et en mémoire, quitte à modifier certaines opérations de son schéma.

2) Comparer les performances de plusieurs schémas entre eux.

Dans un premier temps, l'expert doit d'abord prendre connaissance des algorithmes existants dans PAnTHErS, dont notamment les Spécifiques, et de leurs fonctionnalités. L'expert modélise ensuite son schéma sous la forme d'enchaînement d'algorithmes en maximisant l'utilisation des Spécifiques. Si besoin, l'expert peut également regrouper plusieurs séquences d'instructions entres elles afin de créer de nouveaux Spécifiques. Sans connaissance des Spécifiques, l'expert se réduit à l'utilisation des algorithmes atomiques et homomorphes basiques, ce qui rend moins efficace l'étape de modélisation de son schéma.

Dans un second temps, l'expert doit pouvoir intégrer son schéma modélisé dans PAnTHErS. Il pourra ainsi réutiliser les algorithmes présents dans la bibliothèque de

4. OR(a, b) = XOR(XOR(a, b), AND(a, b)) et NOT(a) = XOR$(a, 1)$.
5. Tutoriel "Adding a new Homomorphic application to PAnTHErS" [Fer18a] [Fer18b].

PAnTHErS, ou alors, s'ils ne s'y trouvent pas déjà, il devra y ajouter les algorithmes qui composent son schéma. Ces intégrations demandent à l'expert de connaître la structure des algorithmes (spécifiques et homomorphes basiques) de PAnTHErS. Des patrons, disponibles dans [Fer18a], sont mis à disposition et donnent la structure générale que doit respecter un schéma et un algorithme spécifique. L'utilisateur de PAnTHErS peut récupérer ces patrons et s'en servir de base pour intégrer ses propres schémas et algorithmes.

Dans un troisième temps, l'expert doit également intégrer dans PAnTHErS les algorithmes d'analyse (complexité et coût mémoire) pour les schémas et Spécifiques nouvellement créées. Un schéma (resp. un Spécifique) d'analyse correspond à un schéma (resp. un Spécifique) modélisé et implémenté dans PAnTHErS dont chaque appel de fonction f interne est remplacé par l'appel de la fonction d'analyse de f.

Par exemple, dans la sous-section 3.2.3 du chapitre 3, chaque algorithme exécutable présent dans l'Algorihme 6 appelé *distriLWE* est remplacé par l'appel de son algorithme d'analyse de complexité algorithmique pour obtenir *Complexity.distriLWE*, référencé comme Algorithme 14. Plus simplement, chaque appel d'algorithme A est remplacé par $Complexity.A$.

À ce jour, la création des fonctions d'analyse de Spécifiques ou d'Homomorphes basiques modélisés doit être effectuée à la main : néanmoins, cette étape pourrait être automatisée.

Comme pour le concepteur $\mathcal{U}_1$, l'expert doit intégrer son schéma dans l'interface en choisissant notamment quels paramètres d'entrée pourront varier. Cette étape peut être réalisée à l'aide d'un tutoriel [6] accompagnant l'outil PAnTHErS.

Une fois les différentes étapes d'intégration terminées, l'expert peut alors utiliser PAnTHErS pour évaluer son schéma en faisant varier ses paramètres d'entrée. L'interface graphique retourne alors les résultats associés aux variations et analyses choisies. Des fichiers au format CSV sont créés et contiennent le récapitulatif des paramètres analysés ainsi que leurs résultats d'analyse respectifs.

L'expert peut également confronter les performances de son schéma face à celles des schémas déjà présents dans PAnTHErS. Il pourra pour cela utiliser une application déjà existante, ou en intégrer une nouvelle comme présenté pour le concepteur $\mathcal{U}_1$.

Développeur expert en chiffrement homomorphe $\mathcal{U}_3$

Le développeur expert en chiffrement homomorphe est intéressée par l'ajout de nouvelles analyses dans PAnTHErS. Comme énoncé dans la section 3.2.5, d'autres métriques peuvent être intégrées dans PAnTHErS comme *p. ex.* l'analyse de l'évolution du bruit. Pour ce faire, celle-ci pourra s'appuyer sur les méthodes déjà réalisées (*p. ex.* complexité et coût mémoire).

Pour cela, la nouvelle analyse du développeur doit pouvoir être décomposée pour chacun des algorithmes atomiques, spécifiques et homomorphes basiques de PAnTHErS. Le développeur doit se conformer à la structure logicielle déjà établie. En effet, chaque composant de PAnTHErS est défini par un modèle.

Pour rappel, un modèle est à la fois une structure de données, ainsi qu'un ensemble de comportements, notamment des fonctions supportées par les objets manipulés. Par exemple, un schéma d'analyse de mémoire doit nécessairement proposer les fonctions d'opération courantes : *KeyGen*, *Encrypt*, *Decrypt*, *Add* et *Mult*.

De plus, un modèle est ouvert en extension *c-à-d* qu'il est possible de le compléter, et il reste fermé en modification pour éviter d'induire d'éventuelles régressions. Le développeur ne peut donc pas altérer l'architecture principale de PAnTHErS. Il peut toutefois proposer

6. Tutoriel "Adding a new Homomorphic Encryption scheme to PAnTHErS" [Fer18a] [Fer18b].

des extensions en se basant sur celle-ci. En effet, le développeur dispose du code existant (*p. ex.* le code de l'analyse de complexité) qui lui fournit un patron naturel qu'il peut réutiliser et adapter pour sa propre analyse.

Une étape importante est l'implémentation de l'analyse des algorithmes atomiques. Ceux-ci sont à la base des algorithmes spécifiques et homomorphes basiques. Une fois que les analyses des Atomiques sont réalisées, le développeur doit implémenter les fonctions d'analyses des Spécifiques. Pour cela, il peut récupérer les fonctions exécutables des Spécifique et remplacer l'appel des fonctions d'exécutions concrètes des Atomiques et/ou Spécifique par l'appel des fonctions d'analyses.

Cette étape pourrait être automatisée et fait partie des perspectives d'évolution de l'outil PAnTHErS. Bien qu'automatisable, le développeur doit vérifier les fonctions d'analyses des Spécifiques : en effet, certains algorithmes spécifiques pourraient être traités différemment en fonction des types d'instructions qui le compose et du type d'analyse réalisée. Par exemple, dans des versions exécutables d'algorithmes, certaines lignes d'instructions visent à modifier la valeur d'un paramètre. Cependant, ce type de lignes d'instructions n'a pas de sens pour l'analyse de la complexité algorithmique ou du coût mémoire qui ne gèrent pas les valeurs des paramètres. Par conséquent, ces lignes d'instructions doivent être retirées manuellement des algorithmes d'analyse.

5.3.4 Perspectives pour PAnTHErS

PAnTHErS est fonctionnel et dispose d'une interface graphique permettant une utilisation plus simple de l'outil. Néanmoins, certaines pistes d'évolution de l'outil peuvent dès à présent être envisagées.

Une première piste d'évolution serait de proposer à un utilisateur d'accéder depuis l'interface graphique à une analyse détaillée (profilage) pour chaque algorithme appelé dans les Homomorphes basiques d'un schéma (*p. ex.* obtenir la Figure 3.8).

Par ailleurs, l'implémentation actuelle de l'interface utilisateur ne laisse pas de liberté dans la manière d'effectuer la p-calibration. Les analyses sont alors réalisées avec une 2-calibration dont le choix des points est systématiquement fixé par la méthode présentée dans la sous-section 4.2.3. Une amélioration de l'interface pourrait permettre à l'utilisateur de choisir le nombre p de points de référence à exécuter pour la p-calibration, mais également de choisir ces points référence (*c-à-d* les jeux de paramètres pour lesquels le schéma sera exécuté concrètement).

Afin de faciliter la création de schémas et d'applications, une fonctionnalité d'ajout de schémas et d'applications pourrait être directement intégrée à l'interface graphique. Cette fonctionnalité renverrait à une autre interface qui permettrait de composer graphiquement de nouveaux schémas et/ou applications. Les schémas pourraient être créés en mettant, à la suite, des boîtes représentant les algorithmes atomiques et spécifiques disponibles dans PAnTHErS. Pour la création d'applications, l'utilisateur aurait directement accès à des boîtes représentant les Homomorphes basiques. Une fois nouveau schéma ou application validé, l'intégration dans l'interface graphique pourrait également être automatisée.

Enfin, pour faciliter le travail de l'expert en chiffrement homomorphe, suite à la création d'un nouveau schéma, un programme pourrait générer automatiquement les algorithmes d'analyse de ce schéma. De plus, lorsque l'expert a modélisé et intégré son schéma (ou application) dans l'outil PAnTHErS, celui-ci pourrait être ajouté automatiquement dans l'interface graphique.

5.4 Conclusion

La méthode d'exploration permet, pour une application donnée, d'analyser tous les schémas et jeux de paramètres d'entrée engendrant la profondeur multiplicative souhaitée, et de sélectionner les schémas produisant les résultats les plus intéressants. En l'occurrence, l'utilisateur peut choisir s'il préfère favoriser les schémas et jeux de paramètres induisant un temps d'exécution plus faible, une consommation mémoire plus basse ou un compromis entre ces deux facteurs.

L'exploration permet de répondre à la demande d'un concepteur, n'ayant pas de connaissance particulière dans le domaine du chiffrement homomorphe, qui souhaiterait intégrer dans sa future application le schéma de chiffrement homomorphe le moins coûteux (en temps comme en mémoire).

Bilan des résultats présentés dans ce chapitre

La méthode a été testée sur quatre applications possédant chacune des caractéristiques différentes. L'exploration de l'application *cinqHB* recommande d'utiliser le schéma YASHE qui présente les meilleurs temps d'exécution et de consommation mémoire pour le jeu de paramètre ($\log(q) = 46, \log(w), t$) = $(46, 10, 2)$. Pour les trois autres applications, l'exploration favorise le schéma FV avec plusieurs jeux de paramètres possibles.

Par rapport à l'exécution concrète de l'application avec les schémas et jeux de paramètres à explorer, l'exploration de tous les jeux de paramètres s'effectue :

- 7.4 fois plus vite, pour l'application *cinqHB*,
- 41.8 fois plus vite, pour l'application *Pédagogique*,
- 9.7 fois plus vite, pour l'application *Croissant*,
- 18.4 fois plus vite, pour l'application *Médicale*.

