Cryptographie homomorphe et transcodage d'image/video dans le domaine chiffré

Donald Nokam Kuaté

université PARIS-SACLAY

NNT: 2018SACLS575

Thèse de doctorat

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

Cryptographie homomorphe et transcodage d'image/vidéo dans le domaine chiffré

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université Paris Sud

École doctorale n°580 STIC
Sciences et technologies de l'information et de la communication
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, CEA LIST - Centre Nano-INNOV
Le 14 décembre 2018, par :

Donald NOKAM KUATÉ

Composition du jury :

M. Patrick Bas Directeur de Recherche, CNRS, École Centrale de Lille	Rapporteur
M. Sébastien Canard Ingénieur de Recherche, HDR, Orange Labs	Co-Directeur de thèse
M. Sylvain Duquesne Professeur à l'Université de Rennes I	Président
Mme Caroline Fontaine Chargée de Recherche CNRS, ENS Cachan	Examinatrice
M. Philippe Gaborit Professeur à l'Université de Limoges	Rapporteur
M. Louis Goubin Professeur à l'UVSQ	Examinateur
Mme Melek Önen Maître de Conférences, EURECOM	Examinatrice
M. Renaud Sirdey Directeur de Recherche CEA	Directeur de thèse

Remerciements

Ces quelques lignes sont pour moi l'occasion de témoigner ma gratitude aux personnes qui m'ont accompagné durant cette thèse. Ces trois dernières années m'ont permis de faire des rencontres et Dieu seul sait à quel point elles m'ont été bénéfiques sur le plan professionnel, personnel et humain. Recevez ces quelques lignes en signe de toute ma reconnaissance.

Pour commencer, je souhaite remercier Patrick Bas et Philippe Gaborit d'avoir accepté de rapporter mon travail. Vos retours ont contribué à améliorer la qualité de ce manuscrit tant sur le plan historique que sur affinage du contexte du dernier chapitre. Je remercie également Sylvain Duquesne, Caroline Fontaine, Louis Goubin, Melek Önen de m'avoir fait l'honneur de compléter mon Jury.

À mes chers directeurs de thèse Sébastien Canard et Renaud Sirdey. Je ne saurai pas trouver des mots assez forts pour vous exprimer ma gratitude tant vous m'avez apporté. Vous avez toujours été bienveillant et à mon écoute, vous avez su voir mes difficultés et m'aider à les surmonter. Vous avez trouvé des mots pour m'aider à garder le cap à chaque fois que vous voyiez que je chavirais. Vous m'avez apporté énormément sur le plan scientifique en guidant mes hypothèses et en vous assurant de leurs pertinences. Sébastien, c'est l'occasion ici de te dire un grand merci pour la confiance que tu m'as accordé, déjà en me donnant l'opportunité de faire un stage sous ta direction et me proposant cette aventure édifiante qu'est la thèse. Renaud merci pour ton enthousiasme, tu as toujours su être positif et pragmatique dans nos échanges. Pour tout cela, le mot le plus fort que je puisse trouver est simplement "Merci".

Cette thèse n'aurait pas été aussi géniale si elle ne s'était pas déroulée dans la formidable équipe SPI (Security, Privacy and Innovation) dirigé par un formidable manager Jean-François. Pour ne pas tous vous citer, tant vous êtes nombreux, je vais simplement vous dire merci. Merci, car c'est auprès de vous que j'ai pu vivre ma première expérience profession (déjà avec mon stage en suite avec la thèse). Vous m'avez énormément apporté de connaissances dans le domaine de la sécurité informatique, moi qui venais d'un milieu mathématique à la base. J'ai appris beaucoup à votre contact, non seulement lors de nos discussions de table qui étaient pour moi l'un des moments préféré de la journée, que lors des pauses-café et des discussions individuelles avec chacun de vous. Je ne peux toutefois clôturer cette partie sans dire un merci particulier à Marie (qui est partie de l'équipe vers d'autres aventures) de m'avoir co-encadré pendant mon stage de M2, merci pour ta bonne humeur, ton enthousiasme et ta perspicacité. Quentin mon collègue et mon co-bureau, je ne t'ai pas oublié. Mon aventure dans cette thèse n'aurait pas été la même sans toi. Tu es quelqu'un de très disponible et une richesse pour la culture scientifique. Grâce à toi, j'ai pu élever encore plus mon niveau de programmation et compris que la Blockchain était une solution à un problème qu'on cherche encore. Adel, Aïda, Guillaume et Maxime nous avons eu une seule année passée ensemble, mais elle a été géniale grâce à vous. Vous avez su me donner le fou rire pendant la période intense qu'est la rédaction de ce manuscrit. Je ne vous remercierai jamais assez.

Merci également à tous les membres actifs de CyberCrypt, ça été une belle aventure extra-thèse. C'était un plaisir de construire tous les équipements du jeu qui on permit de le rendre plus attrayant. C'était également un plaisir de le présenter aux différentes

éditions de la fête de la science et évènement extra liés à la science avec vous. Merci Sébastien pour l'idée de cette activité et de m'avoir inclus.

Aux membres de mon laboratoire du CEA. Je n'ai pas eu l'occasion d'être très souvent là, mais j'ai gardé de bon souvenirs de mes passages sur place. Je souhaite remercier Sergiu pour nos échanges sur l'algorithme de multiplication new FV. À Joel Cathébras, tu as été de bon conseils et nos discussions m'ont permis de me sentir moins seul dans le domaine homomorphe. Je souhaite remercier Damien Ligier qui a bien voulu se prêter à l'exercice de présenter mon article à PST.

Je remercie également mes camarades de promotion de master de l'université de Rennes I, vous avez été géniaux avec moi et m'avez aidé me sentir toute suite à mon aise dans cet environnement où je venais juste d'arriver et que je découvrais progressivement. Je remercie tous mes enseignants de la formation pour la qualité de leur enseignement et les conseils qu'ils m'ont donnés pour mon intégration.

Je souhaite également remercier tous mes enseignants de l'université de Yaoundé I pour la qualité de la formation que vous m'avez apporté. C'est grâce à vous que j'ai pu acquérir une base solide en mathématique. Cette base m'a ensuite permis de mieux m'épanouir dans la suite de ma carrière et en particulier cette thèse. Je souhaite en particulier remercier le Professeur Marcel Tonga qui est le déclencheur indirect de ma carrière en cryptographie à travers la création de ce module. Je remercie également mon déclencheur direct Thony Ezome qui a accepté de venir nous initier à cette discipline formidable qu'est la cryptographie et qui m'a permis de me convaincre que je voulais en faire mon métier.

Pour finir le souhait dire un gros merci à ma famille, aussi grande qu'elle soit et repartie entre la France, les USA et le Cameroun. Vous m'avez été d'un grand soutien moral dans ces moments difficiles où je manquais d'inspirations et que j'avais besoin d'air frais. Je souhaite remercier ici en particulier la famille Kounga qui a su m'accueillir et me conseiller comme leur fils. Merci à toi Prisca d'avoir été toujours à mes côtés pendant les moments difficiles de cette thèse. Merci à mes frères Arnaud et Armel pour disponibilités et joie de vivre contagieuse, ainsi qu'à mes sœurs Verdiane, Laure, Nadine et Sandrine vous m'avez toujours motivé dans mes choix. À mon papa et ma maman qui m'ont toujours accompagné dans leur prière, m'ont donné leur amour et m'ont toujours soutenu dans mes choix de carrière.

Je souhaite dédicacer cette thèse à mon regretté oncle Deffo Alexi. Tu as toujours cru en moi et à mon objectif de Docteur, et cela, depuis mes premiers jours à l'université. Malheureusement tu n'es plus là pour voir ce rêve se concrétiser, mais je sais que tu aurais été très fier de moi.

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Introduction générale

La cryptographie Ce mot est issu du Grec *kruptos* qui signifie cacher et *graphein* qui renvoit à l'écriture. C'est littéralement l'art de cacher l'écriture et de manière plus générale, celui de cacher l'information dans une communication. Son usage initial est d'ordre militaire. Les armées avaient besoin d'un moyen sécurisé pour échanger des informations sensibles, comme des plans ou des stratégies, entre les chefs de commandement, sur les différents champs de bataille. C'est grâce à la création des ordinateurs au début des années 1940 et surtout l'avènement d'Internet au début des années 1990, que son usage s'est développé et diversifié dans plusieurs domaines de la vie publique. Aujourd'hui, on retrouve la cryptographie dans les objets du quotidien comme les cartes bancaires et les téléphones portables ; également lors de certaines actions effectuées sur un ordinateur comme la connexion à un site web sécurisé ou le payement d'un produit sur un site de commerce en ligne. La cryptographie est aujourd'hui présente pour assurer l'intégrité, la confidentialité et l'authentification lors de toutes ces manipulations que nous réalisons sur Internet de manière presque inconsciente.

La cryptographie fait partie d'une discipline beaucoup plus grande appelée cryptologie qui contient en plus la cryptanalyse. Cette dernière est une discipline qui s'oppose à la cryptographie et consiste à étudier et analyser les faiblesses que peuvent contenir un algorithme cryptographique afin de les exploiter et de retrouver par exemple l'information qu'il protège. Les deux disciplines se livrent donc un combat perpétuel ; les chercheurs en cryptographie proposant toujours des nouvelles techniques afin de préserver toutes les infrastructures de communications qui dépendent de la cryptographie et les cryptanalystes cherchant à mettre en évidence d'éventuelles failles de sécuritées. La cryptographie dite classique déployée présentement contient, en plus des mécanismes d'authentifications et autres, deux groupes d'algorithmes de chiffrement : (1) ceux à clé privée ou symétrique qui nécessitent que les interlocuteurs de la communication possèdent la même clé pour effectuer le chiffrement et le déchiffrement d'un message et par conséquent d'avoir trouvé un moyen de s'être échangé cette clé au préalable ; (2) ceux à clé publique ou asymétrique qui ne nécessitent pas d'échange préalable de clé. Ici, chaque utilisateur possède un système de clé publique et privée. La clé publique est transmise à toute personne désireuse d'envoyer un message qui utilise cette clé pour chiffrer ; et seul celui qui possède la clé privée peut déchiffrer un tel message. Pour garantir la sécurité des algorithmes, la cryptographie à clé publique se base sur la difficulté de certains problèmes mathématiques comme la factorisation du produit de deux grands nombres premiers ou encore le calcul du logarithme discret d'un élément d'un groupe. Cependant, l'idée de l'arrivée des ordinateurs quantiques présente un sérieux problème pour la cryptographie à clé publique. En effet un ordinateur quantique réduirait la complexité de ces algorithmes à l'ordre du linéaire ; c'est par exemple le cas avec l'algorithme de P. Shor [Sho99] qui propose un algorithme quantique de factorisation d'un entier N en un temps $\mathcal{O}((\log \mathrm{N})^3)$. Pour le moment on n'en est pas encore là et les chercheurs travaillent à la création d'algorithmes qui en plus de fournir des services actuels de la cryptographie classique, nous donneraient également la sécurité contre les futures ordinateurs quantiques. Les solutions prometteuses à ce jour sont celles de la cryptographie dite post-quantique, qui popose des outils de cryptographie avancée dont la sécurité repose par exemple sur les problèmes dures des réseaux euclidiens ou des codes correcteurs. Par exemple la recherche du vecteur le plus court (Shortest Vector Problem, SVP) dans un réseau euclidien est un problème réputé NP-dure, donc difficile à résoudre même pour un attaquant qui dispose d'un ordinateur quantique. La cryptographie post-quantique propose donc des solutions qui héritent de la difficulté de ce type de problème. Les principaux problèmes auxquels font face les solutions post-quantiques actuelles sont la taille des données à manipuler et l'efficacité des algorithmes. Toutefois,

des améliorations continuent à se faire tant au point de vue algorithmique avec l'affinage des solutions existantes qu'au point de vue électronique avec l'augmentation de la puissance de calcul de nos ordinateurs.

L'arrivée de nouvelles technologies comme l'informatique des nuages (cloud) offre aujourd'hui de nouvelles perspectives de travail pour les utilisateurs : le stockage des données à distance, la délégation de calcul vers un serveur beaucoup plus puissant que notre machine locale, etc. Pour la cryptographie, cela soulève d'énormes problèmes comme celui de continuer à assurer les propriétés d'intégrité, de confidentialité, d'authentification etc... au cours d'une communication avec le nuage ; la protection de la vie privée des utilisateurs du nuage en fournissant des services qui utilisent des données chiffrées. C'est dans cette dernière optique que la cryptographie propose des solutions comme le chiffrement homomorphe qui permet d'effectuer des opérations sur des données chiffrées sans avoir besoin de les déchiffrer. Introduit dans les années 70 par Rivest et al. [RAD78] sous l'appellation d'homomorphisme confidentiel, le chiffrement homomorphe a réellement pris son envol en 2009 lorsqu'un chercheur, Craig Gentry [Gen09b] a proposé la première définition théorique d'un schéma de chiffrement qui permet d'exécuter un système complet d'opérateurs (addition et multiplication) sur les données chiffrées. Les systèmes de chiffrement homomorphe initialement présentés n'étaient pas très efficaces en taille de clés et de chiffrés (de l'ordre de giga octet) mais surtout en temps d'exécution (2 à 3 minutes pour réaliser une opération de multiplication). La recherche dans ce domaine a permis d'améliorer de manière drastique leur performances, et même si aujourd'hui la taille des clés et chiffrés restent encore importantes (quelques méga octet), le temps d'exécution lui atteint l'ordre du milliseconde pour cette même multiplication (137 ms pour le TFHE [CGGI17a]). Ces améliorations du chiffrement homomorphe nous amènent aujourd'hui à tester plus d'applications sur données chiffrées. On peut citer parmi tant d'autres le calcul de moyenne et de variance [LLN15], la calcul des transformations de Fourier [CSVW16b], le calcule du génôme à partir de données chiffrés de patient [SSA+17], ... Dans cette thèse, nous nous sommes penchés sur une application encore plus poussée, à savoir celle du transcodage vidéo.