Les résultats de l'exploration, présentés dans les sections 5.1.3 et 5.2, possèdent des taux d'erreur satisfaisant les exigences fixées en amont : en l'occurrence, les moyennes des taux d'erreur ne dépassent pas les 25 % et plus de 95 % des taux d'erreur sont inférieurs à 45 %. Cependant, en décomposant les explorations par schémas, nous avons noté des écarts par rapport à ces exigences. En effet, pour l'application *cinqHB*, la moyenne des taux d'erreur pour le schéma YASHE dépasse les 25 % exigés de 0.4 %. De plus, pour l'application *Croissant*, plus de la moitié (resp. plus de 15 %) des taux d'erreur, pour la complexité (resp. pour le coût mémoire), engendrés par l'exploration du schéma F-NTRU sont supérieurs à 45 %. Une perspective d'amélioration serait l'ajout de points de référence à exécuter afin d'affiner les résultats de calibrations et de limiter ainsi la propagation d'erreur.

Suite à une exécution concrète complète de tous les jeux de paramètres explorés, un classement de ces jeux de paramètres a été réalisé à l'aide du front de Pareto. À partir de ce classement, nous avons pu identifier la position des résultats retournés à l'issue de chaque exploration, et ainsi évaluer leur optimalité (écart par rapport aux meilleures performances constatées).

Les résultats obtenus pour les quatre applications, présentés dans ce chapitre, sont tous situés dans les 15 premiers centiles, excepté un jeu de paramètres pour l'application *Médicale*. Plus de la moitié des résultats ($13/19$) sont situés dans les cinq premiers centiles. En particulier, ceux obtenus pour l'application *Pédagogique* sont tous positionnés dans les quatre premiers centiles du classement. En revanche, un seul jeu de paramètres (sur les deux retournés par l'exploration de l'application *Médicale*) est situé au delà des 20 premiers centiles. En appliquant la perspective 5.1.3 proposée, l'exploration pourrait retourner des jeux de paramètres plus proches de l'optimal (*c-à-d* des jeux de paramètres situés dans les cinq premiers centiles du classement) grâce à une calibration plus fine.

Usage de PAnTHErS

À travers un outil de modélisation et d'analyse nommé PAnTHErS, un utilisateur (concepteur, expert ou développeur expert) est capable d'analyser un schéma de chiffrement homomorphe ou une application. PAnTHErS donne la possibilité de lancer plusieurs types d'analyses : théorique ou concrète, par schéma ou par application, exploration partielle (variation d'un seul paramètre sur un intervalle donné) ou complète (variation de tous les paramètres). L'utilisateur choisit alors le type d'analyse qui lui convient le mieux en fonction de son niveau d'expertise.

Bénéfices et limitations de PAnTHErS

L'outil PAnTHErS est donc adapté pour différents profils d'utilisateurs qui souhaiteraient analyser leurs schémas et/ou applications. L'interface graphique permet, dès le départ, une prise en main simplifiée notamment pour l'évaluation de schémas et/ou applications déjà intégrées à l'outil PAnTHErS. Dans le cadre d'une évolution future de l'outil, plusieurs perspectives sont envisageables dont l'utilisation de la p-calibration pour $p > 2$ dans le module d'exploration. Les différentes perspectives d'évolution des fonctionnalités de l'outil PAnTHErS sont présentées dans le chapitre suivant.

Chapitre 6

Conclusion générale et perspectives

6.1 Conclusion générale

Le chiffrement homomorphe est prometteur et son utilisation dans un avenir proche est aujourd'hui envisageable. En effet, il permet à un utilisateur de déporter des traitements sur des données privées vers des serveurs distants sans risque de perte de confidentialité. De plus, le caractère post-quantique du chiffrement homomorphe lui permettrait de continuer à être utilisé, même après l'arrivée sur le marché d'ordinateurs quantiques.

Néanmoins, le domaine de l'homomorphe reste encore très jeune et souffre de nombreuses limitations dont :

- une sécurité des schémas non stable,
- une gestion de l'erreur non optimisée,
- un facteur d'expansion des chiffrés important,
- une grande complexité algorithmique,
- une importante consommation mémoire des schémas,
- une large diversité de schémas à comparer.

6.1.1 Objectifs de ce manuscrit

C'est notamment sur les trois derniers points cités précédemment que ce manuscrit apporte des contributions. Les travaux décrits dans ce manuscrit visent à effectuer des analyses rapides de schémas de chiffrement homomorphe, en matière de complexité algorithmique et de coût mémoire, à travers un outil baptisé PAnTHErS. D'abord exprimés de manière théorique, les résultats produits sont ensuite convertis en valeurs concrètes *c-à-d* en temps d'exécution (représentant la complexité algorithmique) et en mébioctets (pour la consommation mémoire). Ces résultats d'analyse peuvent permettre aux chercheurs, utilisateurs de cryptographie homomorphe, de cibler au mieux les opérations coûteuses (en temps et en mémoire) de leurs schémas pour, *in fine*, les optimiser ou les modifier afin de réduire leurs coûts. De plus, les résultats d'analyse permettent également la comparaison des schémas entre eux dans le but de sélectionner ceux engendrant un temps d'exécution plus faible, une consommation mémoire plus basse ou un compromis de ces deux facteurs. Cette comparaison a été automatisée afin qu'un expert, tout comme un non-expert, puissent mettre en avant le meilleur schéma et les paramètres d'entrée fournissant des performances les plus proches de l'optimal pour une application donnée.

6.1.2 Contributions présentées dans ce manuscrit

Afin de réaliser les objectifs cités ci-dessus, une première contribution, décrite dans la section 3.1 chapitre 3 de ce manuscrit et [FLL17], a été d'établir une modélisation des schémas de chiffrement homomorphe pour les représenter dans un format commun. La modélisation proposée, basée sur trois types d'algorithmes (atomique, spécifique et homomorphe basique), permet de modéliser tout schéma de chiffrement homomorphe en une combinaison de ces algorithmes. Appliquée dans un contexte précis dans ce manuscrit, la méthode de modélisation présentée reste flexible et peut être étendue pour modéliser un algorithme quelconque, d'un domaine autre que le chiffrement homomorphe. Ainsi, l'outil PAnTHErS, développé dans le cadre de cette thèse pour adresser les schémas homomorphes, pourrait également servir à modéliser des cryptosystèmes ou algorithmes, et par conséquent leur offrir une capacité d'évaluation pré-implémentation. Par exemple, les cryptosystèmes post-quantiques, proposés dans le cadre de la compétition organisé par le NIST (voir sous-section 2.1.5), pourraient être évalués et comparés entre eux grâce à cet outil.

Une fois les schémas représentés sous un même format, nous avons pu leur appliquer une méthode d'analyse théorique "pire cas" de complexité algorithmique et de coût mémoire (voir section 3.2 du chapitre 3 et [FLL17]). La méthode retenue consiste en l'analyse unitaire des algorithmes disponibles pour la modélisation des schémas *c-à-d* les Atomiques, les Spécifiques et les Homomorphes basiques. Ces analyses théoriques sont produites rapidement, car elles ne nécessitent pas l'exécution concrète des schémas pour chacun des jeux de paramètres analysés. Elles permettent de fournir une vision à haut niveau de l'évolution de la complexité algorithmique et du coût mémoire de ces schémas. Cependant, les résultats théoriques ainsi obtenus, bien qu'intéressants, ne sont pas représentatifs des temps d'exécution et coûts mémoire réels, obtenus lors de l'exécution concrète des schémas. Néanmoins, leur intérêt tient dans la fourniture d'une tendance d'évolution de ces indicateurs, pour des valeurs de paramètres d'entrée fixées.

Pour palier ce manque, une seconde contribution est l'introduction de la p-calibration dans le chapitre 4 et [FLL18]. Cette méthode vise à calibrer les résultats théoriques pire cas en résultats concrets dans le cas moyen. Pour cela, la p-calibration se base sur les résultats obtenus suite à l'exécution concrète d'un schéma pour p jeux de paramètres fixés, appelés points de référence. À partir de ces points de référence, les analyses théoriques de complexité algorithmique et de coût mémoire sont alors converties respectivement en temps d'exécution et en mébioctets. Toutefois, si un utilisateur souhaite effectuer une analyse, il doit choisir les intervalles de valeurs pour les paramètres d'entrée à évaluer. Par conséquent, l'utilisateur requiert un minimum de connaissance des schémas de chiffrement homomorphe.

Souhaitant aussi bien satisfaire la demande d'un expert comme d'un novice en cryptographie, nous avons intégré dans PAnTHErS une troisième contribution : l'exploration des schémas, décrite dans le chapitre 5. Cette exploration consiste à analyser tous les schémas et toutes les combinaisons de paramètres d'entrée pour une application donnée. Les résultats retournés correspondent à une sélection du meilleur schéma, ainsi que du jeu de paramètres d'entrée le plus optimal possible. L'utilisateur aura au préalable choisi s'il souhaite minimiser le temps d'exécution, la consommation mémoire, ou obtenir un compromis entre ces deux facteurs. Par ailleurs, le chapitre 5 présente l'utilisation de PAnTHErS par :

- un concepteur désirant utiliser du chiffrement homomorphe pour son application, sans compétences préalables dans le domaine,
- un expert en chiffrement homomorphe voulant analyser les performances de son schéma et/ou désireux de l'optimiser,
- un développeur expert souhaitant intégrer de nouvelles fonctionnalités d'analyses pour un schéma de chiffrement homomorphe.

6.1.3 Résultats des contributions

Les analyses théoriques des schémas mettent en avant l'influence des paramètres d'entrée sur un schéma. En l'occurrence, pour les schémas FV [FV12], YASHE [BLLN13] et F-NTRU [DS16], les résultats d'analyses théoriques présentés dans la section 3.2.6 montrent qu'une augmentation du module q induit une plus grande complexité algorithmique et consommation mémoire plus importante. En revanche, plus w est grand et plus la complexité algorithmique et le coût mémoire diminuent.

La calibration a permis une première comparaison entre plusieurs schémas. En effet, les résultats d'analyse théoriques montrent que YASHE présente une complexité algorithmique "pire cas" plus importante que FV. Or, après calibration de ces résultats théoriques, ceux-ci deviennent plus représentatifs de résultats qui seraient obtenus lors d'une exécution concrète. Les résultats théoriques calibrés permettent ainsi de déterminer que FV s'exécute plus rapidement que YASHE (Figure 4.3). La section 4.3 montre que 99 % des analyses théoriques calibrées pour la complexité algorithmique (resp. 95 % des analyses théoriques calibrées pour le coût mémoire) affichent un taux d'erreur inférieur à 20 % (resp. 20.1 %). De plus, la moyenne des analyses théoriques calibrées ne dépassent pas les 10 % d'erreur. Ces résultats valident les exigences fixées pour la p-calibration dans la section 4.2.2.