Le transcodage vidéo L'évolution de l'électronique et des technologies de l'information permet aujourd'hui d'avoir des équipements de communication de plus en plus petit et facile à transporter : les ordinateurs portables, les tablettes ou les téléphones intelligents (smartphone) sont la preuve de cette évolution. Ils permettent de remplir énormement de fonctionnalités parmi lesquels la photographie. En effet, il est désormais facile de transmettre à un ami une photo ou une vidéo de nos activités journalières, par l'intermédiaire des réseaux sociaux (facebook, whatsapp, etc...) ou de l'informatique des nuages , sans se soucier du comment ces images ou vidéos vont s'afficher au niveau de l'appareil d'affichage de notre ami. Nous avons juste la certitude que celui-ci la recevra et pourra l'afficher correctement sur son écran. Ceci est en fait possible car les serveurs des réseaux utilisés se chargent de ce travail à notre place. En effet, lorsque nous transmettons une vidéo à un ami par le biais d'un réseau social, le serveur de ce dernier récupère la vidéo qui est encodée en un format défini sur votre appareil de capture d'images (la caméra de smartphone par exemple), et réalise donc un transcodage de cette dernière. Plus précisement, il décode la vidéo, et réalise de nouveau un encodage de celle-ci, cette fois en définissant plusieurs formats d'affichage, de qualité possible de ses images et les mettra à la disposition de toutes personnes possédant les accès à leur plateforme pour regarder la vidéo. Ainsi, lorsque votre ami se connecte à ce serveur, il peut lire le contenu de votre vidéo depuis son ordinateur, tablette ou son smartphone et tout cela avec des qualités qui dépendront de son débit de connexion et du support qu'il utilise à ce moment.

On se rend compte dans le sénario précédent, que'ne fois notre vidéo transmise au

serveur de notre réseau social préféré, celui-ci dispose d'un accès libre et découvert à celle-ci et peut en plus du service de transcodage rendu, analyser et extraire des informations de notre vidéo et l'utiliser plus tard à des fins commercials. Cette analyse de données devient de plus en plus poussée avec l'avènement des algorithmes d'intelligence artificielle qui permettent d'automatiser les calculs et les rendre beaucoup plus performants. Cela soulève donc ici un gros problème de confidentialité de nos données et de protection de notre vie privée. Une solution pour protéger nos données à la source serait d'appliquer un algorithme de chiffrement sur celles-ci, mais cela impliquerait deux issues : soit on partage une clé avec le serveur (on lui fait confiance) et la vidéo chiffrée sera déchiffrée, transcodée puis rechiffrée par le serveur avant d'être mise à la disposition de nos amis, soit on ne partage pas de clé avec le serveur, et dans ce cas on assure la confidentialité de nos données, mais aucune transformation ne pourra être appliquée à notre vidéo et elle ne pourra éventuellement pas se lire sur tous les dispositifs d'affichage existants.

Le chiffrement homomorphe est une troisième issue qui permet d'apporter à la fois la confidentialité par le chiffrement de la vidéo à la source par l'utilisateur et le transcodage de cette dernière par le serveur (en supposant qu'il dispose d'un transcodeur fonctionnant sur données chiffrées). Cela entraine la création de nouvelles vidéos toujours chiffrées qui correspondent aux différents terminaux d'affichage. En plus d'apporter cette confidentialité des données tout au long de leur traitement, les algorithmes de chiffrement homomorphe actuels sont définis sur les problèmes des réseaux euclidiens, donc supposés résister aux futures ordinateurs quantiques.

Notre travail au long de cette thèse est d'investiguer cette troisième solution en identifiant et en proposant des solutions de chiffrement homomorphe qui seraient compatibles avec les différentes transformations que contiennent un transcodeur. Cela revient à analyser le fonctionnement d'un encodeur et décodeur vidéo et effectuer des transformations sur les algorithmes qui les constituent pour qu'ils fonctionnent sur des données chiffrées ; ce qui n'est pas chose aisée au vu des contraintes imposées par le calcul à l'aveugle (voir chapitre 2). D'autres solutions comme le calcul multi parti (MPC) ou le chiffrement fonctionnel (FE) pourraient être envisagées. Cependant, le fait que le MPC nécessite de l'interactivité qui n'est pas souhaitée dans un compresseur vidéo et que le FE ne propose à ce jour pas d'implémentation suffisamment performante nous oblige à laisser ces solutions de côté dans notre étude et de nous concentrer sur celle du chiffrement homomorphe.

1.2 Contribution de la thèse

Nos travaux tout au long de cette thèse visent à proposer les prémisses d'un transcodeur qui manipulerait des vidéos et images chiffrées de la source jusqu'au destinataire final pour garantir la vie privée des utilisateurs. Dans cette optique, nous avons obtenu les résultats suivants.

Vers une compression dans le domaine chiffré avec l'étude de l'algorithme RLE. Nous avons étudié le fonctionnnement d'un compresseur d'image et de vidéo et avons jugé intéressant de commencer par l'algorithme de compression Run-Length Encoding (RLE) qui transforme une séquence d'entrée dans laquelle des symbôles se repètent consécutivement, en une séquence beaucoup plus courte qui contient les même informations que la première séquence, mais encodées sous la forme de couples (symbôle, nombres d'occurrences). Malgré sa simplicité dans le domaine clair, la transformation de cette algorithme pour un fonctionnnement sur des données chiffrées ne s'avère pas si évidente qu'on pourrait le penser à première vue et nous a réclamé des connaissances à la fois algorithmiques et mathématiques pour obtenir une solution assez optimale. Ce travail

nous a permis de nous faire une bonne entrée en matière dans la conversion d'algorithmes du domaine clair vers le domaine chiffré. Nous avons présenté les différentes transformations que nous avons faites pour le rendre compatible sur des données chiffrées. Le résulat de ce travail a été accepté et présenté à la conférence PST 2017 au Canada.

Vers une compression dans le domaine chiffré un cas d'étude avec la compression HEVC et H264. H264 et HEVC sont deux standards de compression de vidéo qui proposent une compression très efficace en temps et en espace par rapport à leur prédécesseurs (H263, H261). Ils fonctionnent en suivant le schéma de compression des tous premiers compresseurs MPEG-1. Dans les transformations de ces standards, on retrouve (dans la partie de transformations spatiales) les mêmes algorithmes que ceux contenus dans un compresseur d'images jpeg. Nous avons donc étudié les différents algorithmes d'un compresseur jpeg, analysé les différents blocs pour réaliser leur transformation dans le domaine chiffré. Nous avons proposé des solutions qui dépendent de la manière dont on choisit de représenter les coefficients des pixels d'une image (représentation des pixels par des entiers ou leur décomposition binaire). Le résultat de ce travail a été proposé, accepté et présenté à la conférence CANS 2018 en Italie.

Manipulation des slots d'un message chiffré Pour des raisons de compression et de compacité, nous avons investigué de nouvelles solutions au problème d'expansion de la taille des chiffrés vis-a-vis de leurs messages clairs associés dans les schémas de chiffrement homomorphe. En effet, le chiffré d'un bit ou un entier correspond à un polynôme de grand degré sur les entiers ou un vecteur sur les entiers qui occupe beaucoup plus d'espace. Pour chiffrer les pixels d'une image que l'on souhaite manipuler de manière indépendante par la suite, il faudrait faire un chiffrement de chaque bit individuellement. Des solutions comme le batching existent pour résoudre ce problème, mais nous proposons une solution qui permet de maximiser le nombre de slots dans le chiffré. Notre solution fournit des algorithmes d'extraction des slots en les représentant sous forme vectorielle, des algorithmes d'addition et de multiplication. Elle contient également un algorithme qui permet de revenir sous la forme d'un seul chiffré à partir du chiffré de plusieurs slots. L'intéret de notre solution est qu'elle se greffe facilement aux algorithmes existant sur les réseaux idéaux, mais également hérite de leur sécurité. Nous avons rédigé un article qui résume toutes ces idées et il sera présenté dans une conférence à venir.

1.3 Structure du manuscrit

Ce manuscrit est divisé en trois parties. La première partie, rédigée en français, est consacrée à des rappels sur le chiffrement homomorphe (Chapitre 2) : son évolution historique de son introduction en 1978 jusqu'à ce jour. Le chapitre 3 aborde les notions liées à la composition d'une image, d'une vidéo numérique et le fonctionnement des compresseurs de ces deux types de flux. Nous étudions dans ce chapitre les différents blocs algorithmiques de ces compresseurs afin d'avoir le maximun d'informations pour réaliser leur transformation vers le domaine homomorphe.

La deuxième partie (rédigée en anglais) contient le recueil de nos travaux sur la transformation des algorithmes de compression d'images et vidéos pour une exécution dans le domaine homomorphe. Le chapitre 4 commence par aborder l'algorithme de compression RLE (Run-Length Encoding). Nous étudions, transformons et implémentons cet algorithme pour un fonctionnement sur des entrées chiffrées en homomorphe. Au chapitre 5, nous regardons les algorithmes de la partie transformation d'un compresseur

d'image jpeg, qui est composée des algorithmes suivants : changement d'espace colorimétrique, transformation de cosinus discrète et quantification, ainsi que les opérations inverses associées. Ces mécanismes sont aussi présents dans la partie transformation spatiale d'un compresseur vidéo. Nous proposons des implémentations de ces différents blocs algorithmiques en mettant l'accent sur les différentes limitations de ceux-ci vis-à-vis des contraintes du chiffrement homomorphe. Nous réalisons également un état des lieux actuel des capacités des algorithmes de chiffrement en présentant les deux principales repésentations des messages chiffrés ainsi que l'arithmétique et les performances qui en découlent.

La dernière partie (rédigée en anglais également) contient un seul chapitre (le chapitre 6) qui s'intéresse à la problématique d'expansion des messages chiffrés par rapports aux messages clairs. Il présente une nouvelle méthode d'encapsulation des messages clairs en un polynôme avant le chiffrement. Il utilise également la structure polynomiale des chiffrés pour proposer un nouvelle représentation sous forme de vecteur de chiffrés qui contiennent comme message clair sous-jascent, les coefficients du polynôme initalement chiffré. Nous proposons également des opérations homomorphes d'addition et de multiplication pour cette forme vectorielle ; ainsi qu'une opération qui permet de passer de la forme vectorielle vers la forme polynômiale. Cette méthode s'applique à tous les schémas de chiffrement sur les réseaux idéaux, mais dans ce chapitre nous avons illustré nos idées en nous basant sur le schéma FV [FV12]. Nous présentons également les avantages de cette technique par rapport au transchiffrement.

1.4 Publications

Dans cette section nous faisons un simple résumé des articles que nous avons redigé et qui ont été acceptés à des conférences internationales. Nous présentons aussi un papier que nous avons rédigé et qui sera soumit dans l'une des conférences de cryptographie à venir. Et enfin nous résumons les communications que nous avons effectuées durant notre thèse.

Articles publiés

[CCNS17] *Running compression algorithms in the encrypted domain : a case-study on the homomorphic execution of RLE* (**PST 2017**).
R. Sirdey, **D. Nokam Kuaté**, S. Canard, S. Carpov.

[NKCS18] *Towards Video Compression in the Encrypted Domain : A Case-Study on the H264 and HEVC Macroblock Processing Pipeline*. (**CANS 2018**).
D. Nokam Kuaté, S. Canard, R. Sirdey.

Article à venir

Towards More Flexible Slot Management in Somewhat Homomorphic Encryption Schemes.
D. Nokam Kuaté, S. Canard, R. Sirdey.

Communications

Running compression algorithms in the encrypted domain : a case-study on the homomorphic execution of RLE(**JC2 2017**).
R. Sirdey, **D. Nokam Kuaté**, S. Canard, S. Carpov.

Projet de recherche : de la théorie à la pratique sur le chiffrement homomorphe (**JD LIST 2016 & 2017**).
D. Nokam Kuaté, S. Canard, R. Sirdey.

Première partie

Préliminaires

Chapitre 2

Le chiffrement homomorphe

Sommaire

2.1 Historique du chiffrement homomorphe

En cryptographie, le chiffrement homomorphe est un outil qui permet d'effectuer des calculs sur des données chiffrés. Il est essentiel pour répondre à un grand nombre de problèmes en informatique, comme la délégation de calculs à un serveur qui n'est pas de confiance, ou encore le problème des millionnaires. L'idée du chiffrement homomorphe ne date pas de nos jours. Elle a été énoncée pour la première fois en 1978 par les chercheurs Rivest, Adleman et Dertouzos [RAD78]. Dans ce papier, ils parlaient de schémas de chiffrement sur lequel on pouvait définir des fonctions algébriques qui s'exécutent sur des données chiffrées. Ils les appelaient des *homomorphismes de confidentialité* (privacy homomorphism en anglais). Ils ont illustré leur intuition par une série d'exemples de schémas basiques de cette époque, qui ont pour la plupart été attaqués quelques années plus tard par Brickell et al. [BY87].

En ayant pour point de départ ces travaux de Rivest et al., l'historique du chiffrement homomorphe peut être découpé en deux grandes périodes : les travaux avant l'année 2009 et ceux de 2009 à nos jours. Nous allons par la suite présenter quelques travaux majeurs qui ont marqué ces deux périodes.

2.1.1 Les schémas avant 2009

Cette époque est marquée par deux grandes classes de schémas homomorphes :

— ceux dit *partiellement homomorphes* (partially homomorphic scheme ou PHE en anglais) ; capable d'exécuter des circuits dont les portes sont constituées d'un seul type d'opération, entre l'addition et la multiplication, sur des données chiffrées ;

— ceux dit *presque homomorphes* (somewhat homomorphic scheme ou SHE en anglais) capable d'exécuter des circuits dont les portes sont constituées d'additions et de multiplications, mais avec un nombre de multiplications fini (défini par le schéma de chiffrement choisi ou le paramètre de sécurité du système) et limité à de faibles profondeurs du circuit.

Les schémas PHE L'un des premiers schémas proposé par Rivest et al. pour illustrer le chiffrement homomorphe est le schéma à clé publique RSA [RSA78]. En effet, pour un module n et un exposant publique e, on remarque qu'à partir des chiffrés de deux messages m_1 et m_2, il est possible d'obtenir celui de leur produit $m_1 \cdot m_2$:

$$\mathrm{E}(m_1) \cdot \mathrm{E}(m_2) = m_1^e \cdot m_2^e \mod n = (m_1 \cdot m_2)^e \mod n = \mathrm{E}(m_1 \cdot m_2).$$

Ce schéma est donc homomorphe pour la multiplication mais pas pour l'addition (puisqu'il n'existe à ce jour aucune définition de l'addition sur ce schéma), il rentre dans la classe des schémas PHE pour la multiplication. Dans la même classe de schéma PHE pour la multiplication, on retrouve également le schéma Elgamal [Gam84] définit par Taher Elgamal en 1984 et qui repose sur le problème du logarithme discret sur les groupes. En 1982 Goldwasser et Micali [GM82] proposent un schéma de chiffrement de bits basé sur les résidus quadratique qui est PHE pour l'addition. En 1994, Benaloh [Ben94] s'inspire du système de Goldwasser et Micali pour définir un schéma pour le chiffrement de messages de taille plus grande. Ce schéma est également homomorphe pour l'addition et sa sécurité repose sur le problème de résidus supérieurs (higher residuosity problem en anglais). Dans cette classe de schéma PHE pour l'addition, on ajoute le schéma de Paillier [Pai99] définit en 1999, dont la sécurité repose sur la version décisionnelle du problème des résidus composés.

Les schémas SHE. Le Graal pour la cryptographie homomorphe serait d'avoir un schéma qui est totalement homomorphe (fully homomorphic encryption ou FHE en anglais) ;

c'est-à-dire capable d'exécuter n'importe quel circuit déterministe, formé d'un nombre arbitraire de portes additives et multiplicatives, tout en préservant la sécurité sémantique du système, ainsi que la compacité des chiffrés (un chiffré qui vient d'être formé doit être indistinguable d'un chiffré obtenu après un certain nombre d'opérations). Pour atteindre cet objectif, les recherches ont conduit dans un premier temps à des schémas SHE. Bien que l'histoire et l'évolution des SHE soit plus marquée par les travaux réalisés après 2009, nous allons présenter ici quelques travaux précurseurs de ce mécanisme.

La première proposition de schéma SHE provient des travaux de Fellow et Koblitz [FK94] en 1994, qui proposent le schéma nommé *Polly Cracker* défini sur l'anneau de polynômes multivariés. La sécurité de leur schéma repose sur le problème d'appartenance d'un polynôme à un idéal (ideal membership problem ou IMP en anglais). Pour une base donnée $\{f_i\}$ de polynômes s'annulant tous sur un secret $(s_0, \cdots, s_n)$, le chiffré d'un message m correspond simplement à tirer un polynôme g dans l'idéal engendré par les f_i et de calculer $c = m + g$. Le déchiffrement s'effectue alors en évaluant $c(s_0, \cdots, c_n) = m$. Tel que défini, ce schéma offre naturellement la possibilité de faire une addition ou une multiplication sur les chiffrés. Cependant, il pose un problème de compacité puisque la taille des chiffrés augmente avec les opérations ; elle évolue de manière exponentielle avec la profondeur multiplicative du circuit. Tout ceci le rend donc inutilisable en pratique. D'autre part, il s'est avéré que ce schéma (pour certains paramètres) était sensible aux attaques basées sur l'algèbre linéaire [LdVMPT09], notamment l'utilisation des bases de Gröbner qui permettent de retrouver la clé secrète à partir d'un certain nombre d'échantillons du chiffré. Des améliorations du schéma Polly Cracker ont également été proposées plus tard par Levy-dit-Vehel et al. en 2004 [LdVP04] puis par Van-Ly Le en 2006 [Le06] qui le rebaptise *Polly Tow*. Ces variantes souffrent également du même problème de compacité des chiffrés qui les rend inapplicables. Quelques années plus tard (2010) Steinwandt [Ste10] a proposé une attaque généralisée contre les schémas de type Polly Cracker.

On retrouve aussi dans la littérature des schémas comme celui de Ishai et Paskin en 2007 [IP07] qui permet d'exécuter des programmes de branchement pour des données chiffrés, et bien que la taille du chiffré final ne varie pas en fonction du nombre de portes du circuit, elle évolue en fonction de sa profondeur. En 2008, anticipant les systèmes homomorphes par niveau de 2ème génération (cf. section suivante), Aguilar-Melchor et al. [AMGH08] (article finalement publié à Crypto 2010 [AMGH10]) proposent une définition théorique d'une technique qui permet de mettre en chaîne plusieurs schémas de chiffrement PHE pour construire un schéma homomorphe pour l'addition, et qui permet d'effectuer un nombre prédéfini d de multiplications consécutives d'un chiffré. Mais comme les précédents schémas, l'opération de multiplication cause une augmentation exponentielle de la taille du chiffré en fonction de la profondeur du circuit à évaluer.

L'un des premiers schémas SHE qui permet de respecter la compacité du chiffré et la sécurité sémantique est celui de Boneh, Goh et Nissim proposé en 2005 [BGN05]. Il est défini sur les groupes admettant une application bilinéaire sécurisée tel que les couplages sur les courbes elliptiques. Il permet d'exécuter n'importe quel circuit de profondeur multiplicative égale à deux ; c'est-a-dire qu'il est capable d'effectuer un nombre infini d'additions sur les chiffrés, mais juste une seule multiplication. Bien que limité en nombre d'opérations de multiplications, ce schéma a fortement inspiré Gentry lors de la construction du premier schéma de chiffrement totalement homomorphe.

2.1.2 Les schémas après 2009

2009 est une année phare pour le chiffrement homomorphe, puisque c'est en cette année que Craig Gentry a présenté la première description théorique d'un schéma totalement homomorphe. Plus précisément, il a proposé dans sa thèse [Gen09b] une méthode pour transformer un schéma SHE en un schéma FHE via une méthode originale et ingénieuse : le réamorçage (bootstrap en anglais). Son idée part du constat que les

schémas SHE souffrent soit de l'expansion de la taille des chiffrés en fonction des opérations, soit de l'augmentation du bruit à chaque opération homomorphe (pour des schémas qui utilisent un bruit pour cacher le message lors du chiffrement), limitant ainsi le nombre d'opérations possibles. Pour supprimer ce phénomène d'expansion de taille (et éventuellement du bruit), une solution consisterait à déchiffrer ces "vieux" chiffrés, puis utiliser l'algorithme de chiffrement pour construire un nouveau chiffré de taille plus petite (et contenant du bruit plus petit). Le souci évident de cette solution est qu'il est nécessaire de posséder la clé secrète pour la réaliser ! L'idée de Gentry est basée sur le fait qu'un schéma SHE est capable d'évaluer n'importe quel circuit de profondeur d sans rien apprendre des éléments manipulés. En transformant l'algorithme de déchiffrement (squashing) en un circuit dont la profondeur est strictement inférieure à d, il devient possible de l'évaluer sans rien apprendre du message clair sous-jacent ni de la clé secrète de déchiffrement. Ainsi, en fournissant d'une part la clé secrète sous forme chiffrée ainsi qu'un surchiffrement du "vieux" chiffré, il devient possible d'obtenir un " nouveau" chiffré qui soit de taille (de bruit) assez proche de celui issu directement de l'algorithme de chiffrement. La sécurité de cette méthode repose sur ce que Gentry a appelé *la sécurité circulaire* [1] puisqu'il est nécessaire d'ajouter un chiffré de la clé secrète dans les composantes de la clé publique.

Partant des travaux de Gentry, plusieurs schémas SHE avec des algorithmes de déchiffrement facilement transformables ont vu le jour, ainsi que des mécanismes de gestion de bruit (et de taille de chiffrés) et des techniques de réamorçage pour les transformer en schéma FHE. Dans la littérature, ils sont regroupés en terme de générations, et de nos jours, il est communément admis qu'il en existe quatre.

Première génération de FHE. Cette génération inclut les premiers travaux de Gentry sur le FHE. Pour illustrer sa description du réamorçage, Gentry propose ([Gen09b] chapitre 5) un schéma définit sur les idéaux non pas dans l'anneau des polynômes multivariés, mais sur les réseaux euclidiens. La sécurité de ce schéma repose sur la complexité du problème décisionnel de l'appartenance d'un polynôme à une classe d'équivalence d'un idéal du réseau (idéal coset problem en anglais). En 2010, il propose, en collaboration avec Van Dijk et al. [VDGHV10], un schéma SHE définit sur les entiers et reposant sur le problème d'Approximation du grand diviseur commun d'un entier (Approximate-GCD problem en anglais). Bien que la description de ces schémas soit simple, les chiffrés contiennent un bruit qui augmente très rapidement (exponentiellement avec la multiplication), empêchant ainsi d'exécuter des circuits de grande profondeur. D'autres propositions de schéma sur les idéaux de réseaux euclidiens peuvent être trouvées dans [SV10, GH11a, GH11b], mais se confrontent tous au même problème d'expansion du bruit. Pour contrôler l'expension de la taille des chiffrés, ces schémas utilisent dans leurs paramètres publiques une *clé de relinéarisation* qui dépend de la clé secrète et permet de ramener la taille du chiffré à une taille fixe après une opération de multiplication homomorphe.

Deuxième génération de FHE. Cette génération est marquée par les travaux de Brakerski. En 2011, il propose, en collaboration avec d'autres chercheurs, une série de travaux permettant de réduire la croissance importante du bruit dans les chiffrées [BV11b, BGV12, Bra12]. En utilisant de manière différente l'approche par produit tensoriels initiée dans [AMGH10] (voir aussi [AMBGH11]), ils proposent de nouveaux schémas qui permettent de passer d'une croissance linéaire à une croissance, ce qui conduit à des schémas dits *totalement homomorphes par niveau* (leveled fully homomorphic encryption ou LFHE en anglais), capable d'évaluer n'importe quel circuit de profondeur fixe et prédéfinie

1. Il est impossible d'obtenir une quelconque information sur la clé secréte à partir d'un chiffré de celle-ci

à l'avance. Ces travaux ont également permit de définir des schémas LFHE tels que BGV [BGV12] et le schéma proposé par Fan et Vercuateren communément appelé FV [FV12] qui sont facilement transformables en schéma FHE. Plus encore, la sécurité de ces schémas repose sur les problèmes dures des réseaux euclidiens comme LWE [Reg05], ainsi que sa version sur les anneaux des polynômes, RLWE [LPR13]. Ceux-ci offrent une meilleure exploitation de la structure algébrique des réseaux ainsi que de nombreux outils d'optimisation de calculs issus de l'algèbre linéaire, comme par exempe la transformé en nombres théoriques (number theoritic transform ou NTT en anglais). D'autres techniques ont été mises en œuvre pour améliorer les opérations sur le chiffrement homomorphe, notamment le *SIMD* introduit par Smart et al. [SV11] qui consiste à mettre plusieurs valeurs du message clair dans le même chiffré, permettant ainsi de réduire le facteur d'expansion important qu'il existe entre message clair et message chiffré. Plusieurs techniques d'amélioration du réamorçage ont été également proposées [DM14].

La troisième génération de FHE. Le schéma le plus marquant de cette génération est GSW [GSW13]. Il propose une approche légèrement différente des schémas de la deuxième génération qui consiste à considérer les chiffrés comme des matrices dont la clé secrète est une approximation du vecteur propre associé à une valeur propre qu'est le message. Plus précisément, si $\vec{v}$ est un vecteur représentant la clé secrète et C un chiffré associé au message μ, alors $C \cdot \vec{v} = \mu \cdot \vec{v} + \vec{e}$ où $\vec{e}$ représente une petite erreur. Cette approche permet de définir un schéma LFHE qui est à la fois sûr et compact puisqu'il n'a pas besoin de clés de relinéarisation pour contrôler l'expansion des chiffrés au fil des opérations. Il conserve également l'augmentation logarithmique du bruit dans les chiffrés. La sécurité de ce schéma repose cependant sur le problème LWE et non pas sur sa version polynomiale. De ce fait, il ne peut donc pas bénéficier des optimisations apportées par les schémas de la deuxième génération, ce qui le rend un peu moins performant que ceux-ci. Il reste toutefois transformable en un schéma FHE, via le procédé de Gentry.

La quatrième génération de FHE. Cette génération est définie par les travaux de Chillotti et al. [CGGI17b] qui proposent un schéma qui exploite à la fois l'idée derrière le schéma de la troisième génération et les optimisations proposées dans la deuxième. Leur schéma est défini sur un tore et la sécurité est obtenue par une réduction des problèmes des réseaux à leur version équivalente sur le tore. La principale avancée du schéma de cette génération est qu'il propose une technique de réamorçage très optimisée, permettant ainsi d'obtenir un schéma de chiffrement totalement homomorphe qui fonctionne en un temps raisonnable pour des applications pratiques.

2.1.3 En résumé

L'évolution du chiffrement homomorphe n'a pas été chose aisée. En effet, il est marqué d'un vide d'environ 30 ans depuis l'énoncé sur concept par Rivest et al., puis a connu une suite beaucoup plus favorable grâce aux travaux de Gentry en 2009. Le travail des chercheurs est aujourd'hui quasiment continu dans ce domaine, et nous sommes aujourd'hui capable d'atteindre des schémas utilisables en pratique dans plusieurs domaines, offrant ainsi un avenir prometteur au chiffrement homomorphe. Beaucoup de choses restent cependant encore à faire, notamment en ce qui concerne le facteur d'expansion entre le chiffré et le message clair, la taille des clés et l'amélioration des performances sur les opérations de base et le réamorçage.

Dans cette thèse, nous visons un usage pratique du chiffrement homomorphe à travers une application de transcodage vidéo sur les données chiffrées. Pour nous aider dans cette réalisation nous nous interrogeons sur la faisabilité de ce projet et surtout des

outils que l'on dispose à cet effet. Nous allons donc regarder, dans la suite, ce que dit la théorie sur l'usage du chiffrement homomorphe en pratique.

2.2 La machine homomorphe

Une machine homomorphe est une machine de Turing qui est capable de fonctionner sur des données chiffrées. Dans cette section, nous allons présenter celle-ci théoriquement, ainsi que ses capacités et ses limitations. Nous allons ensuite identifier les premières conséquences qu'ont ses caractéristiques sur la réalisation d'un transcodeur vidéo fonctionnant sur des données chiffrées.

2.2.1 Que dit la théorie ?

Une machine de Turing consiste en la donnée de trois tables :
— $\mathrm{F} : \mathrm{S} \times \Sigma \longrightarrow \mathrm{S}$; fonction de mise à jour des états ;
— $\mathrm{G} : \mathrm{S} \times \Sigma \longrightarrow \Sigma$; fonction de mise à jour des symboles ;
— $\mathrm{D} : \mathrm{S} \times \Sigma \longrightarrow \{-1, 1\}$; fonction de déplacement de la tête de lecture ;
où S dénote l'ensemble des états de la machine et où Σ dénote son alphabet.

En tant qu'objet théorique, la machine fonctionne sur une bande supposée infinie. Au démarrage de son exécution, la machine est dans un état donné q_0 et sa tête de lecture est à la position 0 sur la bande. En cours d'exécution, les données nécessaires au fonctionnement de la machine sont simplement l'état s dans lequel elle se trouve et la position p de sa tête de lecture sur la bande. Ainsi, à chaque étape de son exécution, la machine commence à lire le symbole σ se trouvant sur la bande en position p puis à écrire $\mathrm{G}[s, \sigma]$ en position p et faire $p := p + \mathrm{D}[s, \sigma]$ (déplacement de la tête de lecture) puis $s := \mathrm{F}[s, \sigma]$ (mise à jour de l'état).

Tout l'enjeu de la cryptographie homomorphe consiste à pouvoir réaliser n'importe quel calcul public (du point de vue du calculateur) sur des données chiffrées. Dans les termes précédents, cela signifie que les tables F, G et D ne sont pas chiffrées mais que la bande, l'état et la position de la tête de lecture de la machine le sont. Exécuter une machine de Turing, et donc n'importe quel calcul, ne nécessite donc que deux opérateurs :

1. un opérateur d'addition dans le domaine chiffré (pour la mise à jour de la position de la tête de lecture) ;
2. un opérateur de déréférencement d'un tableau (clair) avec un indice chiffré qui étant donné un tableau t et un entier i chiffré (dont on sait qu'il est valide pour t) retourne $t[i]$ sous forme chiffrée (un tel opérateur permet de réaliser les déréférencements des trois tables, ainsi que la lecture sur la bande).