Les schémas analysés d'abord de façon théorique, puis calibrés, ont ensuite chacun été explorés par la méthode d'exploration automatisée présentées dans les sections 5.1.3 et 5.2. Plus de 220500 jeux de paramètres ont ainsi été explorés afin de retourner 14 jeux de paramètres pour les trois schémas cumulés pour l'application *cinqHB* et 5 jeux de paramètres en tout pour les applications *Pédagogique*, *Croissant* et *Médicale*. De plus, tous les jeux de paramètres à explorer ont été exécutés concrètement afin d'évaluer la pertinence des résultats obtenus via notre exploration. Les résultats issus de ces exécutions concrètes ont donc été classés en utilisant la technique du front de Pareto, qui permet de favoriser en priorité les résultats affichant un temps d'exécution et un coût mémoire plus faibles (voir section 5.1.3). Pour l'application *cinqHB*, les 14 jeux de paramètres proposés par l'exploration se situent ainsi dans les 15 premiers centiles du classement, 10 d'entre eux sont situés dans les cinq premiers centiles. Concernant les applications *Pédagogique*, *Croissant* et *Médicale*, 4 jeux de paramètres sur 5 sont positionnées dans les six premiers centiles et 1 dans les 30 premiers centiles. Ce dernier jeu de paramètre est éloigné des points optimaux. D'après les résultats des quatre applications, les exigences, exprimées lors de la définition de l'exploration, ne sont pas entièrement remplies, notamment pour l'application *Croissant* avec le schéma F-NTRU et pour l'application *cinqHB* avec le schéma YASHE. Une perspective d'amélioration (décrite dans la section suivante et décrite également dans la sous-section 5.1.3) est facilement envisageable afin de garantir les exigences initialement posées et une sélection de jeux de paramètre plus proche de l'optimal *c-à-d* situés dans les cinq premiers centiles du classement.

6.2 Perspectives

L'outil PAnTHErS, présenté dans ce manuscrit, a été publié en libre accès sur la plateforme GitHub [Fer18a] sous la licence CeCILL 2.1 [CEA]. La documentation de l'outil est également disponible sur HAL et sur GitHub [Fer18a] : celle-ci permet, à différents usagers, d'exploiter les fonctionnalités de PAnTHErS, allant d'une simple utilisation de l'interface actuelle à l'ajout de schémas et d'applications. De plus, la documentation fournit également la procédure d'installation de l'outil.

Toutefois, plusieurs perspectives d'évolution de PAnTHErS sont réalisables à court et à moyen terme. Ces perspectives sont principalement axées sur PAnTHErS et le domaine de l'homomorphe.

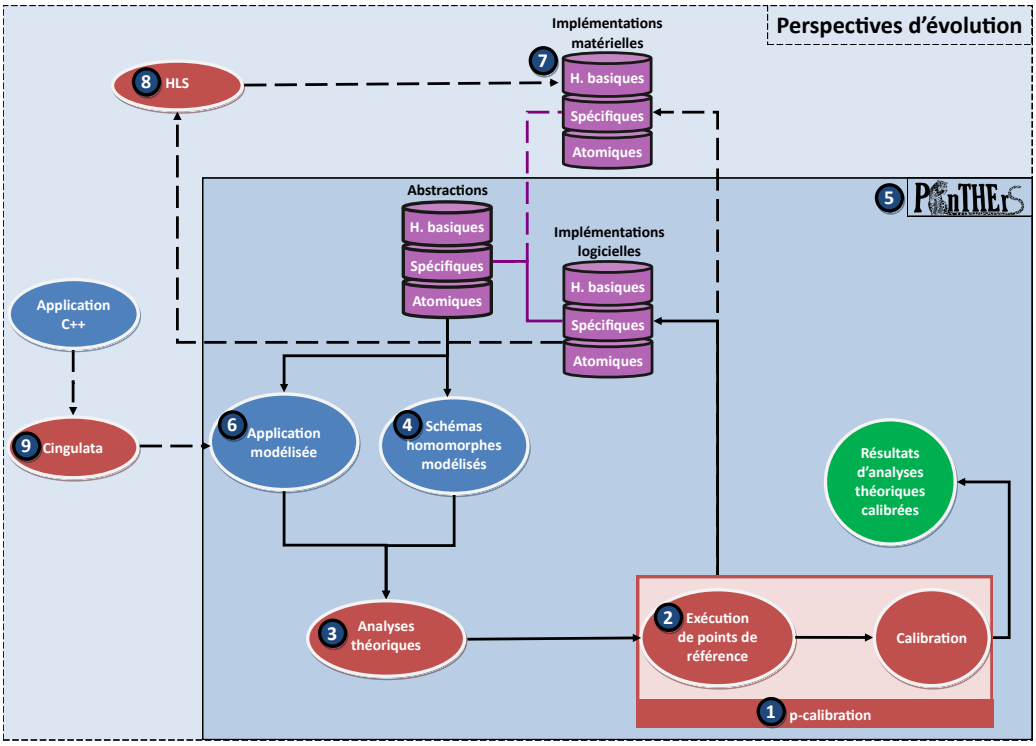


FIGURE 6.1 – Perspectives d'évolution pour l'outil PAnTHErS.

La Figure 6.1 représente le flot de traitements de l'outil PAnTHErS incluant les perspectives d'évolution listées dans cette section. Chaque numéro placé sur la figure renvoie à une ou plusieurs améliorations possibles pour l'étape de traitement référencée.

À court terme, des améliorations et des fonctionnalités peuvent être ajoutées dans PAnTHErS :

Sur les méthodes disponibles dans PAnTHErS :

Deux premières évolutions, simples techniquement mais à valeur ajoutées, sont :

• Actuellement, l'outil PAnTHErS ne permet d'effectuer que des 2-calibrations. Une amélioration possible serait d'ajouter une fonctionnalité permettant de réaliser des p-calibrations alternatives avec $p > 2$. De plus, cette fonctionnalité pourrait permettre de choisir précisément les valeurs pour les points de référence à exécuter concrètement avant la calibration. Cette amélioration est illustrée en (1) de la Figure 6.1.

• L'exploration se base sur un nombre de points de référence restreint (4 pour F-NTRU et 8 pour FV et YASHE). Une évolution de l'exploration serait de permettre à un utilisateur de choisir le nombre de points de référence à exécuter concrètement en fonction de la taille et de la configuration de l'ensemble de jeux de paramètres à évaluer. L'exploration pourrait alors bénéficier des avantages de la p-calibration alternative avec $p > 2$ ((2) de la Figure 6.1).

Une troisième évolution, plus significative, est l'ajout d'une nouvelle métrique dans PAnTHErS.

• Durant ce doctorat, un stagiaire (étudiant en master 2 de cryptographie) a réalisé une étude de l'évolution du bruit. Cette étude appliquée à un schéma permet d'obtenir deux résultats : tout d'abord, l'évolution du bruit produit par les additions et multiplications homomorphes (sans calcul mathématique préalable) et ensuite, à partir du jeu de paramètres d'entrée, déduire la profondeur multiplicative engendrée. L'étude de bruit sera intégrée prochainement dans l'outil PAnTHErS en tant que nouveau module d'analyse de schémas. L'exploration pourra bénéficier de cette nouvelle étude pour pouvoir déterminer automatiquement tous les jeux de paramètres impliquant une profondeur multiplicative d fixée (3 de la Figure 6.1).

Sur la facilité d'utilisation de PAnTHErS :

• Un expert souhaitant intégrer un schéma (resp. Spécifique ou application) modélisé doit créer "à la main" son équivalent en fonctions d'analyse. Cette étape pourrait être automatisée afin de faciliter l'utilisation de l'outil PAnTHErS (4 de la Figure 6.1).

Sur la modularité de PAnTHErS :

• La calibration est effectuée en se basant sur des exécutions concrètes de schémas codées en Python. Une piste d'amélioration serait de tester la calibration en prenant comme référence des exécutions concrètes, codées dans d'autres langages de programmation. Pour cela, les programmes pris en référence doivent être tout d'abord modélisés le plus fidèlement possible dans PAnTHErS. Cette extension, illustrée en 2 de la Figure 6.1, permettrait de pouvoir analyser, voire comparer, les bibliothèques de la littérature (*p. ex.* SEAL [LCP17], FV-NFLlib [Cry16] et HElib [HS14a]).

Des perspectives d'améliorations sont également réalisables à moyen terme :

Sur la fiabilité et l'efficacité de PAnTHErS :

• Dans le chapitre 4, nous avons remarqué que le choix des points de référence pour la p-calibration a un impact sur la qualité de la calibration. Une étude pourrait être réalisée afin de déterminer une sélection optimale de ces points *c-à-d* comment bien choisir les points de référence dans le but de réduire au maximum les taux d'erreur (2 de la Figure 6.1). Par exemple, des études complémentaires pourraient être effectuées sur les résultats théoriques afin de déterminer de nouveaux points de référence. Ces points pourraient correspondre à des zones d'intérêt où le risque d'écart d'estimation est élevé (zones de fortes variations, points d'inflexion, minimum, maximum,...).

• PAnTHErS est entièrement implémenté en Python sans optimisation quelconque. PAnTHErS pourrait être optimisé (*p. ex.* en effectuant de la programmation parallèle) ou implémenté dans un autre langage (*p. ex.* C++) qui permettrait d'exécuter plus rapidement les analyses (5 de la Figure 6.1).

Sur l'extension de PAnTHErS :

• De nouvelles analyses (comme *p. ex.* la gestion du bruit) peuvent toujours être intégrées dans PAnTHErS. L'ajout d'un nouvelle analyse doit permettre d'obtenir des

informations utiles sur un algorithme (qu'il soit homomorphe ou non). Cette extension intervient en (3) de la Figure 6.1.

• Par ailleurs, d'autres types d'algorithmes pourraient être intégrés dans PAnTHErS afin d'explorer ses capacités d'adaptation. Entre autres, ces algorithmes pourraient provenir du domaine de la cryptographie, ou encore des mathématiques, ou même d'un autre domaine. La seule contrainte, afin de pouvoir réaliser les analyses, est que l'algorithme ajouté ne possède pas de boucle non bornée et non déterministe. Ces nouveaux algorithmes seraient intégrés en (6) de la Figure 6.1.

• Un frein à l'utilisation est bien entendu la complexité des traitements donc le temps d'exécution. Par conséquent, l'exécution des schémas et applications pourrait bénéficier d'une accélération matérielle.