Bien entendu, une telle machine ne peut travailler que sur une approximation à bande finie.

Etant donné un système homomorphe sur $\mathbb{Z}_2$, il existe de nombreuses façons de réaliser les deux opérateurs ci-dessous (cf. [FSF+13a] pour un exemple parmi d'autres). On notera cependant que les opérateurs de déréférencement et d'affectation d'un tableau avec un indice chiffré ont une complexité linéaire en la taille du tableau (versus une complexité constante sur une machine usuelle).

La théorie nous enseigne donc que l'on peut réaliser n'importe quel calcul dans le domaine chiffré sans limitation de principe. Il est même possible en principe d'aller jusqu'à cacher la spécification de la machine elle-même, et donc de faire fonctionner un ordinateur conjointement sur un algorithme et des données qu'il ne connait pas ! Cette remarque n'a cependant d'intérêt que théorique en l'état actuel de l'art sur le chiffrement homomorphe.

De nos connaissances actuelles en chiffrement homomorphe, il faut par contre travailler dans $\mathbb{Z}_2$ pour atteindre la Turing-complétude.

Pas de problème de principe en vue pour réaliser un encodeur ou un décodeur vidéo semble-t-il ? Nous allons déjà voir que même en principe, les choses ne sont pas si simples qu'elles y paraissent initialement.

2.2.2 Capacités opérationnelles de la machine homomorphe

Lorsqu'elle travaille sur des entiers n-bits, la machine homomorphe permet de réaliser l'ensemble des opérateurs usuels sur ces entiers : on peut les additionner, les multiplier, faire des changements de signe, des soustractions ou encore des décalages (avec ou sans propagation de signe pour le décalage vers la droite). Nous proposons au lecteur de se reporter de nouveau à [FSF+13a] pour les détails de mise en œuvre de ces opérateurs qui ne présentent pas de difficultés particulières.

C'est dans la mise en œuvre des comparaisons, et plus précisément dans l'exploitation qu'il est possible de faire de résultats de comparaisons que nous allons trouver l'essence des capacités de la machine homomorphe. Avec les opérateurs précédemment évoqués à notre disposition, il est aisé de réaliser un opérateur $<: \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n} \longrightarrow \mathbb{Z}_2$ qui suffit pour faire tout type de comparaisons ($>$, $\geq$, $\leq$, $==$, ...). Nous reviendrons dessus dans le chapitre 4. La subtilité est que la valeur de retour de l'opérateur est elle-même chiffrée et donc inaccessible pour le calculateur qui exécute l'algorithme. En particulier, cela signifie que ce dernier peut évaluer des conditions mettant en jeu des données du domaine chiffré mais ne peut en aucun cas exploiter directement les résultats de ces évaluations, par exemple pour s'orienter dans le flot de contrôle du programme exécuté.

Il suit que la machine homomorphe ne peut pas réaliser de structures de type **si ... alors ... sinon ...** qui mettraient en jeu des données du domaine chiffré. En tout cas pas directement. Une autre conséquence est qu'elle ne peut pas exécuter directement de structures de boucle avec critère d'arrêt dépendant de données chiffrées. Relativement au domaine chiffré, la machine homomorphe ne peut donc exécuter que des programmes dits à structure de contrôle statique (i.e., programmes pouvant toujours être ramenés a priori à une séquence linéaire d'instructions).

Cependant, plusieurs techniques de régularisation existent, permettant de faire rentrer des programmes à structure de contrôle dynamique dans le carcan de la structure de contrôle statique. Pour la structure **si ... alors ... sinon ...**, il est par exemple possible d'avoir recourt à une instruction d'affectation conditionnelle (i.e., analogue à l'opérateur `?:` du language **C**). Celui-ci nécessite d'exécuter systématiquement les deux branches de la structure et de recombiner les résultats de manière appropriée en fonction de la valeur de la condition. L'opération d'affectation conditionnelle se réalise alors simplement dans le domaine chiffré par :

$$x := c?a : b \equiv x := c \otimes a \oplus (1 \oplus c) \otimes b.$$

Nous donnerons plus d'explications sur ce point dans le chapitre 4.

En résumé, nous allons devoir travailler avec une machine aux caractéristiques inhabituelles, avec des structures conditionnelles régularisées, des boucles dont le nombre d'itérations est a priori borné et des algorithmes qui réalisent toujours *au moins* leur complexité pire cas (relativement à une machine "classique"), sachant que, par construction, le temps d'exécution d'un algorithme sur la machine homomorphe ne varie pas en fonction des données chiffrées.

2.3 Panorama sur les schémas de chiffrement homomorphe actuels

Après avoir réalisé notre étude théorique sur la possibilité de réaliser une machine de Turing homomorphe, nous allons passer à la présentation des schémas de chiffrement

homomorphe utilisés de nos jours, ainsi qu'un simulateur de notre machine de Turing pour un choix particulier de schéma FHE.

Definition d'un schéma de chiffrement homomorphe De manière formelle, un schéma de chiffrement homomorphe est la donnée de quatre algorithmes (`KeyGen`, `Enc`, `Dec`, `Eval`) probabilistes qui s'exécutent en temps polynomial et fonctionnent de la manière suivante.

- Un algorithme de génération de clé `KeyGen` prend en entrée un paramètre de sécurité 1^λ et retourne des paramètres publics PP, une clé secrète sk, une clé publique pk.
- Un algorithme de chiffrement `Enc` qui prend en entrées les paramètres publics PP, un message clair m, la clé publique pk et retourne un chiffré $c = \texttt{Enc}(pp, pk, m)$.
- Un algorithme de déchiffrement `Dec` qui prend en entrées les paramètres publics PP, un chiffré c, la clé secrète sk et retourne un message $\mathrm{M}' = \texttt{Dec}(pp, sk, c)$.
- Un algorithme d'évaluation `Eval` qui prend en entrées les paramètres publics PP, la clé publique pk, un circuit C défini sur n, des chiffrés $(c_1, c_2, \dots c_n)$ portant sur les messages $(m_1, m_2, \dots, m_n)$ respectivement. Il retourne un chiffré $c = \texttt{Enc}(pp, pk, \mathrm{C}(m_1, m_2, \dots m_n))$.

Les paramètres publics PP incluent bien souvent la clé de relinéarisation qui est utilisée par l'algorithme `Eval` pour contrôler l'expansion des chiffrés. Si le schéma est homomorphe par niveau LFHE, on y retrouve aussi un paramètre L qui fixe la profondeur maximale des circuits que le schéma peut évaluer. Et dans le cas d'un schéma totalement homomorphe, PP contient également un chiffré de la clé secrète qui est utilisée, comme expliqué précédemment dans le réamorçage des chiffrés.

La plupart des schémas utilisables aujourd'hui en pratique sont définit sur les réseaux euclidiens et reposent sur le problème *LWE* [GSW13] et ses variantes (sur l'anneau des polynômes Ring-LWE [BGV12, FV12] et sur le tore TLWE [CGGI17b]) ou sur le problème *NTRU* [BLLN13b]. Les schémas sur *NTRU* sont de moins en moins utilisés depuis l'attaque en 2016 de Ducas et al. [ABD16]. En effet, les auteurs déconseillent d'utiliser le polynôme $x^n + 1$ comme module pour les polynômes, ce qui réduit l'efficacité algorithmique des schémas homomorphes basés sur ce problème. Dans la suite de notre étude, nous nous sommes intéressés aux trois schémas les plus répandus aujourd'hui : *BGV*, implémenté dans la bibliothèque HElib, TFHE, implémenté dans la bibliothèque qui porte le même nom et FV, utilisé dans la bibliothèque SEAL, et pour lequel nous avons réalisé notre propre implémentation.

Pour nous aider dans notre travail, nous avons eu besoin d'un simulateur de machine de Turing homomorphe. Nous avons utilisé pour cela la bibliothèque *Cingulata* [LIS] qui permet d'effectuer des calculs comme sur un compilateur classique du type C ou C++. La figure 2.1 décrit l'architecture générale de ce compilateur. *Cingulata* offre un cadre de

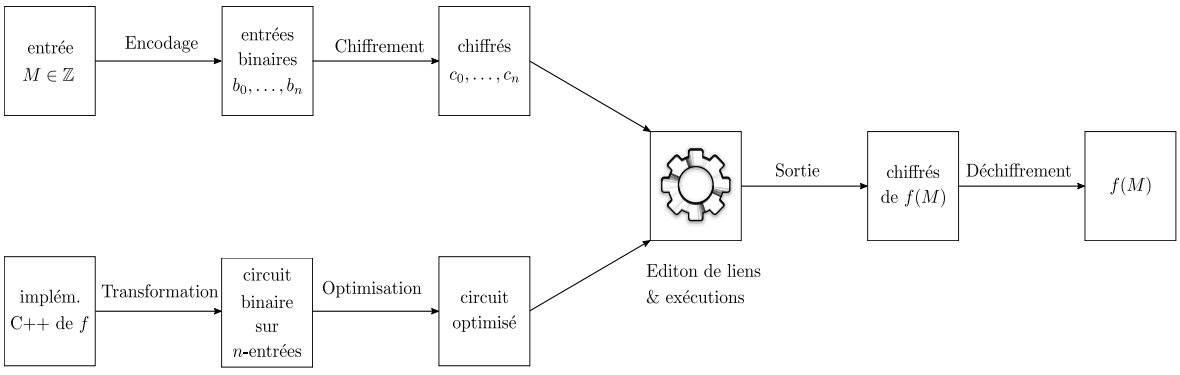

FIGURE 2.1 – Architecture *Cingulata* pour le calcul d'une fonction f sur la donnée M chiffrée en homomorphe

travail très intéressant avec un grand nombre d'opérations homomorphes sur bits chiffrés et dans lequel il est assez aisé d'implémenter les circuits à évaluer. Il est également modulable et offre la possibilité de choisir un schéma de chiffrement homomorphe qui permettra de chiffrer les entrées du circuit.

2.3.1 Conclusion

L'objectif de ce chapitre était de faire un tracé clair et concis de l'évolution du chiffrement homomorphe, en présentant les différentes idées émises par la communauté cryptographique au fil du temps. Il démontre l'existence d'une machine de Turing sur des données chiffrées et présente un certain nombre de schémas de chiffrement homomorphe que l'on retrouve dans la littérature et que nous avons étudié, implémenté et utilisé dans ce manuscrit. Il en ressort que le chiffrement a connu un grand nombre d'obstacles, principalement lié à l'expansion des chiffrés et à l'augmentation du bruit lors des opérations effectuées sur des données chiffrées. Bien que des solutions efficaces aient été apportées à ces problèmes, elles ne sont certainement pas encore les plus optimales et il reste du travail à effectuer. Nous avons également noté qu'il était possible de construire une machine homomorphe permettant de travailler sur des entrées chiffrées, exactement comme le ferait une machine de Turing sur des données claires. Nous avons présenté les schémas les plus réputés actuellement et sans doute les plus prometteurs dans le domaine de la cryptographie homomorphe. L'un des objectifs de notre recherche étant de proposer un transcodeur vidéo qui fonctionne sur des images chiffrées, nous allons maintenant nous intéresser à la manière dont les images et vidéo sont codées. Notre objectif sera alors de proposer des algorithmes qui permettraient de manipuler des images chiffrées. Nous savons que cela est théoriquement possible puisqu'il est possible de définir une machine de Turing homomorphe. Nous allons montrer comment le rendre pratique.

Chapitre 3

Compression d'image et de vidéo

Sommaire

3.1 Préliminaires sur la compression vidéo

L'application centrale de cette thèse consiste à pouvoir mettre en place un transcodeur vidéo qui fonctionne dans le domaine chiffré. Nous avons dans le chapitre précédent fait une étude des outils cryptographiques qui pourraient nous aider dans cette tâche. Il est maintenant temps de passer à l'étude du fonctionnement d'un transcodeur. A première vue, transcoder consiste à décoder un flux vidéo, puis le ré-encoder en utilisant les paramètres souhaités pour le flux de sortie. Cela nous conduit donc à devoir comprendre en détails ce qu'est une vidéo et comment fonctionnent la compression et la décompression. C'est tout l'objet de ce chapitre.

3.1.1 Positionnement de l'étude par rapport aux standards

Dans cette étude, nous nous focalisons principalement sur H.264/MPEG-4 AVC qui est le standard le plus usité aujourd'hui (standardisation en 2003/04) ainsi que sur le plus récent H.265/HEVC (standardisation en 2013), sans chercher à positionner l'un par rapport à l'autre, mais en essayant d'exploiter les avantages de chacun d'entre eux pour notre étude. En effet, le premier standard nous paraît déjà être suffisamment représentatif et riche d'intéressantes difficultés et le second possède beaucoup d'optimisations par rapport au premier, et nous tenterons de les exploiter lors de nos implémentations. Les ouvrages de référence ayant servi de base à l'écriture de ce manuscrit sont donc [Ric10a] et [SBS14]. Les autres références seront citées au fur et à mesure.

Il est à noter que l'implémentation ne serait-ce que d'un décodeur H.264 est déjà un exercice d'une redoutable complexité, il nous semble donc suffisant à ce stade de l'étude de regarder ce standard.

À noter également que pour ces deux standards, il existe plusieurs implémentations open source dont les plus respectées sont celles du projet VideoLAN [1] : x264 [2] et x265 [3]. Ces implémentations nous ont servis à extraire des implémentations de noyaux algorithmiques dans notre étude.

3.1.2 La compression JPEG

Description d'une image matricielle. Une image matricielle ou bitmap (par opposition à une image vectorielle) est un tableau rectangulaire constitué de pixels (voir figure 3.1). Elle est caractérisée par trois paramètres : sa hauteur qui est le nombre de pixels sur une colonne, sa largeur qui est le nombre de pixels sur une ligne et le type de pixel (noir/blanc, gris, et couleur). Une image noir/blanc a des pixels qui sont représentés par un bit d'information : généralement 0 =noir et 1 =blanc. Les pixels d'une image en gris eux sont des entiers entre 0 et un maximum $d = 2^n - 1$ ($n = 4, \ldots, 16$) définis par la norme de codage (les entiers dans cet intervalle représentent les différentes nuances de gris allant du noir pour 0 au blanc pour d). Une image couleur quant à elle a des pixels qui sont généralement représentés par un système de trois entiers R, V et B (chacun variant généralement entre 0 et un maximum de $2^n - 1$ [4]) qui représentent respectivement les couleurs de base rouge (R), vert (V) et bleu (B) qui sont utilisées pour représenter un grand nombre de couleurs du monde réel. Suivant la qualité souhaitée de l'image, les entiers R, V et B varient dans un grand intervalle qui définit les nuances de couleurs

1. `www.videolan.org`.
2. `www.videolan.org/developers/x264.html`.
3. `www.videolan.org/developers/x265.html`.
4. $n = 8$ est la valeur de nuance de couleur que l'on retrouve le plus souvent dans les appareils d'affichage d'image. On peut avoir un plage beaucoup plus large (les images HDR utilisent des plages allant jusqu'à 32 bits), mais cela nécessite d'avoir un afficheur capable d'interpréter cette large gamme de nuances

que l'on souhaite pouvoir représenter. Plus cet intervalle est grand, meilleures sont les nuances apportées à l'image. Mais la taille du fichier est de plus en plus grande.

En pratique, pour stocker ou transmettre une image, on l'encode suivant un format (JPEG, PNG etc...) afin de réduire la quantité d'informations à manipuler. Dans notre étude, nous nous intéressons principalement à la compression JPEG (voir figure 3.2) puisque c'est ce type de compression qui permet d'atteindre un bon niveau de compression tout en préservant une qualité raisonnable de l'image. C'est aussi elle qui est utilisée dans l'encodeur de la plupart des standards de compression comme par exemple le H264 qui nous interesse. (voir figure 3.3 pour l'encodeur du standard H264).

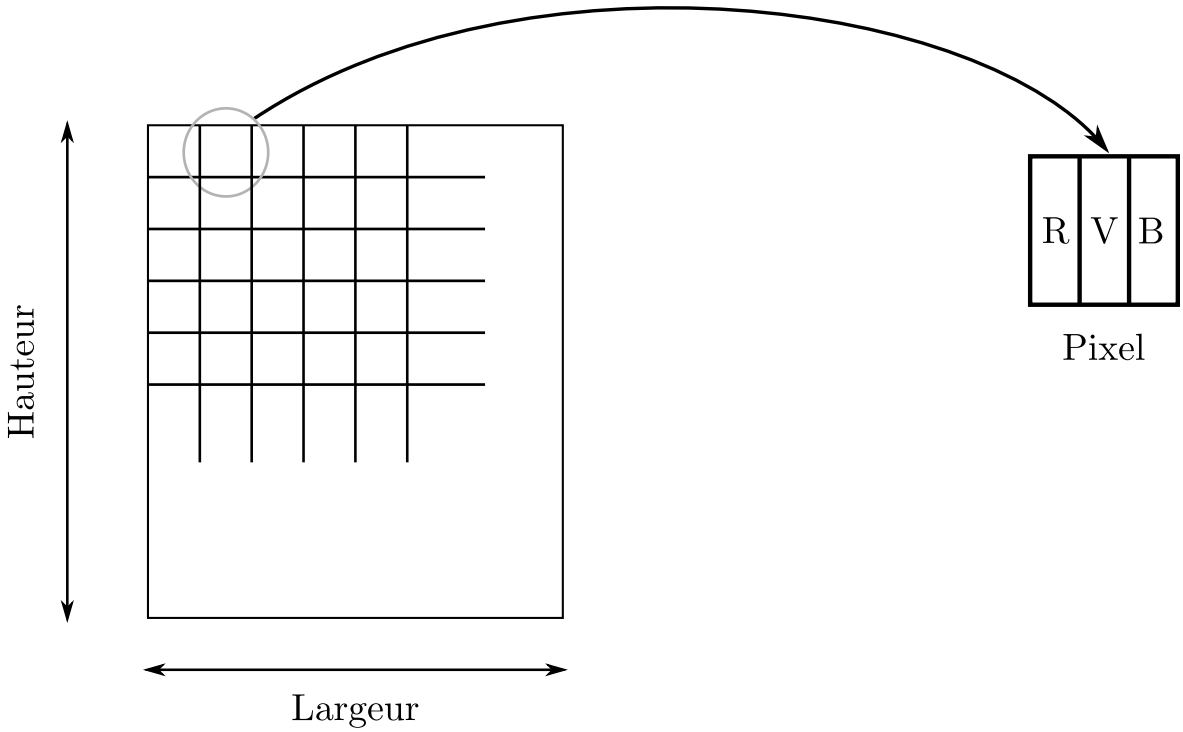

FIGURE 3.1 – Composition d'une image numérique en couleur

Présentation du codec JPEG. De son vrai nom ISO/CEI 10918-1 UIT-T Recommendation T.81, JPEG (*Joint Photographic Experts Group*) est une norme qui définit le format d'enregistrement et l'algorithme de décodage pour une représentation numérique compressée d'une image. C'est la norme de compression la plus répandue sur Internet puisqu'elle permet d'obtenir de très bon ratio de compression pour une dégradation modérée de l'image finale. Les algorithmes qui interviennent dans le pipeline de la compression JPEG (la figure 3.2 présente leur ordre d'exécution) sont les suivants.

- **La transformation de couleurs** qui change le système de représentation de l'image couleur de RVB vers le système YCrCb de luminance et chrominance. Ce dernier système sépare la couleur en terme d'intensité 'lumineuse' (luminance Y) et de 'teinte' de la lumière (chrominance rouge Cr et bleu Cb). L'oeil humain étant plus sensible à la variation de lumière que de teinte, ce système permet ainsi de compresser l'image en réduisant les teintes.
- **Le sous-échantillonnage** consiste à sélectionner les chrominances suivant un certain format définit dans le standard (il en existe quatre : 4 :4 :4, 4 :2 :2, 4 :1 :1 et 4 :2 :0). Par exemple, le format d'échantillonnage 4 :2 :0 permet de conserver la matrice de luminance Y inchangée et, sur celle des chrominances, de considérer une colonne sur deux en verticale, et une ligne sur deux en horizontale. Les matrices de chrominances de l'image entière se retrouvent alors chacunes avec

une taille quatre fois plus petite que celle des luminances.

— **Le découpage en blocs** permet de découper l'image représentée dans le nouveau système YCrCb en de petits blocs et les regroupés en *macroblocs*. Un macrobloc est donc composé d'un bloc de luminance associé à deux blocs de chrominance. La taille de bloc est généralement de 8×8 mais cela peut changer suivant l'échantillonnage choisit (pour l'exemple au format 4 :2 :0, la taille des blocs dans le macrobloc est de 8×8 pour les luminances, et 4×4 pour les chrominances). La taille de ces blocs a été fixée de sorte que les implémentations matérielles des transformations qui suivent cette étape soient les plus optimales possible.
— **La Transformé en cosinus discret** (DCT) permet de convertir les coéfficients des blocs vers une représentation en fréquence, qui a pour avantage de concentrer les informations importantes du bloc dans un petit nombre de coefficients (coefficients de basse fréquence) contenues dans la partie supérieure gauche du bloc transformé et les autres coefficients 'peu importants' (coefficients de haute fréquence)[5] dans la partie inférieure droite.
— **La quantification** est la partie qui entraine la perte dans la compression et qui permet aussi de gagner le plus de place. Elle consiste à mettre à zéro les coefficients de haute fréquences du bloc en divisant les coefficients du bloc par un entier appelé coefficient de quantification. Plus cet entier sera grand, plus on aura des coefficients nuls dans le bloc et moins la qualité de l'image sera bonne, et inversement pour les petits entiers.
— **Le codage du bloc** permet de représenter le bloc de manière à ce que les informations redondantes (les coefficients à zéro du bloc) se succèdent et puissent être codées par la suite en utilisant le moins de caractères possibles. Dans le codec JPEG, c'est le codage en *zigzag* qui transforme le bloc en un tableau à une dimension dont les premières entrées sont les coefficients de basse fréquence du bloc et les derniers sont ceux de haute fréquence (généralement des valeurs nulles suivant le coefficient de quantization utilisé).
— **La compression RLE** permet de coder le tableau en une suite de couples (*entrée, occurrence*), où *entrée* représente la valeur d'un coefficient contenu dans le tableau et *occurence* représente le nombre de fois que 'entrée' se répète successivement (ex. $5,5,5,3,3,3,3,2,2,2,0,0,0,0,0$ devient $(5,3),(3,4),(2,3),(0,5)$).
— **Le codage de Huffman** est un codage probabiliste qui permet de représenter les caractères les plus fréquents avec le moins de bits possibles.

Excepté l'algorithme de quantification qui élimine certains coefficients DCT, causant une perte d'information, tous les autres algorithmes précédent sont inversibles. La décompression consiste donc simplement à appliquer leur inverse respectif sur le flux compressé. Sauf que cette fois-ci, l'exécution des algorithmes se fait dans le sens inverse que celui présenté ci-dessus.

La compression JPEG joue un rôle important dans la compression vidéo. Elle est appliquée sur toutes les images d'une vidéo, après une étape de traitement sur les images qui vise à supprimer le maximum de redondance. Nous allons dans la section suivante entrer plus dans le détail de la compression d'une vidéo.

5. Les coefficients de haute fréquence ne sont pas détectable par l'œil humain, donc peuvent être atténués sans entrainer une grosse pertubation visuelle de l'image finale

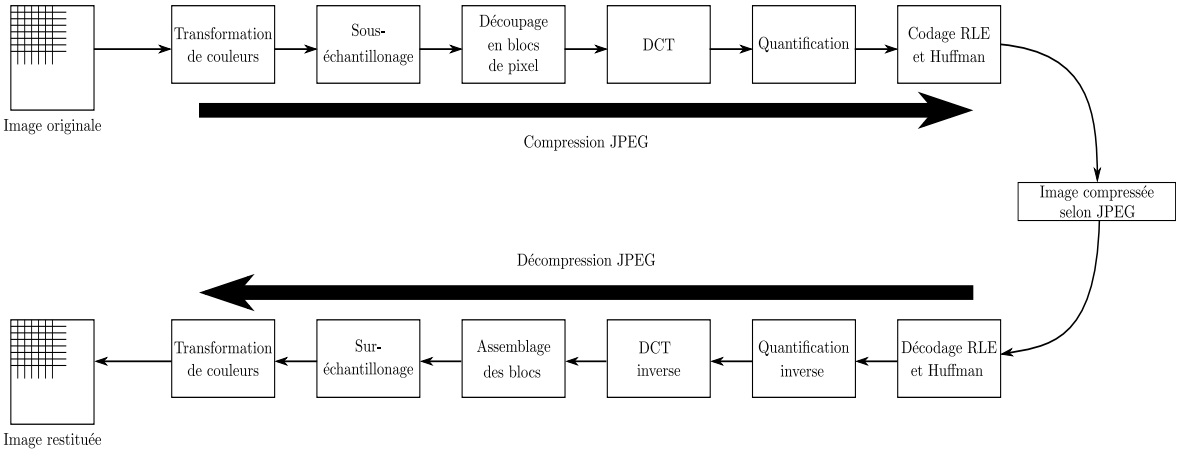

FIGURE 3.2 – model théorique d'un compresseur/décompresseur JPEG

3.2 Bases d'un schéma de compression vidéo

3.2.1 Qu'est ce qu'une vidéo ?

Une vidéo est une succession d'images ou trames (généralement accompagnées d'une bande son) qui défilent à une cadence bien définie et donne, à l'œil humain, la sensation de mouvement des objets dans une image. L'oeil humain est en effet capable de distinguer environ 20 images par seconde (trame/seconde). En affichant un nombre d'images supérieur à cette fréquence, il est possible de le tromper et lui donner l'impression qu'une image est animée.

Si on devait transmettre les images contenues dans une vidéo sans compression, cela nous donnerait une vidéo de très grande bonne qualité mais poserait un très fort problème de transmission et de stockage. En effet, si on considère une image de taille 1024×768, elle est composée d'environ 786432 pixels. Sachant qu'un pixel couleur correspond à environ 24 bits, soit 3 octets ; une telle image nécessiterait 2.25Mo pour la stocker. Une seconde d'une vidéo formée de telles images correspondrait donc à transmettre 25 images (comme préconisé par PAL/SECAM, standard de diffusion de vidéo en Europe), et donc d'avoir un débit d'au moins 57Mo/sec qui correspond à une consommation assez importante de la bande passante d'un réseau. D'autre part, l'augmentation de la puissance des CPU (Loi de Moore) fait de la bande passante le maillon faible pour la diffusion d'une vidéo de haute qualité, puisque les traitements pour l'affichage d'une image se font presque instantanément. Le but de la compression vidéo est donc de pouvoir réduire les redondances d'informations contenues dans une vidéo en réalisant une compression de ses images non seulement dans le temps, mais également dans l'espace. La figure 3.3 représente un compresseur d'image vidéo tel que définit dans le standard MPEG-4 [6][Ric10a] qui permet d'avoir de très bon taux de compression de la vidéo.