D'une part, l'outil PAnTHErS pourrait être enrichi en intégrant des travaux existants qui visent à accélérer l'exécution de schémas de chiffrement homomorphe en s'appuyant, entre autres, sur des composants matériels ((7) de la Figure 6.1). De plus, les analyses devront prendre en considération ces facteurs d'accélération pour effectuer au mieux la phase de calibration des résultats théoriques ((1) et (3) de la Figure 6.1). Par exemple, PAnTHErS pourrait bénéficier des travaux de Migliore et al. [MRL+18], qui visent à accélérer les multiplications polynomiales en se basant sur une approche mêlant traitements logiciel et matériel.

D'autre part, les algorithmes Atomiques et Spécifiques de la bibliothèque de l'outil PAnTHErS pourraient être automatiquement convertis vers des implémentations matérielles équivalentes à travers des outils de HLS [1] ((8) de la Figure 6.1).

• Un interfaçage avec l'outil Cingulata [Tea17] permettrait d'intégrer directement dans PAnTHErS une application. D'abord codée en C++, celle-ci serait automatiquement convertie au format BLIF, puis en enchaînement d'opérations logiques (XOR, OR, AND et NOT). Enfin, l'application serait par la suite automatiquement intégrée à l'outil PAnTHErS ((9) de la Figure 6.1).

Sur l'interface graphique de PAnTHErS :

Bien que moins prioritaires, plusieurs améliorations sont envisageables sur l'interface graphique ((5) de la Figure 6.1). Les plus notables sont les suivantes :

• Certaines fonctionnalités sont disponibles dans PAnTHErS (*p. ex.* l'analyse détaillée des schémas au niveau des algorithmes atomiques, ou profilage) mais non accessible depuis l'interface graphique. L'interface graphique devrait pouvoir offrir un accès direct à toutes les fonctionnalités disponibles dans PAnTHErS.

• L'interface graphique pourrait aussi proposer l'ajout et la création de nouveaux schémas de façon graphique. De cette manière, cela éviterait à un utilisateur de devoir éditer le code source de PAnTHErS afin d'y ajouter son schéma ou son application.

Comme nous l'avons présenté dans ce manuscrit, l'outil PAnTHErS est d'ores et déjà exploitable et permet un gain de productivité pour plusieurs profils d'usagers. Ces usagers ont à leur disposition une documentation détaillée, qui nous l'espérons, permettra de fédérer une communauté en facilitant l'accès à l'outil. Cette situation est satisfaisante mais ne dispense pas de prévoir un certain nombre d'évolutions qui ont été reprises dans la section précédente. Ces évolutions vont de l'ajout de boutons et saisie de paramètres dans l'interface graphique à des changements plus impactant tels que l'ajout de schéma, ou encore, l'accélération de traitement.

1. High Level Synthesis, en anglais. Synthèse de haut niveau, en français.

Les travaux présentés dans cette thèse amènent donc à de nombreuses poursuites potentielles. Afin de les réaliser, des ressources humaines sont nécessaires. La publication des sources en libre accès, telle que nous l'avons réalisée via GitHub, va permettre la mobilisation d'une communauté de développeurs afin de continuer à faire évoluer l'outil PAnTHErS.

Liste de publications

Conférences internationales

- Cyrielle Feron, Vianney Lapotre et Loïc Lagadec, *PAnTHErS : A Prototyping and Analysis Tool for Homomorphic Encryption Schemes*, in Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4 : SECRYPT, Madrid, Spain, pages 359–366, 2017, [FLL17].
- Cyrielle Feron, Vianney Lapotre et Loïc Lagadec, *Fast Evaluation of Homomorphic Encryption Schemes based on Ring-LWE*, in New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on, pages 1–5, IEEE, 2018, [FLL18].

Communications orales dans des séminaires

- Cyrielle Feron, Loïc Lagadec et Vianney Lapotre, *Toward Automated Analysis and Prototyping of Homomorphic Encryption Schemes*, Journées "Codes et Cryptographie", 2017.
- Cyrielle Feron, Vianney Lapotre et Loïc Lagadec, *Parameter Exploration for Homomorphic Encryption Schemes Analysis*, CryptArchi 2018.

Communications par poster

- Cyrielle Feron, Loïc Lagadec et Vianney Lapotre, *A Homomorphic Encryption Analysis and Prototyping Tool*, GDR SoC-SiP, 2016.
- Cyrielle Feron, Loïc Lagadec et Vianney Lapotre, *PAnTHErS : From Homomorphic Encryption Experts to Designers*, Séminaire MOCS, 2018.

Code source et tutoriels

- Logiciel PAnTHErS (code source et tutoriels) diffusé sous la licence CeCILL 2.1 [CEA], disponible en libre téléchargement sur : `https://github.com/cferon/PAnTHErS` [Fer18a].
- Guide d'utilisateur de PAnTHErS publié sur HAL [Fer18b].

Glossaire

AES	Advanced Encryption Standard
A-GCD	Approximate Greatest Common Divisor
BLIF	Berkeley Logic Interchange Format
CBC	Cipher Block Chaining
CEA	Commissariat à l'Énergie atomique et aux Énergies alternatives
CSA	Consumer, Science & Analytics
CSV	Comma-Separated Values
CVP	Closest Vector Problem
DDR	Double Data Rate
ECB	Electronic Code Book
EJBCA	Enterprise JavaBeans Certificate Authority
FFT	Fast Fourier Transform
FHE	Fully Homomorphic Encryption
F-NTRU	Flattening NTRU
FPGA	Field-Programmable Gate Array
FV	Fan-Vercauteren
HLS	High Level Synthesis
HTTPS	HyperText Transfer Protocol Secure
LWE	Learning With Errors
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
NP	Nondeterministic Polynomial time
NTL	Number Theory Library
NTRU	Nth Degree Truncated Polynomial Ring Units
OTP	One Time Pad
PAnTHErS	Prototyping and Analysis Tool for Homomorphic Encryption Schemes
PDF	Portable Document Format
PKI	Public Key Infrastructure
PRNG	Pseudo-Random Number Generator
RLE	Run Length Encoding
R-LWE	Ring Learning With Errors
RNS	Residue Number System
RSA	Rivest, Shamir et Adleman
SEAL	Simple Encrypted Arithmetic Library
SHA	Secure Hash Algorithm
SHIELD	Scalable Homomorphic Implementation of Encrypted Data-Classifiers
SIMD	Single Instruction Multiple Data
SSL	Secure Sockets Layer

SVP Shortest Vector Problem
TLS Transport Layer Security
YASHE Yet Another Somewhat Homomorphic Encryption

Bibliographie

[ABD16] Martin Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched ntru assumptions. In *Annual Cryptology Conference*, pages 153–178. Springer, 2016.

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Jeffrey Hoffstein, Kristin Lauter, Satya Lokam, Daniele Micciancio, et al. Homomorphic encryption standard. 2018.

[Ajt98] Miklós Ajtai. The shortest vector problem in l 2 is np-hard for randomized reductions. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 10–19. ACM, 1998.

[Alb17a] Martin Albrecht. LWE Estimator : Sage module for estimating the concrete security of LWE instances. https://bitbucket.org/malb/lwe-estimator/, 2017.

[Alb17b] Martin Albrecht. LWE Generator : Sage module for generating LWE instances. https://bitbucket.org/malb/lwe-generator/, 2017.

[Alb17c] Martin R Albrecht. On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 103–129. Springer, 2017.

[AMBG+16] Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrede Lepoint. Nfllib : Ntt-based fast lattice library. In *Cryptographers' Track at the RSA Conference*, pages 341–356. Springer, 2016.

[AMGH10] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *Annual Cryptology Conference*, pages 138–154. Springer, 2010.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3) :169–203, 2015.

[ASP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *International Cryptology Conference*, pages 297–314. Springer, 2014.

[BEHZ16] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. *IACR Cryptology ePrint Archive*, 2016 :510, 2016.

[Ber92] University of California Berkeley. Berkeley logic interchange format (blif). *Oct Tools Distribution*, 2 :197–247, 1992.

[BF17] Guillaume Bonnoron and Caroline Fontaine. A note on ring-lwe security in the case of fully homomorphic encryption. In *International Conference in Cryptology in India*, pages 27–43. Springer, 2017.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography Conference*, pages 325–341. Springer, 2005.

[BGV12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *ITCS*, 2012.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4) :506–519, 2003.

[BLLN13] J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *Cryptography and Coding IMA*, 2013.

[BR15] Jean-François Biasse and Luis Ruiz. Fhew with efficient multibit bootstrapping. In *International Conference on Cryptology and Information Security in Latin America*, pages 119–135. Springer, 2015.

[Bra12] Z. Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Proc. CRYPTO*, pages 868–886, Santa Barbara, CA, USA, 2012.

[Buk12] Tim Buktu. NTRUEncrypt public key encryption algorithm in Java and C. http://tbuktu.github.io/ntru/, 2012.

[BV11a] Z. Brakerski and V. Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. *FOCS 2011*, pages 97–106, 2011.

[BV11b] Z. Brakerski and V. Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Proc. CRYPTO*, pages 505–524, Santa Barbara, CA, USA, 2011.

[BV14] Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In *Proc. Innovations in Theoretical Computer Science*, pages 1–12, Princeton, NJ, USA, 2014.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers : A practical solution for efficient homomorphic-ciphertext compression. In *International Conference on Fast Software Encryption*, pages 313–333. Springer, 2016.

[CCK+13] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. Batch Fully Homomorphic Encryption over the Integers. In *Proc. EUROCRYPT*, pages 315–335, Athens, Greece, 2013.

[CCKS17] Sébastien Canard, Sergiu Carpov, Donald Nokam Kuate, and Renaud Sirdey. Running compression algorithms in the encrypted domain : a case-study on the homomorphic execution of rle. *IACR Cryptology ePrint Archive*, 2017 :392, 2017.

[CCSV17] Joël Cathébras, Alexandre Carbon, Renaud Sirdey, and Nicolas Ventroux. An analysis of fv parameters impact towards its hardware acceleration. In *International Conference on Financial Cryptography and Data Security*, pages 91–106. Springer, 2017.

[CDS15] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo : a compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 13–19. ACM, 2015.

[CEA] Inria CEA, CNRS. Licence cecill.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption : Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 3–33, 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe : Fast fully homomorphic encryption library over the torus. https://tfhe.github.io/tfhe/, 2017.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[CLT14] J.-S. Coron, T. Lepoint, and M. Tibouchi. Scale-Invariant Fully Homomorphic Encryption over the Integers. In *Proc. Public-Key Cryptography - PKC*, pages 311–328, Buenos Aires, Argentina, 2014.