3.2.2 Compression vidéo

De manière plus schématique, compresser une séquence vidéo consiste à exploiter les redondances spatiales, temporelles et spatio-temporelles (voir figure 3.4) qui apparaissent naturellement dans un flux vidéo destiné à être interprété par un œil humain. Une longue explication n'est pas nécessaire : une scène naturelle contient généralement des zones assez homogènes (redondance spatiale), des zones (ex., un fond lointain) qui ont tendance à persister au cours du temps (redondance temporelle) et du fait de la

6. H.264, ou MPEG-4 AVC (Advanced Video Coding), ou MPEG-4 Part 10, est une norme de codage vidéo développée conjointement par l'UIT-T Q.6/SG16 Video Coding Experts Group (VCEG) ainsi que l'ISO/CEI Moving Picture Experts Group (MPEG) et est le produit d'un effort de partenariat connu sous le nom Joint Video Team (JVT) ; confère wikipedia

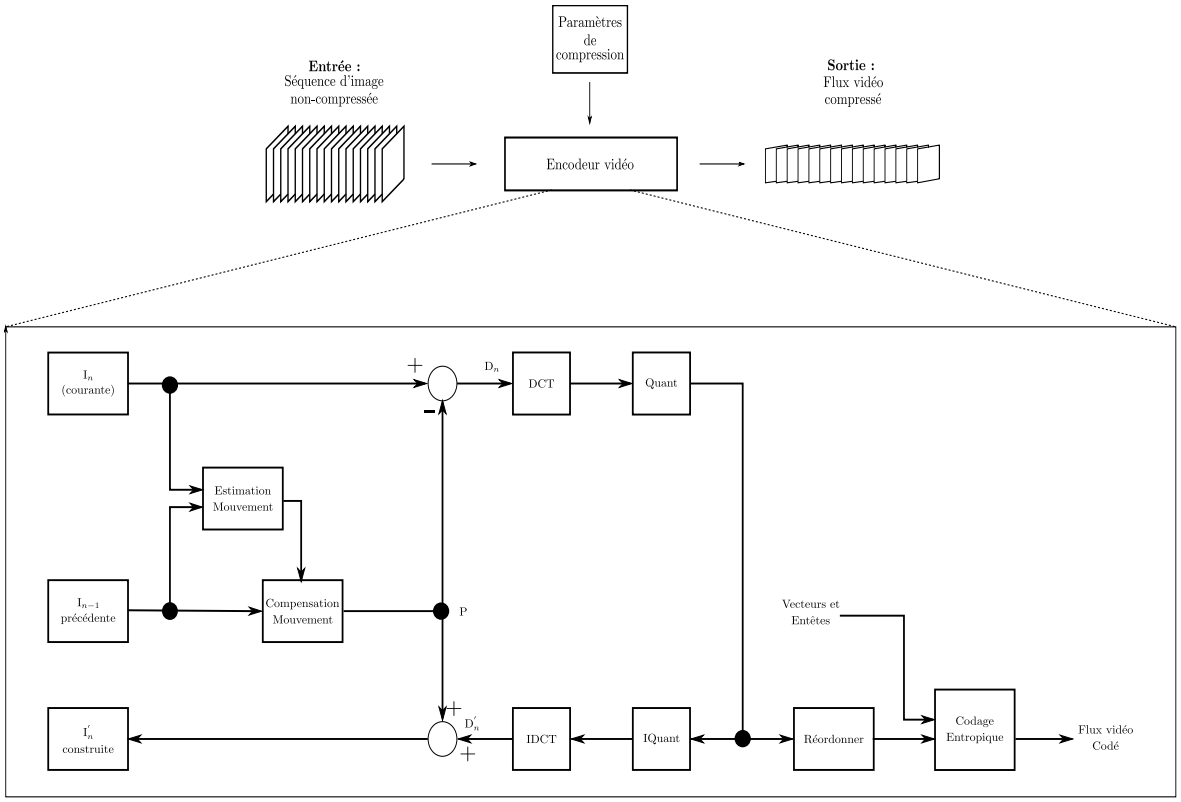

FIGURE 3.3 – Conception théorique d'un flux vidéo H264

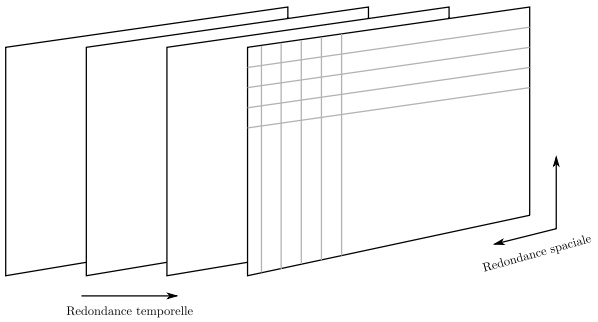

FIGURE 3.4 – Redondances spaciales et temporelles

continuité du monde physique ainsi que de la relative rigidité des objets qui compose des zones exhibant une persistance à des transformations géométriques simples près (redondance spatio-temporelle).

Dans ce contexte, un procédé de compression vidéo vise à éviter de transmettre ces informations redondantes (ainsi qu'à coder le plus efficacement possible l'information résiduelle) afin de diminuer les exigences de bande passante nécessaires. Également, la grande majorité des schémas de compression vidéo modernes codent l'information résiduelle de manière non conservative en utilisant les techniques de compression JPEG, ce qui signifie que l'image en sortie du décompresseur est différente de l'image en entrée du compresseur. Pour que cette perte ne se remarque pas ou peu, on exploite le fait que le cerveau humain est plus sensible à certaines informations qu'à d'autres (essentiellement les basses fréquences spatiales sont plus importantes que les hautes, de manière analogue au contexte voix).

Pour améliorer le taux de compression, toutes les images d'une vidéo ne sont pas compressées de la même manière, On distingue en effet trois types d'image dans le flux de sortie d'un compresseur vidéo :

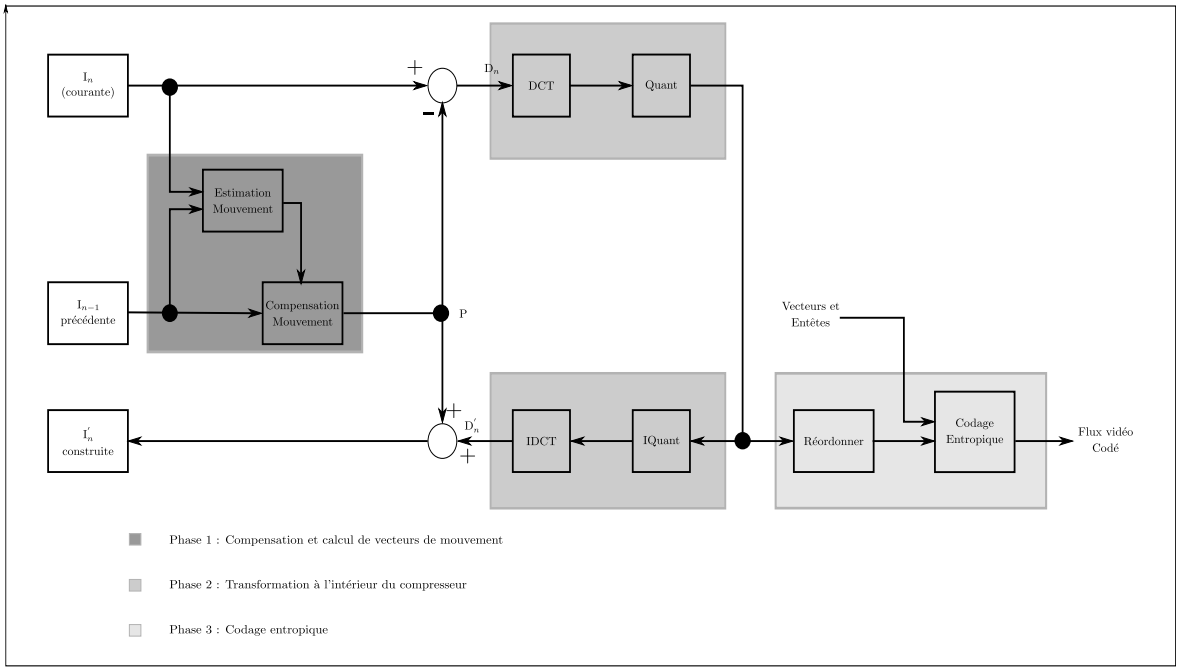

FIGURE 3.5 – Les trois phases de l'encodage vidéo

— une image I ou intra, est généralement construite en premier lors du changement de plan dans une vidéo. elle est compressée exactement comme une image JPEG ;
— une image P ou inter, est obtenue par prédiction à partir d'une image I ou P précédement créée ;
— une image B ou bidirection, c'est une image générée à partir des images I et P précédement construites mais également celle à venir.

Les images I sont donc indépendantes et le compresseur applique le pipeline de compression d'une image JPEG. En revanche, les images P et B doivent subir un prétraitrement avant de pouvoir être compressées.

En général, un compresseur vidéo fonctionne sur une décomposition de l'image à compresser (quelque soit son type I, P ou B) en macroblocks par exemple 16×16 (la taille des macroblocks n'est pas nécessairement homogène) qui sont traités en grande partie de manière indépendante.

À haut niveau, pour un macroblock donné, un pipeline de compression se décompose en trois phases dont les deux dernières sont similaires à la compression JPEG (voir figure 3.5) :

1. une phase dite de compensation de mouvement qui produit la différence entre le macroblock de l'image courante et un macroblock de l'image de référence causalement antérieure ;

2. une phase dite de transformation où le macroblock différence est passé dans une représentation qui tend à concentrer l'information utile sur un petit nombre de coefficients (utilisant la DCT), les coefficients trop petits étant éliminés (grace à la quantification, d'où la perte dans la compression). Nous incluons également dans cette phase les transformations inverse appliquées par le décodeur interne du compresseur ;

3. une phase de codage dit entropique qui consiste à compresser, cette fois sans perte, l'information résiduelle résultante.

3.2.3 Compensation de mouvement dans une vidéo

Pour un macroblock donné, l'étape de compensation de mouvement consiste à rechercher un bloc de taille équivalente qui en est le plus proche possible (au sens d'une distance donnée) au sein d'une fenêtre de recherche donnée. Généralement cette fenêtre de recherche correspond à un voisinage rectangulaire centré sur le macroblock dans une ou plusieurs images antérieures (macroblock codé en inter) complémenté d'une fenêtre causale dans l'image courante (macroblock codé en intra[7])[8]. Une distance de type somme des différences absolues (SAD) est couramment utilisée[9] mais de nombreuses autres distances sont possibles, le principe étant que l'on souhaite une distance plus rapide à évaluer que le taux de compression in fine du macroblock mais qui reflète bien ce que sera ce taux une fois le macroblock passé dans l'intégralité du pipeline de codage. Il est également prévu que la fenêtre de recherche puisse permettre de compenser des mouvements au niveau sous-pixelique, auquel cas on s'autorise à comparer le macroblock à coder à des blocs qui n'existent pas mais qui sont obtenus par interpolation (généralement) linéaire.

La compression d'un macroblock étant avec perte, une subtilité est que la recherche ne doit pas être réalisée dans le flux vidéo d'entrée mais dans le flux vidéo qui sera décodé. Pour cette raison, l'encodeur décode en permanence sa propre sortie afin de fournir le flux de référence par rapport auquel seront encodés les prochains macroblocks/images. Tout encodeur embarque donc sa propre fonction de décodage, au moins jusqu'à l'étape de transformation (puisque l'étape de codage entropique est sans perte).

Au delà de la définition de la fenêtre de recherche, de nombreuses façons de la parcourir sont possibles ainsi que de nombreux critères d'arrêt de la recherche qui peut être exhaustive ou tronquée en qualité (on s'arrête dès que l'on trouve un bloc dont la distance avec le macroblock à coder est inférieure à un seuil), en temps (on garde le meilleur bloc rencontré dans un budget temps donné) ou en nombre de distances calculées.

L'étape de compensation de mouvement consomme généralement la part la plus importante du temps de calcul d'un encodeur. Elle est l'étape à la fois qui autorise la plus grande liberté d'implémentation (un standard indique comment décoder, pas comment coder) et qui nécessite le savoir-faire le plus important. Dans tous les cas, cette étape est source d'un parallélisme important. Néanmoins, dans un contexte de transcodage homomorphe, il est vraisemblablement intéressant d'éviter de recalculer les vecteurs de mouvement en réutilisant ceux du flux entrant.

À la sortie de cette étape, on dispose d'une différence de macrobloc qui sera compressé comme un bloc JPEG, et du vecteur de mouvement qui permet de le lier à un autre macrobloc de l'image précédent et/ou suivante selon que l'on compresse une image P ou une image B.

3.2.4 Transformation à l'intérieur d'un compresseur

L'étape de compensation de mouvement revient à construire une prédiction de ce que va être un macroblock de l'image courante et calcule la différence entre cette prédiction et la réalité afin d'obtenir un résidu. Si la prédiction est de qualité, ce macroblock résidu doit contenir significativement moins d'information (d'entropie) que le macroblock et donc doit pouvoir être codé sur moins de bits. C'est l'objet de la suite du pipeline de compression macroblock que nous allons détailler dans cette section et la suivante.

7. À noter que des images exclusivement codées en intra, donc compressées de manière sous-optimale, sont régulièrement forcées afin de limiter dans le temps la propagation d'éventuelles erreurs.

8. En principe, il est même possible d'étendre la fenêtre de recherche à des images futures mais dans ce cas elle devront être encodées avant.

9. D'où de nombreuses extensions processeurs la supportant, par ex. l'instruction `PSADBW` sur la gamme Intel.

L'étape de transformation consiste généralement en une transformation *linéaire* apparentée à une transformation de Fourier (discrète) visant à concentrer l'information du macroblock résidu sur peu de coefficients (typiquement ceux qui représentent les basses fréquences spatiales). Les transformations les plus utilisées dans la compression d'image, sont la transformée de cosinus discrète (DCT) ou encore, dans une très moindre mesure, des transformées en ondelettes (discrètes) qui permetent d'obtenir de meilleurs résultats, mais au prix d'une mémoire vive (RAM) très important durant la transformation. Dans notre étude, nous nous sommes limité à la transformation DCT (voir section 5.4.2).

Les transformations sur les macroblocs mettent en jeu des nombres au format flottant (les données images brutes sont par ailleurs déjà dans ce format puisque les entiers RVB sont rapidement basculés au format YCrCb), il en est de même pour les macroblocs issus de la DCT qui doivent être quantifiées, i.e., ramenées sur un alphabet à 2^n symboles (n petit). Cette quantification peut être linéaire (à intervalles constants en divisant les coefficients du bloc par un unique coefficient de quantification) ou non (à intervalles non constants en divisant par une matrice de coefficients de quantification qui atténuent les coefficients de haute fréquence du bloc et préserve ceux de basse fréquence à une haute précision).

Le macroblock ainsi transformé est enfin soumis à une première étape de compression par un algorithme RLE, que nous avons déjà présenté dans la section 3.1.2 et sur lequel nous reviendrons plus longuement dans le chapitre 4. Pour ce faire, les coefficients de la transformée sont parcourus dans un ordre fixé a priori (généralement en zig-zag) visant à maximiser l'efficacité d'un tel codage de la transformée.

3.2.5 Codage entropique

Le caractère non conservatif des techniques de compression vidéo présentées dans cette section est uniquement dû à l'étape de quantification que nous venons de décrire (appliquée aux macroblocks indépendamment les uns des autres). Une fois cette étape faite, les données engendrées pour chacun des macroblocks (vecteur de mouvement, échelle de quantification, coefficients significatifs de la transformée, ...) forment un flux sur lequel est appliqué un algorithme de compression conservatif combinant codage prédictif (par exemple cherchant à exploiter des redondances potentielles entre les vecteurs de mouvement de macroblocks voisins), tenant compte de la sémantique des éléments de données constitutifs du flux, puis un codage de longueur variable le plus simple d'entre eux étant un codeur de Huffman avec table de symboles statique.

3.2.6 Problématique du transcodage

L'objet de cette section était de faire un court état des lieux des bases de la compression vidéo. Dans ce contexte, on peut dire qu'il ne semble pas vraiment y avoir de "problématique" autour du transcodage qui se résume simplement à décompresser, transformer (si nécessaire) puis à (re)compresser (voir figure 3.6). Soulignons que ce pragmatisme est tout à fait justifié sur le plan génie logiciel : un décodeur (et a fortiori un encodeur) étant une entité logicielle très complexe, la dernière chose que l'on souhaite est de devoir développer des logiques de décodage/encodage intrinquées avec chaque nouvelle transformation de flux vidéo. Toute déviation de ce schéma trivial ne saurait donc être justifiées que par des contraintes d'optimisation intrinséquement liées aux caractéristiques de la machine d'exécution.

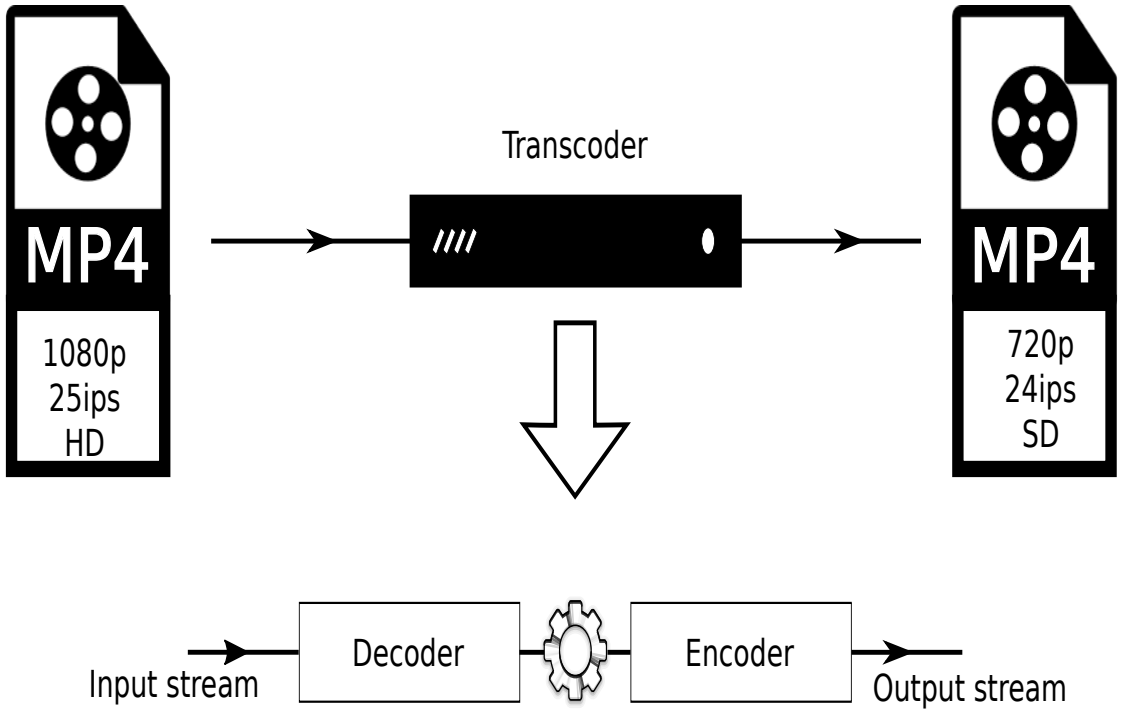

FIGURE 3.6 – Trascodage avec modification des paramètres du flux vidéo

Conclusion

Maintenant que nous avons une assez bonne idée du fonctionnement d'un compresseur de vidéo, particulièrement d'image, nous allons alors pouvoir nous atteler à notre analyse de la mise en place d'un transcodeur homomorphe (qui fonctionne sur les images chiffrés). Pour notre étude, nous considérons initialement une image au format *.bmp* pour les couleurs et au format de *.pgm* pour celle en gris. Nous chiffrons leurs coefficients avec un algorithme homomorphe, et le résultat constitue notre image chiffrée qui est l'entrée de base des algorithmes que nous allons étudier par la suite. Manipuler une image chiffrée nous impose donc de transformer l'algorithme de compression afin qu'il puisse être compatible avec les limitations en terme de structure et d'arithmétique qu'on retrouve dans le domaine homomorphe. Nous allons dans les chapitres suivants étudier et transformer les différents blocs d'algorithmes présents dans chaque phase (figure 3.5) d'un encodeur.

Deuxième partie

Prémisses d'une compression vidéo dans le domaine chiffré

Chapitre 4

Running compression algorithms in the encrypted domain : a case-study on the homomorphic execution of RLE

Sommaire

4.1 Introduction

In the realm of algorithms, data compression ones are particular beasts. As everyone knows, at least in loose form, such an algorithm is expected, given an input x, to turn it into an output $y = \alpha(x)$ such that y is much smaller than x and that there exists α^{-1} such that $\alpha^{-1}(y) = x$ (in this chapter we will consider only lossless compression [1]). In essence, any data compression algorithm fails to achieve its goal on almost all of its possible inputs, most often generating y's which are much larger than the corresponding x's. This is so because the practical compression ratio of an algorithm is always defined relatively to a non-uniform probability distribution over its input domain. Then, a useful compression algorithm is expected to have short outputs only for the most likely sequences in this domain.

In this chapter, we study the way such compression algorithm can be executed when the input data are encrypted, using fully homomorphic encryption (FHE). It should be emphasized here that running compression algorithms in such encrypted domain is an important topic for the longer-term applications of FHE techniques. Indeed, in the context of the cloud computing and particularly the video and image distribution, one current problem is to know how to process videos such that they can be easily distributed over different types of screen (TV, smartphone, computer, etc.). To solve this problem, the existing solutions make use, in particular, of a compression algorithm. In case a video is protected using an encryption scheme, it would be interesting to know how to compress data in such encrypted domain.

Take for example the "Hello world !" of data compression which we are going to study in depth in this chapter : the Run-Length Encoding (RLE) algorithm. Given an input sequence of symbols, this algorithm straightforwardly outputs a sequence of pairs counter/symbol which compresses symbols repetitions. Although this algorithm fails to compress straightforward non random sequences, e.g., of alternating symbols, it works well-enough in certain practical contexts such as low color depth image coding (e.g., as in the BMP format) or quantized DCT coefficient coding (as in most video coding standards, H264 ahead [Ric10b]). So, although there are many more sophisticated algorithms which avoid this kind of pitfalls and work quite well on more general purpose data, studying such a simple algorithm is already of significant practical relevance.

In fact, one can define a universal (yet not computable) compression algorithm to output the smallest Turing machine able to output a given sequence [2]. But such algorithm still runs into the same kind of difficulties as the framework of Kolmogorov complexity theory precisely tells us that most strings are incompressible (that term being synonymous to random w.r.t. that theory) [LV09]. In essence, any compression algorithm has a very bad worst-case behavior and, *when ran in the encrypted domain using FHE*, any algorithm intrinsically realizes its worst-case behavior. The contribution of this chapter is a first attempt to reconcile these two antagonistic points of view. Despite of its simplicity, the study of the RLE algorithm is rich enough for illustrating our points as well as drawing general conclusions.

The chapter is organized as follows. In Sect. 4.2, after briefly reviewing the capabilities and limitations of the "FHE computer" (i.e. the abstract machine which emerges from the low-level API of an FHE scheme), we derive a regularized variant of the RLE algorithm which fits the constraints of that machine. Then, in Sect. 4.3, we make some attempt at effectively running several variants of this algorithm, more and more optimized towards

1. Lossy compression would be modeled as requiring the existence of a decompression algorithm β such that $\beta(y)$ would be close enough to x according to some criterion which may even be subjective as in audio or video coding.

2. This universal compression algorithm would be able to compress strongly encrypted data as, unless another more compact encoding exists, it could at least be able to retrieve the cleartext data, the key and the encryption algorithm, then output a small Turing machine outputting the cleartext data as well as an another small Turing machine for the encryption step.

"FHE friendlyness". In doing so, we try to point out general (both manual and automatic) optimization techniques and best practices for the "FHE programmer". Lastly, Sect. 5.5 concludes the chapter with some remarks and perspectives.

4.2 Regularization of RLE

We now propose to study in depth, in the case of its execution in the encrypted domain, the most elementary compression algorithm there is : the RLE algorithm (still, it is practically used in a number of image formats, e.g., the BMP format, as well as for coding the quantized DCT coefficients in most video coding formats). The RLE algorithm simply consists, given a sequence of symbols α_1, α_2, ..., in producing a sequence of pairs counter/symbol which compresses symbols repetitions. For example, the sequence 0, 0, 2, 3, 3, 3, 4, 0, 0, 0, 0 would be turned into $\{2,0\}$, $\{1,2\}$, $\{3,3\}$, $\{1,4\}$, $\{4,0\}$. Despite of its simplicity, this algorithm cannot be implemented per se to run homomorphically. We shall now look into the reasons as to why it is so and, most importantly, into how and to which extent an implementation (compliant with an encrypted domain execution) is yet possible in principle. In this section, we have opted for a tutorial-style presentation to help the reader following our reasoning in a step by step fashion.

4.2.1 Operational capabilities of the FHE computer

Let $\mathbb{Z}_t$ be the cleartext domain (usually the cleartext domain may even be enlarged to a polynomial ring of degree n with coefficients in $\mathbb{Z}_t$) and Ω be ciphertext domain (e.g. pairs of polynomials in a certain polynomial ring), then, at the lowest level, most FHE schemes provide the following "API" :

— $\mathsf{enc}_{\mathsf{pk}} : \mathbb{Z}_t \longrightarrow \Omega$ (encryption).
— $\mathsf{dec}_{\mathsf{sk}} : \Omega \longrightarrow \mathbb{Z}_t$ (decryption).
— $\mathsf{add}_{\mathsf{pk}} : \Omega \times \Omega \longrightarrow \Omega$ (encrypted domain addition, e.g. a XOR gate when $t = 2$).
— $\mathsf{mul}_{\mathsf{pk}} : \Omega \times \Omega \longrightarrow \Omega$ (encrypted domain multiplication e.g. an AND gate when $t = 2$).

Usually, with (somewhat)FHE schemes such as BGV [BGV12] or FV [FV12] (which are presently the two most efficient implemented FHE candidates) the mul operator is several orders of magnitude more costly (say by a factor of a thousand) than the add one.

So, when it works on n-bits integers (with $\mathbb{Z}_2$ as the cleartext domain), the FHE computer allows to perform all usual operators : addition, multiplication, sign change, subtraction, left and right shift (with or without sign extension) and so on. The reader is referred to [FSF+13b] for details on how to implement such operators.

It is when performing comparisons, or more precisely when trying to use the results of comparisons between encrypted-domain values, that the true essence of the FHE computer is revealed. With the aforementioned operators, it is easy to create an operator $<: \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n} \longrightarrow \mathbb{Z}_2$ which enables any other comparison operator ($>$, $\geq$, $\leq$, $==$, ...). The subtlety is of course that the result of the evaluation of such an operator remains inaccessible to the computer running the algorithm. In particular, this means that our machine *can* evaluate any condition depending on encrypted-domain data but *cannot* exploit these results for example in order to influence the control flow of the program it is running !

Hence, the FHE computer cannot per se execute an "if ... then ... else" construct with a condition depending on encrypted-domain data, at least not directly. Also, it cannot directly execute a loop structure with a termination criterion depending on a data in the encrypted domain. Another peculiarity is that although we can perform array dereference and assignment (with an encrypted index) in the FHE domain, we can only do so in $\mathrm{O}(n)$, rather than the usual $\mathrm{O}(1)$ [FSF+13b]. More generally, relatively to encrypted-domain data, the FHE machine can only execute so-called static control structure programs (i.e., programs which can always be turned into a linear sequence of instructions).

Static control structure programs are formally equivalent to boolean circuits (i.e. networks of logical gates modelled by acyclic directed graphs) and, when it comes to efficient encrypted domain execution, *the* key parameter that need to be optimized is the multiplicative depth of the circuit which is the largest number of AND gates on any path from an input to an ouput of the circuit. Indeed, with FHE schemes such as BGV [BGV12] and FV [FV12], increasing the multiplicative depth results in a nonlinear increase in both ciphertext size and mul operator cost (as we shall see in Sect. 4.4, it is even sometimes advantageous to perform much more AND gates albeit at lower depth). This is illustrated in Table 4.1.

TABLEAU 4.1 – Illustration of the ciphertext size and AND gate computation time in terms of the multiplicative depth (FV scheme with $t = 2$ and $\lambda \approx 128$.

depth	kb/bits	ms/AND
0	3	-
1	10	4
2	21	8
5	83	28
7	148	66
10	282	150
15	601	376
20	1039	680

Still, a number of regularization techniques do exist in order to turn dynamic control structure programs into static ones. As an example, for the "if ... then ... else" construct, all is needed is a conditional assignment operator (e.g. the C language : ? operator) which can be implemented as

$$x := c?a : b \equiv x := c \otimes a \oplus (1 \oplus c) \otimes b.$$

One can then execute both branches of the construct and then appropriately recombine the results according to the condition value using the latter operator, without learning any information about the (encrypted) values x, a, b and c.

In summary, we have to work with an unusual machine allowing only regularized conditional structures and loops with an a priori bounded number of iterations as well as having the property that algorithms always realize (at least) their worst case complexity (relatively to a "classical" machine) | keeping in mind that the running time of an algorithm on the FHE machine cannot *by construction* vary in terms of the encrypted data.

4.2.2 Regularization of the algorithm control flow

Let us now walkthrough the consequences of this in the context of RLE.

As a starting point, Fig. 4.1-RLE-0 provides some C-style pseudo-code of the usual implementation of the RLE algorithm (in the clear domain). As already hinted, this implementation is not compliant with the constraints of the FHE machine : the loop of line 9 progresses by variable increments which depend on the number of iterations of the loop of line 11 which stopping criterion depends on data which would be encrypted in an homomorphic execution (input). So let us attempt to regularize this algorithm in a step by step fashion. For this purpose, we give our intermediate results on Fig. 4.1-RLE-1 and Fig. 4.2-RLE-2 respectively. The final description we propose for the RLE algorithm is then given in Fig. 4.2-RLE-3 and will be explained in the next section.