[CMNT11] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully Homomorphic Encryption over the Integers with Shorter Public Keys. In *Proc. CRYPTO*, pages 487–504, Santa Barbara, CA, USA, 2011.

[CNS+16] Sergiu Carpov, Thanh Hai Nguyen, Renaud Sirdey, Gianpiero Constantino, and Fabio Martinelli. Practical privacy-preserving medical diagnosis using homomorphic encryption. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 593–599. IEEE, 2016.

[CNT12] J.-S. Coron, D. Naccache, and M. Tibouchi. Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. In *Proc. EUROCRYPT*, pages 446–464, Cambridge, UK, 2012.

[Cry16] CryptoExperts. FV-NFL : Library implementing the Fan-Vercauteren homomorphic encryption scheme. https://github.com/CryptoExperts/FV-NFLlib#fv-nfllib, 2016.

[CS15] Sergiu Carpov and Renaud Sirdey. A compression method for homomorphic ciphertexts. *IACR Cryptology ePrint Archive*, 2015 :1199, 2015.

[DCP08] Christophe De Canniere and Bart Preneel. Trivium. In *New Stream Cipher Designs*, pages 244–266. Springer, 2008.

[DG14] Nagarjun C Dwarakanath and Steven D Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3) :159–180, 2014.

[DGHV10] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In *Proc. EUROCRYPT*, pages 24–43, French Riviera, 2010.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6) :644–654, 1976.

[DHP+18] Yarkın Doröz, Jeffrey Hoffstein, Jill Pipher, Joseph H Silverman, Berk Sunar, William Whyte, and Zhenfei Zhang. Fully homomorphic encryption from the

finite field isomorphism problem. In *IACR International Workshop on Public Key Cryptography*, pages 125–155. Springer, 2018.

[Die08] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[DM15] Léo Ducas and Daniele Micciancio. Fhew : bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.

[DPT] Alastair Denniston, Bletchley Park, and Alan Turing. Enigma machine.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal : Rijndael. 1999.

[DS16] Yarkin Doröz and Berk Sunar. Flattening NTRU for evaluation key free homomorphic encryption. *IACR Cryptology ePrint Archive*, 2016.

[EMST78] William F Ehrsam, Carl HW Meyer, John L Smith, and Walter L Tuchman. Message verification and transmission error detection by block chaining, February 14 1978. US Patent 4,074,066.

[Fer18a] Cyrielle Feron. Logiciel PAnTHErS (Prototyping and Analysis Tool for Homomorphic Encryption Schemes). 2018.

[Fer18b] Cyrielle Feron. PAnTHErS : User Guide. Technical report, ENSTA Bretagne ; Lab-STICC, September 2018.

[FIP12] NIST FIPS. 180-4 : Secure hash standard (shs). *US Department of Commerce, National Institute of Standards and Technology (NIST)*, 2012.

[FIP13] NIST FIPS. 186-4 : Digital signature standard (dss). *US Department of Commerce, National Institute of Standards and Technology (NIST)*, 2013.

[FKK11] Alan Freier, Philip Karlton, and Paul Kocher. The secure sockets layer (ssl) protocol version 3.0. 2011.

[FLL17] Cyrielle Feron, Vianney Lapotre, and Loïc Lagadec. PAnTHErS : A Prototyping and Analysis Tool for Homomorphic Encryption Schemes. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4 : SECRYPT, Madrid, Spain*, pages 359–366, 2017.

[FLL18] Cyrielle Feron, Vianney Lapotre, and Loic Lagadec. Fast evaluation of homomorphic encryption schemes based on ring-lwe. In *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*, pages 1–5. IEEE, 2018.

[FV12] J. Fan and F. Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012 :144, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GH11] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–148. Springer, 2011.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P Smart. Better bootstrapping in fully homomorphic encryption. In *International Workshop on Public Key Cryptography*, pages 1–16. Springer, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the AES circuit. In *Advances in cryptology–crypto 2012*, pages 850–867. Springer, 2012.

[GSW13] C. Gentry, A. Sahai, and B. Waters. Homomorphic Encryption from Learning with Errors : Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Proc. CRYPTO*, pages 75–92, Santa Barbara, CA, USA, 2013.

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In *Cryptography and Lattices*, pages 51–66. Springer, 2001.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU : A ring-based public key cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, pages 267–288, 1998.

[HS14a] S. Halevi and V. Shoup. HElib - An implementation of homomorphic encryption. https://github.com/shaih/HElib, 2014.

[HS14b] Shai Halevi and Victor Shoup. Algorithms in helib. In *International cryptology conference*, pages 554–571. Springer, 2014.

[HS14c] Shai Halevi and Victor Shoup. Bootstrapping for helib. Cryptology ePrint Archive, Report 2014/873, 2014. https://eprint.iacr.org/2014/873.

[KCB97] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. Hmac : Keyed-hashing for message authentication. 1997.

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched ntru parameters. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–26. Springer, 2017.

[KGV16] Alhassan Khedr, P. Glenn Gulak, and Vinod Vaikuntanathan. SHIELD : Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Trans. Computers*, 2016.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of multi-digit numbers on automata. In *Doklady Akad. Nauk SSSR*, pages 595–596. Translation in Soviet Physics-Doklady, 44(7), 1963.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[LCP17] Kim Laine, Hao Chen, and Rachel Player. Simple encrypted arithmetic library - SEAL (v2.1). https://sealcrypto.codeplex.com/, 2017.

[LN14] T. Lepoint and M. Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *AFRICACRYPT*, 2014.

[LP11] R. Lindner and C. Peikert. Better Key Sizes (and Attacks) for LWE-Based Encryption. In *Proc. - CT-RSA 2011*, pages 319–339, San Francisco, CA, USA, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[LPVG03] Loïc Lagadec, Bernard Pottier, and Oscar Villellas-Guillen. An lut-based high level synthesis framework for reconfigurable architectures. *Domain-Specific Processors : Systems, Architectures, Modeling, and Simulation*, pages 19–39, 2003.

[Mar02] Robert C Martin. *Agile software development : principles, patterns, and practices*. Prentice Hall, 2002.

[MBF17] Vincent Migliore, Guillaume Bonnoron, and Caroline Fontaine. Determination and exploration of practical parameters for the latest Somewhat Homomorphic Encryption (SHE) schemes. *Working paper or preprint*, 2017.

[MRL+18] Vincent Migliore, Maria Mendez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, and Guy Gogniat. Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using karatsuba algorithm. *IEEE Trans. Computers*, 67(3) :335–347, 2018.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical ? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[Par64] Vilfredo Pareto. *Cours d'économie politique*, volume 1. Librairie Droz, 1964.

[Ped17] Fabian Pedregosa. Memory_profile, module python (version 0.45), 2017. https://pypi.org/project/memory_profiler/.

[Pol71] John M Pollard. The fast fourier transform in a finite field. *Mathematics of computation*, 25(114) :365–374, 1971.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11) :169–180, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :120–126, 1978.

[RVVV17] Sujoy Sinha Roy, Frederik Vercauteren, Jo Vliegen, and Ingrid Verbauwhede. Hardware assisted fully homomorphic function evaluation and encrypted search. *IEEE Transactions on Computers*, 66(9) :1562–1572, 2017.

[SAA16] William A Stein, T Abbott, and M Abshoff. SageMath. http://www.sagemath.org/, 2016.

[Shi13] John W Shipman. Tkinter 8.4 reference : a gui for python. *New Mexico Tech Computer Center*, 2013.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2) :303–332, 1999.

[Sho17] Viktor Shoup. NTL library : A Library for doing Number Theory. http://www.shoup.net/ntl/, 2017.

[Sol] PrimeKey Solutions. Ejbca (enterprise javabeans certificate authority) - the open source ca.

[SS11] Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 27–47. Springer, 2011.

[SV10] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[SV14] Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, codes and cryptography*, 71(1) :57–81, 2014.

[Tea17] CEA-LIST Crypto Team. Cingulata : open-source compiler toolchain. https://github.com/CEA-LIST/Cingulata/, commit fbc40d9e9948630047e4a60312c925594d14fc46, 2017.

[TKW18] Louis Tajan, Moritz Kaumanns, and Dirk Westhoff. Pre-computing appropriate parameters : How to accelerate somewhat homomorphic encryption for cloud auditing. In *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*, pages 1–6. IEEE, 2018.

[Too63] Andrei L Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. In *Soviet Mathematics Doklady*, volume 3, pages 714–716, 1963.

[Tuk77] John W. Tukey. *Exploratory data analysis*. Addison-Wesley series in behavioral science : quantitative methods. Addison-Wesley, 1977.

[Tur40] AM Turing. Mathematical theory of enigma machine. *Public Record Office, London*, 3, 1940.

Annexe A

Résultats de l'exploration des applications *Pédagogique*, *Croissant* et *Médicale*

Cette annexe fournit tous les détails des résultats pour les applications *Pédagogique*, *Croissant* et *Médicale*.

A.1 Application *Pédagogique*

La Table A.1 renseigne le nombre de jeux de paramètres à explorer ainsi que, pour chaque schéma :

- le temps nécessaire pour déterminer l'ensemble des jeux de paramètres à explorer,
- la durée de l'exploration pour ces jeux de paramètres,
- la durée effective pour l'exécution concrète de l'ensemble des jeux de paramètres pour l'application *Pégagogique*.

La Table A.1 donne également le facteur d'accélération entre le temps requis pour effectuer l'exploration et la durée requise pour réaliser l'ensemble des exécutions concrètes.

La Figure A.1 illustre la répartition des taux d'erreur visibles entre les analyses théoriques calibrées et les temps d'exécutions et coûts mémoire concrets obtenus après exécution.

La Table A.2 liste les résultats retournées par l'exploration pour l'application *Pédagogique*.

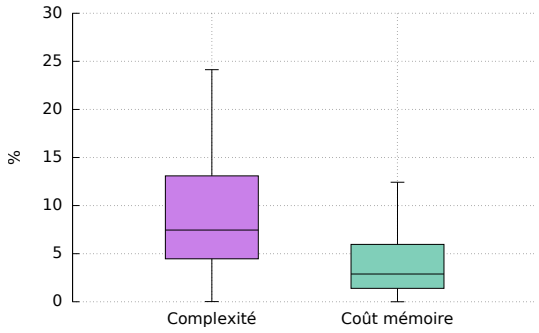
La Table A.3 liste les résultats présents sur le front de Pareto des exécutions concrètes de l'application *Pédagogique*.

Dans la Figure A.2, représentant la répartition des jeux de paramètres explorés, les croix bleues représentent les résultats d'analyse concrète obtenus suite à l'exécution des jeux de paramètres. Les carrés rouges sont les points optimaux (*c-à-d* identifiés sur le front de Pareto) des exécutions (croix bleues). Les ronds verts représentent les résultats sélectionnés par l'exploration de l'application pour un schéma fixé.

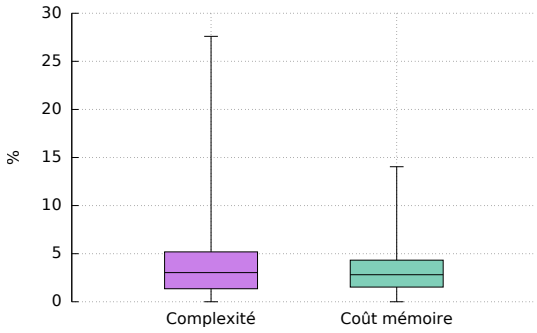
La Table A.4 donne un classement (position absolue et en pourcentage) des résultats optimaux des exécutions concrètes de l'application *Pédagogique*.

TABLE A.1 – Informations sur l'exploration réalisée pour chaque schéma pour l'application *Pédagogique*.

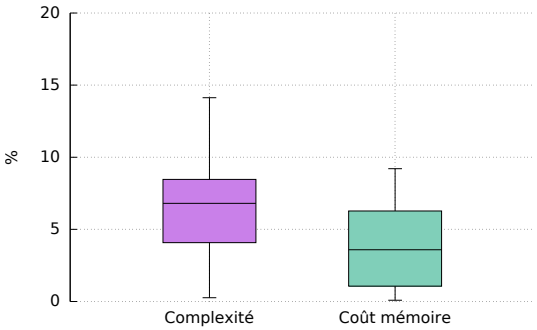
Schéma	Nombre de jeux	Temps pour trouver les jeux	Durée d'exploration des paramètres	Temps total des exécutions	Accélération
FV	15565	0 h 27 min 27 sec	4 h 13 min 56 sec	84 h 1 min 23 sec	19.9
YASHE	7342	2 h 21 min 58 sec	14 h 36 min 6 sec	890 h 40 min 2 sec	61.0
F-NTRU	62	0 h 0 min 12 sec	5 h 39 min 34 sec	49 h 49 min 53 sec	8.8
TOTAL *Pédagogique*	22969	2 h 49 min 37 sec	24 h 29 min 36 sec	1024 h 31 min 17 sec	41.8



(a) FV (99 centiles).



(b) YASHE (99 centiles).



(c) F-NTRU (99 centiles).

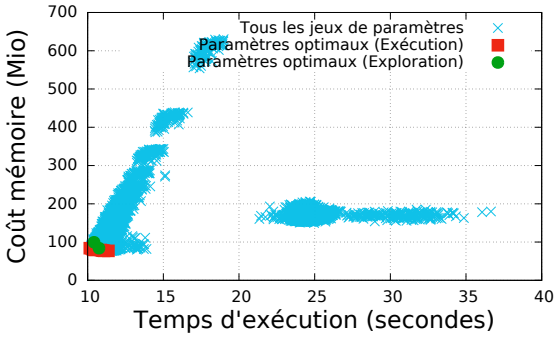
FIGURE A.1 – Boîte de Tukey des taux d'erreur obtenus pour les résultats de l'exploration comparés aux résultats des exécutions concrètes, pour chaque schéma (avec l'application *Pédagogique*) et chaque analyse théorique calibrée (complexité et coût mémoire).

TABLE A.2 – Résultats retournés par la méthode d'exploration de l'application *Pédagogique* pour les schémas FV, YASHE et F-NTRU.

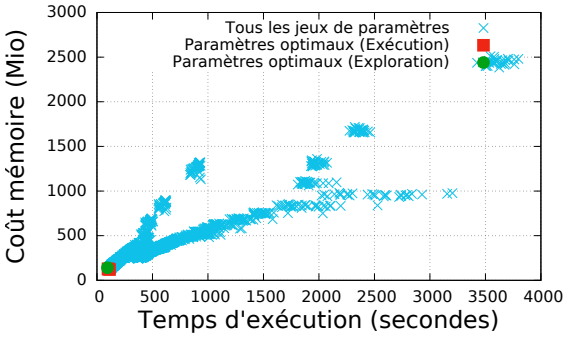
Schéma	$\log(q)$	$\log(w)$	t	Temps (sec)			Coût mémoire (Mio)		
				Estim.	Exéc.	Erreur	Estim.	Exéc.	Erreur
FV	297	23	2	10.4223	10.2558	1.6235 %	99.3713	93.8	5.9395 %
	339	68	2	10.7477	11.5291	6.7776 %	83.9337	80.2	4.6555 %
YASHE	345	50	2	94.2215	102.3478	7.9399 %	140.7406	128.3	9.6965 %
F-NTRU	146	7	–	1112.1230	1124.7167	1.1197 %	230.9965	228.5	1.0926 %

TABLE A.3 – Résultats retournés après l'exécution complète de tous les jeux de paramètres de l'application *Pégagogique* pour les schémas FV, YASHE et F-NTRU. En rouge, les jeux de paramètres présents également dans la Table A.2.

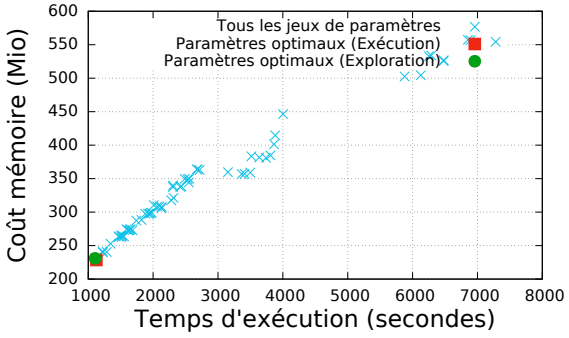
Schéma	$\log(q)$	$\log(w)$	t	**Temps (sec)**	**Coût mémoire (Mio)**
FV	307	35	2	10.3194	83.6
	310	35	2	10.1390	84.1
	312	40	2	10.3946	79.9
	313	38	2	10.2580	83.6
	314	40	2	10.3467	81.1
	315	40	2	10.3775	80.3
	315	43	2	10.3656	80.6
	320	46	2	10.5102	79.4
	328	56	2	11.3746	77
	331	57	2	10.8673	79.3
	337	57	3	10.9461	78.3
	338	63	2	11.0836	77
	342	70	2	11.0078	77.4
YASHE	328	33	2	113.8491	121.3
	348	51	2	100.3905	127.1
	348	52	2	101.3121	126.8
F-NTRU	146	7	–	1124.7167	228.5



(a) FV.



(b) YASHE.



(c) F-NTRU.

FIGURE A.2 – Identification des points retournés par l'exploration pour l'application *Pédagogique*.

TABLE A.4 – Classement des points sélectionnés par l'exploration de l'application *Pégagogique* et écart de ces points avec le résultat optimal obtenu par exécution concrète.

Schéma	$\log(q)$	$\log(w)$	t	**Classement**		**Écart avec le point optimal**	
				Position	**En %**	**Temps (sec)**	**Coût mémoire (Mio)**
FV	297	23	2	119/15565	0.7581	0.1168	16.8
	339	68	2	736/15565	4.7221	1.3902	3.2
YASHE	345	50	2	8/7342	0.0953	1.9573	7.0
F-NTRU	146	7	–	1/62	0	0.0	0.0

A.2 Application *Croissant*

La Table A.5 renseigne le nombre de jeux de paramètres à explorer ainsi que, pour chaque schéma :

- le temps nécessaire pour déterminer l'ensemble des jeux de paramètres à explorer,
- la durée de l'exploration pour ces jeux de paramètres,
- la durée effective pour l'exécution concrète de l'ensemble des jeux de paramètres pour l'application *Croissant*.

La Table A.5 donne également le facteur d'accélération entre le temps requis pour effectuer l'exploration et la durée requise pour réaliser l'ensemble des exécutions concrètes.

La Figure A.3 illustre la répartition des taux d'erreur visibles entre les analyses théoriques calibrées et les temps d'exécutions et coûts mémoire concrets obtenus après exécution.

La Table A.6 liste les résultats retournées par l'exploration de l'application *Croissant*.

La Table A.7 liste les résultats présents sur le front de Pareto des exécutions concrètes de l'application *Croissant*.

Dans la Figure A.4, représentant la répartition des jeux de paramètres explorés, les croix bleues représentent les résultats d'analyse concrète obtenus suite à l'exécution des jeux de paramètres. Les carrés rouges sont les points optimaux (*c-à-d* identifiés sur le front de Pareto) des exécutions (croix bleues). Les ronds verts représentent les résultats sélectionnés par l'exploration de l'application pour un schéma fixé.

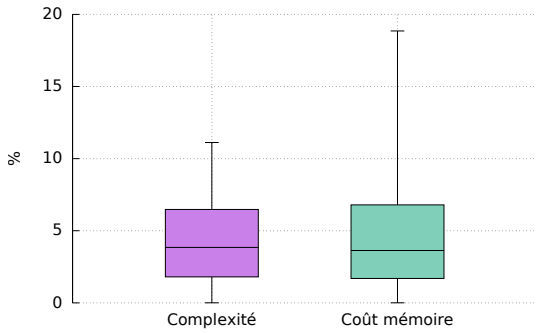
La Table A.8 donne un classement (position absolue et en pourcentage) des résultats optimaux des exécutions concrètes de l'application *Croissant*.

TABLE A.5 – Informations sur l'exploration réalisée pour chaque schéma pour l'application *Croissant*.

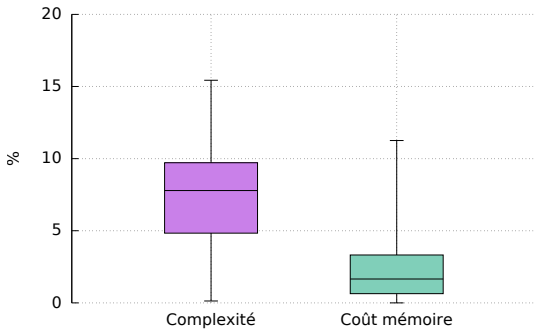
Schéma	Nombre de jeux	Temps pour trouver les jeux	Durée d'exploration des paramètres	Temps total des exécutions	Accélération
FV	2665	0 h 17 min 54 sec	0 h 35 min 2 sec	9 h 36 min 6 sec	16.4
YASHE	2760	2 h 9 min 14 sec	1 h 24 min 42 sec	103 h 54 min 26 sec	73.6
F-NTRU	259	0 h 0 min 33 sec	27 h 53 min 33 sec	175 h 45 min 30 sec	6.3
TOTAL *Croissant*	5684	2 h 27 min 41 sec	29 h 53 min 17 sec	289 h 16 min 2 sec	9.7

TABLE A.6 – Résultats retournés par la méthode d'exploration de l'application *Croissant* pour les schémas FV, YASHE et F-NTRU.

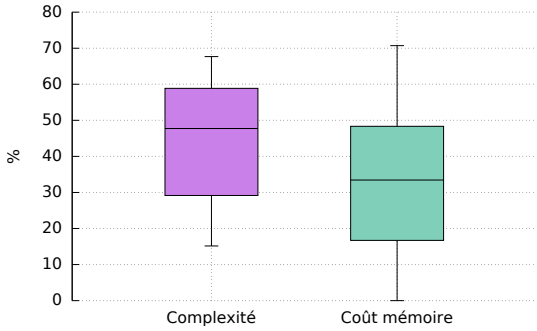
Schéma	$\log(q)$	$\log(w)$	t	Temps (sec)			Coût mémoire (Mio)		
				Estim.	Exéc.	Erreur	Estim.	Exéc.	Erreur
FV	243	81	2	12.9071	13.1468	1.8229 %	39.2560	39.8	1.3668 %
YASHE	214	31	2	110.9752	105.4862	5.2035 %	288.1556	287.1	0.3677 %
	219	37	2	103.7007	97.7876	6.0469 %	288.5537	284.7	1.3536 %
	228	46	2	96.4357	89.6419	7.5788 %	290.8168	282	3.1265 %
	243	61	2	89.1144	83.0979	7.2402 %	295.8598	276.7	6.9244 %
	272	91	2	81.7118	90.1536	9.3637 %	307.3409	305.9	0.4710 %
F-NTRU	70	4	–	994.3613	1950.8202	49.0286 %	102.6563	360.6	71.5318 %
	80	6	–	521.8772	1196.8958	56.3974 %	104.7982	283.8	63.0732 %
	82	6	–	513.2818	1237.2852	58.5155 %	110.2251	283.9	61.1747 %
	85	7	–	438.7469	1060.0179	58.6095 %	110.8015	263	57.8702 %
	86	7	–	433.0001	1045.1416	58.5702 %	113.3989	263.1	56.8989 %
	88	7	–	427.7399	1034.9479	58.6704 %	118.8157	266.9	55.4831 %



(a) FV (99 centiles).



(b) YASHE (99 centiles).

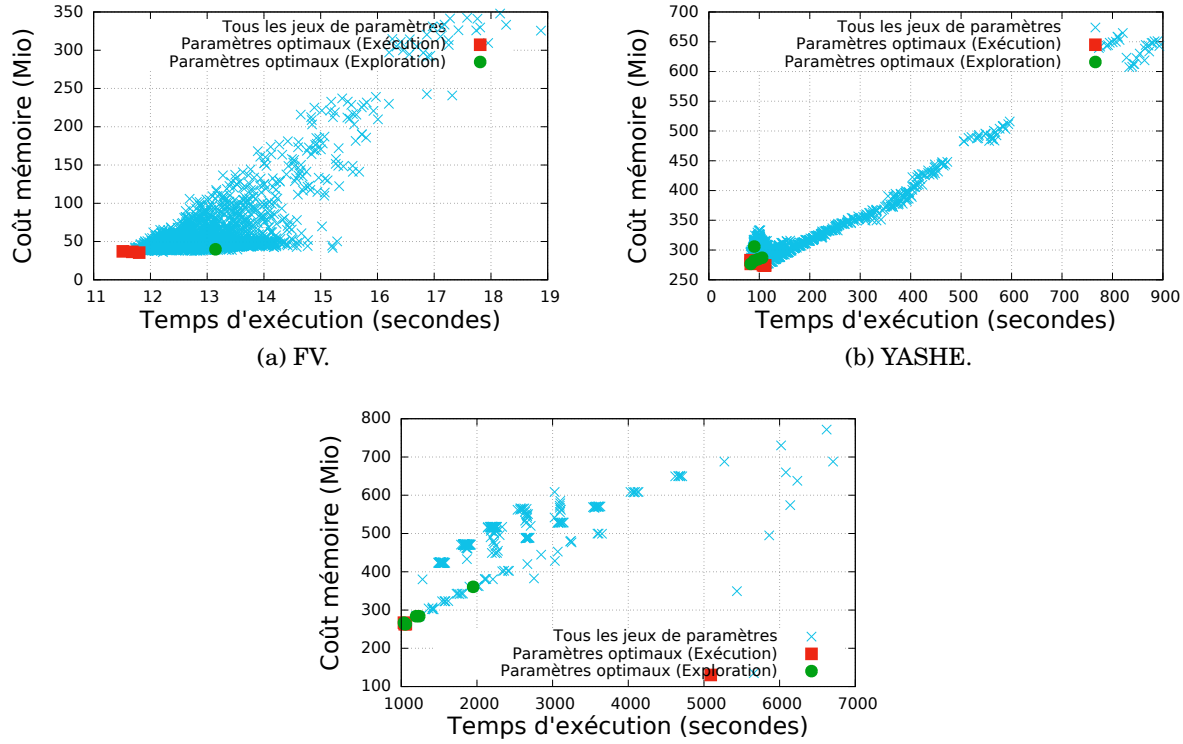


(c) F-NTRU (95 centiles).

FIGURE A.3 – Boîte de Tukey des taux d'erreur obtenus pour les résultats de l'exploration comparés aux résultats des exécutions concrètes, pour chaque schéma (avec l'application *Croissant*) et chaque analyse théorique calibrée (complexité et coût mémoire).

TABLE A.7 – Résultats retournés après l'exécution complète de tous les jeux de paramètres de l'application *Croissant* pour les schémas FV, YASHE et F-NTRU. En rouge, les jeux de paramètres présents également dans la Table A.6.

Schéma	$\log(q)$	$\log(w)$	t	**Temps (sec)**	**Coût mémoire (Mio)**
FV	253	89	2	11.8020	35.5
	265	94	2	11.5139	37.2
	276	99	2	11.6809	36.5
YASHE	211	27	2	111.7140	274.1
	211	28	2	110.8153	274.1
	212	27	2	107.2027	274.3
	212	28	2	110.4897	274.1
	213	28	2	110.0183	274.1
	243	61	2	83.0979	276.7
	244	62	2	82.4774	282.9
F-NTRU	61	2	–	5089.4322	130.3
	85	7	–	1060.0179	263
	86	7	–	1045.1416	263.1
	88	7	–	1034.9479	266.9



(a) FV.

(b) YASHE.

(c) F-NTRU.

FIGURE A.4 – Identification des points retournés par l'exploration pour l'application *Croissant*.

TABLE A.8 – Classement des points sélectionnés par l'exploration de l'application *Croissant* et écart de ces points avec le résultat optimal obtenu par exécution concrète.

Schéma	$\log(q)$	$\log(w)$	t	**Classement**		**Écart avec le point optimal**	
				Position	**En %**	**Temps (sec)**	**Coût mémoire (Mio)**
FV	243	81	2	309/2665	11.5572	1.6328	4.3
YASHE	214	31	2	550/2760	19.8913	23.0087	13
	219	37	2	254/2760	9.1667	15.3102	10.6
	228	46	2	15/2760	0.5072	7.1645	7.9
	243	61	2	1/2760	0.0000	0.6205	2.6
	272	91	2	339/2760	12.2464	7.6761	31.8
F-NTRU	70	4	–	59/259	22.3938	915.8723	230.3
	80	6	–	7/259	2.3166	161.9479	153.5
	82	6	–	8/259	2.7027	202.3373	153.6
	85	7	–	1/259	0.0	25.0700	132.7
	86	7	–	1/259	0.0	10.1937	132.8
	88	7	–	1/259	0.0	0.0	136.6

A.3 Application *Médicale*

La Table A.9 renseigne le nombre de jeux de paramètres à explorer ainsi que, pour chaque schéma :

- le temps nécessaire pour déterminer l'ensemble des jeux de paramètres à explorer,
- la durée de l'exploration pour ces jeux de paramètres,
- la durée effective pour l'exécution concrète de l'ensemble des jeux de paramètres pour l'application *Médicale*.

La Table A.9 donne également le facteur d'accélération entre le temps requis pour effectuer l'exploration et la durée requise pour réaliser l'ensemble des exécutions concrètes.

La Figure A.5 illustre la répartition des taux d'erreur visibles entre les analyses théoriques calibrées et les temps d'exécutions et coûts mémoire concrets obtenus après exécution.

La Table A.10 liste les résultats retournées par l'exploration de l'application *Médicale*.

La Table A.11 liste les résultats présents sur le front de Pareto des exécutions concrètes de l'application *Médicale*.

Dans la Figure A.6, représentant la répartition des jeux de paramètres explorés, les croix bleues représentent les résultats d'analyse concrète obtenus suite à l'exécution des jeux de paramètres. Les carrés rouges sont les points optimaux (*c-à-d* identifié sur le front de Pareto) des exécutions (croix bleues). Les ronds verts représentent les résultats sélectionnés par l'exploration de l'application avec le schéma fixé.

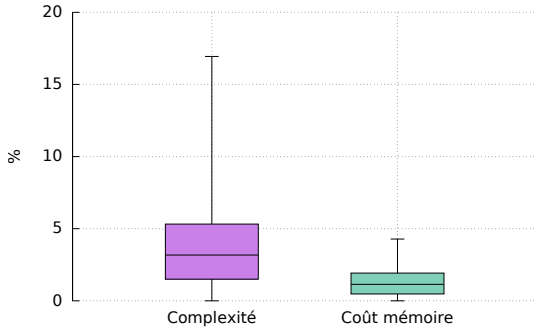
La Table A.12 donne un classement (position absolue et en pourcentage) des résultats optimaux des exécutions concrètes de l'application *Médicale*.

TABLE A.9 – Informations sur l'exploration réalisée pour chaque schéma pour l'application *Médicale*.

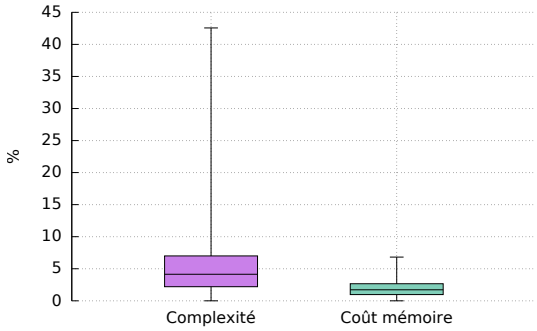
Schéma	**Nombre de jeux**	**Temps pour trouver les jeux**	**Durée d'exploration des paramètres**	**Temps total des exécutions**	**Accélération**
FV	2846	0 h 11 min 14 sec	3 h 47 min 57 sec	137 h 37 min 8 sec	36.2
YASHE	2936	1 h 21 min 24 sec	29 h 24 min 9 sec	4043 h 16 min 48 sec	137.5
F-NTRU	26	0 h 0 min 6 sec	286 h 17 min 0 sec	1691 h 16 min 0 sec	5.9
TOTAL *Médicale*	5808	1 h 32 min 44 sec	319 h 29 min 6 sec	5872 h 9 min 56 sec	18.4

TABLE A.10 – Résultats retournés par la méthode d'exploration de l'application *Médicale* pour les schémas FV, YASHE et F-NTRU.

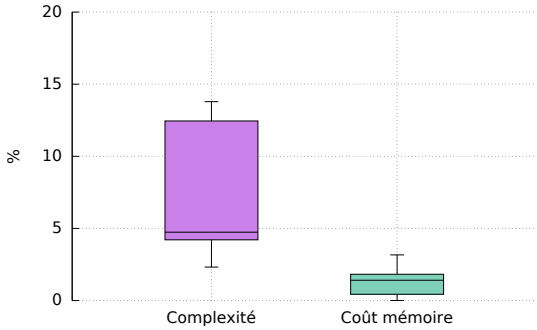
Schéma	$\log(q)$	$\log(w)$	t	**Temps (sec)**			**Coût mémoire (Mio)**		
				Estim.	**Exéc.**	**Erreur**	**Estim.**	**Exéc.**	**Erreur**
FV	393	44	2	167.2526	175.2254	4.5500 %	269.5541	265.5	1.5270 %
	437	88	2	171.0225	167.4467	2.1355 %	237.2856	240	1.1310 %
YASHE	422	47	2	2444.5887	2249.0564	8.6940 %	2117.1779	2071.5	2.2051 %
	429	54	2	2250.6326	2211.2626	1.7804 %	2121.0134	2147.2	1.2196 %
	438	63	2	2056.2451	2067.7614	0.5569 %	2128.8599	2146.9	0.8403 %
	451	76	2	1860.2292	1926.6023	3.4451 %	2147.6454	2203.4	2.5304 %
	469	94	2	1662.2165	1693.0730	1.8225 %	2179.3963	2220	1.8290 %
F-NTRU	143	4	–	117572.2446	121169.8245	2.9690 %	7391.2586	7395.6	0.0587 %



(a) FV (99 centiles).



(b) YASHE (98 centiles).



(c) F-NTRU (99 centiles).

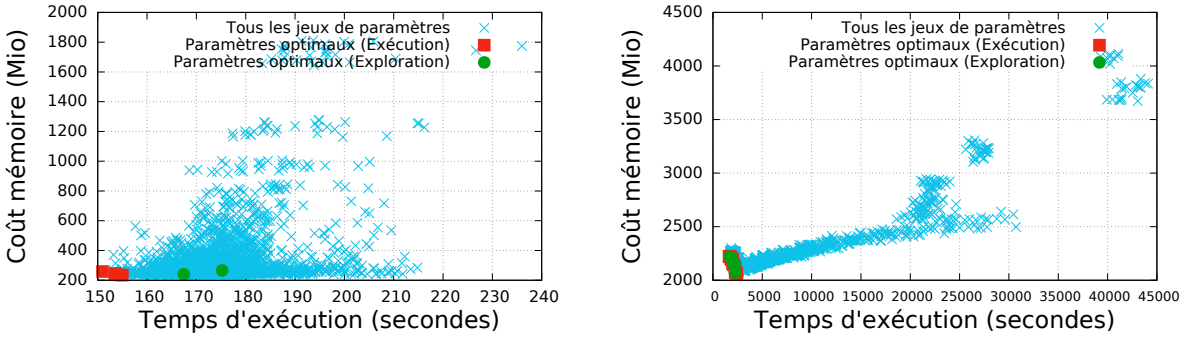
FIGURE A.5 – Boîte de Tukey des taux d'erreur obtenus pour les résultats de l'exploration comparés aux résultats des exécutions concrètes, pour chaque schéma (avec l'application *Médicale*) et chaque analyse théorique calibrée (complexité et coût mémoire).

TABLE A.11 – Résultats retournés après l'exécution complète de tous les jeux de paramètres de l'application *Médicale* pour les schémas FV, YASHE et F-NTRU. En rouge, les jeux de paramètres présents également dans la Table A.10.

Schéma	$\log(q)$	$\log(w)$	t	**Temps (sec)**	**Coût mémoire (Mio)**
FV	412	58	2	150.9541	258.7
	449	84	2	153.4657	245.3
	449	90	2	154.9951	234.6
	451	98	2	154.3875	235.8
	460	99	2	153.5041	237.3
YASHE	422	47	2	2249.0564	2071.5
	423	47	2	2417.4495	2056.9
	424	47	2	2365.4440	2071.1
	427	48	2	2239.8140	2104.5
	435	57	2	2126.4436	2143.6
	436	49	2	2228.6443	2125.2
	438	63	2	2067.7614	2146.9
	439	63	2	2046.8687	2149.9
	440	63	2	2010.3081	2153.9
	440	65	2	1988.3878	2166.7
	456	81	2	1836.3401	2201.2
	457	80	2	1840.0052	2199.3
	462	82	2	1961.8914	2197
	462	84	2	1897.7200	2197.1
	463	84	2	1985.9449	2195.4
	465	81	2	1968.9694	2196.3
	466	87	2	1883.8757	2197.2
	466	91	2	1873.0054	2198.2
	469	94	2	1693.0730	2220
	475	96	2	1644.5184	2220.5
	476	95	2	1831.7879	2214.9
	476	99	2	1633.4326	2224.7
	482	96	2	1815.3890	2215.9
	483	96	2	1823.8517	2215.5
F-NTRU	143	4	–	121169.8245	7395.6

TABLE A.12 – Classement des points sélectionnés par l'exploration de l'application *Médicale* et écart de ces points avec le résultat optimal obtenu par exécution concrète.

Schéma	$\log(q)$	$\log(w)$	t	**Classement**		**Écart avec le point optimal**	
				Position	**En %**	**Temps (sec)**	**Coût mémoire (Mio)**
FV	393	44	2	1734/2846	60.8925	24.2713	30.9
	437	88	2	133/2846	4.6381	16.4926	5.4
YASHE	422	47	2	1/2936	0	615.6237	14.6
	429	54	2	49/2936	1.6349	577.8300	90.3
	438	63	2	1/2936	0	434.3287	90
	451	76	2	419/2936	14.2371	293.1697	146.5
	469	94	2	1/2936	0	59.6404	163.1
F-NTRU	143	4	–	1/26	0	0.0	0.0



(a) FV.

(b) YASHE.

(c) F-NTRU.

FIGURE A.6 – Identification des points retournés par l'exploration pour l'application *Médicale*.

Titre : PAnTHErS : un outil d'aide pour l'analyse et l'exploration d'algorithmes de chiffrement homomorphe.

Mots clés : Cryptographie, Chiffrement Homomorphe, Modélisation, Exploration de l'Espace de Conception

Résumé : Le chiffrement homomorphe est un système de cryptographie permettant la manipulation de données chiffrées. Cette propriété offre à un utilisateur la possibilité de déléguer des traitements sur ses données privées, à un tiers non fiable sur un serveur distant, sans perte de confidentialité.

Bien que les recherches sur l'homomorphe soient, à ce jour, encore récentes, de nombreux schémas de chiffrement ont été mis au point. Néanmoins, ces schémas souffrent de quelques inconvénients, notamment, de temps d'exé-cution particulièrement longs et de coûts mémoire importants. Ces limitations rendent difficile la comparaison des schémas afin de déterminer lequel serait le plus adapté pour une application donnée, c'est-à-dire le moins coûteux en temps et en mémoire.

Cette thèse présente PAnTHErS, un outil rassemblant plusieurs fonctionnalités permettant de répondre à la problématique citée ci-dessus. Dans l'outil PAnTHErS, les schémas de chiffrement homomorphe sont tout d'abord représentés dans un format commun grâce à une méthode de modélisation. Puis, une analyse théorique estime, dans le pire cas, la complexité algorithmique et le coût mémoire de ces schémas en fonction des paramètres d'entrée fournis. Enfin, une phase de calibration permet la conversion des analyses théoriques en résultats concrets : la complexité algorithmique est convertie en un temps d'exécution estimé en secondes et le coût mémoire en une consommation estimée en mébioctets.

Toutes ces fonctionnalités associées ont permis la réalisation d'un module d'exploration qui, à partir d'une application, sélectionne les schémas ainsi que les paramètres d'entrée associés produisant des temps d'exécution et coûts mémoire proches de l'optimal.

Title : PAnTHErS: a tool for analyzing and exploring homomorphic encryption algorithms.

Keywords : Cryptography, Homomorphic Encryption, Modeling, Design Space Exploration

Abstract : Homomorphic Encryption (HE) is a cryptographic system allowing to manipulate encrypted data. This property enables a user to delegate treatments on private data to an untrusted third person on a distant server, without loss of confidentiality.

Even if current researches in HE domain are still young, numerous HE schemes have been created. Nevertheless, those schemes suffer from some drawbacks, especially, from too long execution times and important memory costs. These restrictions make difficult to compare schemes in order to define which one is the most appropriate for a given application, *i. e.* the less expensive in terms of time and memory.

This thesis presents PAnTHErS, a tool gathering several features to answer to the previous problem. In the tool PAnTHErS, homomorphic encryption schemes are first represented into a common structure thanks to a modeling method. Then, a theoretical analysis evaluates, in the worst case, computational complexity and memory consumption of those schemes according to given input parameters. Finally, a calibration phase enables conversion of theoretical analysis into concrete results: computational complexity is converted into an estimated execution time in seconds and memory cost into an estimated consumption in mebibytes.

These gathered features allowed the creation of an exploration method which, from an application, selects best schemes and associated input parameters that implies close to optimal execution times and memory costs.