The first step (Fig. 4.1-RLE-1) consists in getting rid of the non-constant increments of the loop index i. This is done at line 10 of RLE-1. Still, this requires to put the output of j (the symbol counter) and the associated symbol in an if statement which condition

```
01. int main(void) {
02.   int n_chars;
03.   char *input;
04.   cin>>n_chars;
05.   input=new char[n_chars];
06.   assert(input);
07.   for(int i=0;i<n_chars;i++)
08.     cin>>input[i];
09.   for(int i=0;i<n_chars;) {
10.     int j=0;
11.     while(i+j<n_chars && input[i+j]==input[i])
12.       j++;
13.     cout<<j<<" "<<input[i]<<endl;
14.     i+=j;
15.   }
16. }
```

RLE-0

```
01. int main(void) {
02.   int n_chars;
03.   char *input;
04.   cin>>n_chars;
05.   input=new char[n_chars];
06.   assert(input);
07.   for(int i=0;i<n_chars;i++)
08.     cin>>input[i];
09.   int i,j=1;
10.   for(i=1;i<n_chars;i++) {
11.     if(input[i]!=input[i-1]) {
12.       cout<<j<<" "<<input[i-1]<<endl;
13.       j=1;
14.     }
15.     else
16.       j++;
17.   }
18.   cout<<j<<" "<<input[i-1]<<endl;
19. }
```

RLE-1

FIGURE 4.1 – Pseudo-code for variants of the RLE algorithm, from the usual implementation towards a variant which fits the constraints of an homomorphic execution. Would-be encrypted variables are in light-gray. See text.

```
01. int main(void) {
02.   int n_chars;
03.   char *input;
04.   cin>>n_chars;
05.   input=new char[n_chars];
06.   assert(input);
07.   for(int i=0;i<n_chars;i++)
08.     cin>>input[i];
09.   int i,j=1;
10.   for(i=1;i<n_chars;i++) {
11.     if(input[i]!=input[i-1])
12.       cout<<j<<" "<<input[i-1]<<endl;
13.     j=input[i]!=input[i-1]?1:j+1;
14.   }
15.   cout<<j<<" "<<input[i-1]<<endl;
16. }
```

RLE-2

```
01. int main(void) {
...
13.   for(int i=0;i<n_pairs;i++) {
14.     output_chr[i]='a';
15.     output_ctr[i]=0;
16.   }
17.   int i,j=1,k=0;
18.   for(i=1;i<n_chars;i++) {
19.     for(int l=0;l<n_pairs;l++) {
20.       output_ctr[l]=l!=k?output_ctr[l]:j;
21.       output_chr[l]=l!=k?output_chr[l]:input[i-1];
22.     }
23.     k=input[i]!=input[i-1]?k+1:k;
24.     j=input[i]!=input[i-1]?1:j+1;
25.   }
26.   for(int l=0;l<n_pairs;l++) {
27.     output_ctr[l]=l!=k?output_ctr[l]:j;
28.     output_chr[l]=l!=k?output_chr[l]:input[i-1];
29.   }
30.   for(int i=0;i<n_pairs;i++)
31.     cout<<output_ctr[i]<<" "<<output_chr[i]<<endl;
32. }
```

RLE-3

FIGURE 4.2 – Pseudo-code for variants of the RLE algorithm, from the usual implementation towards a variant which fits the constraints of an homomorphic execution. Would-be encrypted variables are in light-gray. See text.

depends on the (encrypted) input sequence (either the symbol changes and the algorithm prints the symbol counter and the associated symbol or it does not change and the symbol counter is just incremented). Then, the assignment of j (lines 13 and 16 of RLE-1) can be regularized by means of a conditional assignment as discussed in section 4.2.1. This has been done at line 13 of the version RLE-2 (Fig. 4.2), in which we now focus.

So far, no significant restructuring of the algorithm has been necessary. Unfortunately, the core difficulties stand in the if statement at line 11 of RLE-2. We have seen that in order to regularize an if statement we have to (schematically) execute (unconditionally) both branches and then recombine their effects, using conditional assignments, consistently with the *encrypted* result of the evaluation of the condition. In particular, this means that going through either branches of an if must be indistinguishable in terms

of effect [3]. For the if in question, the effect is to print something when the condition is satisfied and not to print anything otherwise (i.e., different effects on the program output). Hence, in order to have a similar effect whatever the branch we have no other choices than either :

— systematically print nothing (which is not very useful) ; or
— systematically print something but ensuring that the kind of "something" which is printed when there is nothing to print is indistinguishable from the kind of "something" which is printed when there is something to print.

This latter property is given us by the (semantic) security [4] of the underlying cryptosystem. For example, we could print an (encrypted) zero for the counter value followed by an arbitrary symbol (this would be transparent to the RLE decoder). This would mean replacing the if in question (lines 11 and 12 of RLE-2) by a systematic ouput of both (input[i] !=input[i-1] ?j :0) and input[i-1].

Of course, the issue is that (by construction) we do not compress anything anymore as the output stream is now systematically twice larger than the input one ! As such, none of the above options is thus satisfactory.

4.2.3 Regularization of the algorithm outputs

Starting from the version given by RLE-2, let us now derive a meaningful "FHE-friendly" variant of the algorithm, which is described in Fig. 4.2-RLE-3.

In order to compress, we have to make the algorithm work with an *a priori given* compression rate and, as a consequence, relax another characteristic of the algorithm : its conservativeness. In order to do so, we have to work with a constant number of pairs counter/symbol both during the inner working of the algorithm as well as in its outputs. Then, the resulting variant (Fig. 4.2-RLE-3) needs an additional parameter, n_pairs, which allows to fix the number of (encrypted) pairs which are to be systematically outputted by the algorithm. All of these pairs are initialized to 0 (for the counter) and an arbitrary symbol (lines 13 to 16 of RLE-3). Then the compression step is run (described just after) and the algorithm prints all the n_pairs (encrypted) pairs counter/symbol. Then, regardless of its input, the algorithm always outputs the same number of (encrypted, indistinguishable from one another) pairs. Two scenarios are then possible :

1. the a priori fixed number of pairs is sufficient to represent the input sequence in which case the reconstruction of the sequence by the decoder will be correct but it can be that a number of pairs were unused (0 counter) and then that we have compressed less than possible (with RLE),
2. otherwise, the decoder will not be able to reconstruct the correct input sequence and some form of padding will be required for the decoder to at least output a sequence of correct length.

Some compression is achieved when

$$(\alpha + \beta) \times \text{n\_pairs} < \alpha \times \text{n\_chars}$$

where α and β are the number of bits for respectively representing the symbols and the counters.

The compression step (lines 17 to 29 of RLE-3) works as follows. Integer k is initially set to 0 (or, more precisely, to an encryption of 0), this counter points to the current working pair in the pairs buffer. Then, we execute the loop on the input sequence symbols

3. Whether we want it or not the "API" of the homomorphic machine and its underlying encryption system does not, by definition, let us do otherwise.

4. Recall that semantic security is a well known property of probabilistic encryption schemes which in particular ensures that two encryptions of the same message are (computationally) indistinguishable from the encryptions of two different messages.

(line 18). The for loop of line 19 corresponds to an array assignment operator *with an encrypted index* (it is a loop because the complexity of such an assignment is linear in the array length as discussed in Sect. 4.2.1). Functionally, this loop is equivalent to doing output_ctr[k]=j and output_chr[k]=input[i-1] but with an encrypted k. Then, k and j are respectively incremented on symbol change and non-change by means of conditional assignments (lines 23 and 24). Lastly, once the loop of line 18 is done, the loop at line 26 is another (the last) assignment to the pairs buffer (still with encrypted k), this is the "equivalent" of the last print instruction of RLE-2 (line 15).

As an example, in order to avoid that the need for padding has an impact on the decoder, it is possible, still within the constraints of an homomorphic execution, to (i) count the number of symbols ℓ that is accounted for the output pairs (i.e., to sum, in the encrypted domain, the entries in output_ctr) and (ii) update the last counter (i.e., output_ctr[n_pairs-1]) in order to repeat the last symbol of the input sequence n_chars$-\ell$ more times. That way, it is the encoder which drives the padding which has sometimes to be done by the decoder. Other more complex strategies are of course possible, but they will remain lossy. It is also possible to add a flag to the compressed stream, indicating whether or not compression was lossless, and letting the decoder taking appropriate actions in either case. Of course, a simple sufficient condition for the compression to have been lossless is when the last counter is 0. This condition is most likely good enough for all practical purposes.

4.2.4 A more general result

We are now in a position to state a more general negative result which forbids any form of lossless compression in the FHE setting. Indeed, as long as a compression algorithm admits some difficulties to compress some its *possible* input sequences, then the associated output to such a sequence is not guaranteed to fit within an a priori fixed memory budget smaller than the input length. From a theoretical viewpoint, we know that there exists sequences which are not compressible for all compression algorithms | in fact almost all sequences are so [LV09]. It follows that for any compression algorithm, when run homomorphically, either we (always) do not compress (i.e. achieve the worst-case compression ratio) and we are lossless or we compress (at an a priori fixed rate) and we are lossy !

This state of affairs is quite intuitive as a FHE run of an algorithm *by construction* masks all the data dependant features of the execution. This is so for the computation time and, as discussed in Sect. 4.2.1, that explain why any algorithm always realizes its worst-case complexity when run homomorphically, as well as for any (other) of the algorithm outputs. In essence, revealing the extent to which an (encrypted) input sequence is compressed by a given algorithm (i.e. the length of the output sequence) would leak a lot of information on the input sequence itself.

Of course, by fixing an a priori memory budget for the compressed sequences some a priori information is revealed on the probability distribution of the inputs expected by the system designer (i.e. this would tautologically reduce the search space for an attacker to input sequences which are compressed by the algorithm in question within the budget in question [5]). Still, as an example, the designer of a video compression scheme may even accept to change that memory budget during the encrypted-domain execution of a video coder thereby hinting at the kind of scene being compressed at a given time. Whether or not this kind of a priori knowledge disclosure is acceptable depends on the application and the overall threat model.

5. With respect to the universal compressor [LV09], fixing a budget smaller than the input length reveals a threshold on the expected algorithmic complexity (i.e. degree of randomness) of the input.

4.3 Experimental results

In the previous section, we have defined a variant of the RLE algorithm which can *in principle* be run homomorphically. Having done so, we are only half-way and we now turn to the issue of concrete refactoring and optimization of this variant to make it run homomorphically *as efficiently as possible*.

All the results in this paper have been obtained using the Fan-Vercauteren homomorphic scheme [FV12] (FV for short) and the parameters were computed as described by the authors of that paper. The scheme was dimensioned to assure 128-bits of security and the minimal multiplicative depth needed by the executed boolean circuits.

Again, in this section, we have opted for a tutorial-style presentation in order to share the concrete return on experiment we gained in doing the job of porting RLE.

4.3.1 The Armadillo compiler

In our experiments, we have used the Armadillo FHE compiler [6] [CDS15]. The Armadillo compilation chain is an easy to use environment which turns C++ programs into optimized boolean circuits which can then be executed homomorphically with the Armadillo FHE runtime environment. The compilation chain is composed of 3 layers : a front-end, a middle-end and a back-end. The front-end transforms code written in C++ into a boolean circuit representation. The middle-end layer optimizes the boolean circuit produced by the front-end using ABC [7] tuned for FHE-oriented circuit optimizations. The back-end then either constructs a binary (linked with a concrete FHE library) which homomorphically executes the boolean circuit or relies on the Armadillo RTE (which is essentially a FHE boolean circuit interpreter) to do so. In both cases, OpenMP is intensively used which allows to obtain almost linear speedups on SMP machine (say up to 100-200 cores). Two HE are presently supported by Armadillo : (i) an in-house implementation of [FV12] and (ii) the publicly available library HElib [HS14, HS15a]. The constraints of the FHE machine (Sect. 4.2.1) are enforced via the programming model which is exposed by the compiler (C++ types supporting only those operators compliant with the FHE constraints).

The Armadillo compiler is very useful when it comes to prototyping a given algorithm for homomorphic execution, allowing the programmer to have a short feedback loop when trying FHE-oriented optimizations.

4.3.2 A first trial

Before starting to run the RLE-3 variant (Fig. 4.2-RLE-3) which is not a "simple" algorithm (with respect to present homomorphic standards), we wanted to start with something a little bit more elementary. So our first test consisted in, given an input sequence of symbols, to output a sequence of booleans flagging the symbol changes. With the Armadillo compiler, we write the following seemingly standard code :

```
Integer8 input[NUM_OF_CHARS];
for(int i=0;i<NUM_OF_CHARS;i++)
  cin>>input[i];
for(int i=0;i<NUM_OF_CHARS-1;i++)
  cout<<(input[i]==input[i+1]);
```

Above, Integer8 is a type provided by the compiler environment for declaring encrypted integer variables, here on 8 bits. The NUM_OF_CHARS constant defines the number of

6. At present, the Armadillo compiler can be made available as part of research collaborations.

7. ABC (www.eecs.berkeley.edu/alanmi/abc) is a well-known open source System for Sequential Synthesis and Verification.

symbols in the input stream. It should be emphasized that this latter constant must be known at compile-time in order to be able to build a (by definition static) boolean circuit. Note also that, in practice, it is not necessarily an issue. For example, in the context of a video coder, the RLE algorithm is executed only on a few (standardized) macro-block sizes (e.g. 4x4, 8x8, 16x8, 8x16, 16x16). Thanks to C++ operator overloading capabilities, our Integer8 variables are manipulated like usual integers. For example, the >> operator expresses a reading operation (in fine of encrypted bits stored in files which location will be provided to the runtime environment), the $==$ operator allows to test the equality of two variables of type Integer8 (the result being and *encrypted* boolean) and the << operator will trigger the ouput of the Integer8 variables (in fine of encrypted bits stored in files which location will be provided to the RTE). Of course, there are many more operators, but our point is not to describe them exhaustively in this paper.

Letting the above program through the compiler gives a multiplicative depth 3, independently of the input sequence length (ABC is not able to optimize it further). This is in line with our expectations since this depth is due only to the $==$ operator, implemented in Armadillo as an arborescent product of the bits $x_i \oplus y_i \oplus 1$ of x and y (depth 3 for 8 bits, then). Running this program with FV ($\lambda \approx 128$ and depth 3) is done in 0.270 s on the average 8 cores machine used throughout this paper unless otherwise stated. This corresponds to 1.726 s of *cumulated* computing time over the 8 cores of the machine (hence a speedup of 6 was obtained by the compiler *transparently* to the programmer), a duration which is dominated (without surprise) by the execution of the 63 homomorphic multiplications (AND gates) in fine required to execute the algorithm.

After this first test, let us move on to RLE-3.

4.3.3 Porting RLE-3

In order to start, we have simply rewritten RLE-3 (as in Fig. 4.2-RLE-3) using the API of Armadillo. Essentially, this means that we have declared all the supposedly encrypted variables in RLE-3 as Integer8 (that is input[NUM_OF_CHARS], j, k as well as output_chr[NUM_OF_PAIRS] and output_ctr[NUM_OF_PAIRS]) and that we have used the construction select(c,a,b) in lieu of c ?a :b as the ? : construct cannot be overloaded in C++.

This time, it is not only the number of input symbols (NUM_OF_CHARS), but also the number of output pairs (NUM_OF_PAIRS) which must be a priori known. As discussed in Sect. 4.2, this is an intrinsic constraint as we can compress in the encrypted domain but only with a fixed rate. As an example, for 10 symbols and 5 pairs, passing that program through Armadillo gives a multiplicative depth of 22 | which does not appear prohibitive | but as we shall later see, this depth is not independent of the input sequence length. Still, in the same conditions as in the preceding section, the resulting program runs in around 3 min 41 s (taking into account a speedup of 7.75 (which is almost optimal for an 8 cores machine). This running time is dominated (99.7%) by the execution of the 1323 AND gates in the resulting boolean circuit. So far so good, but scaling to longer input sequences requires more work.

Before turning to the optimizations which will allow us to better scale to longer input sequences, let us first illustrate one of the benefits in using the Armadillo compiler. In order to do so, let us replace the following code (equivalent to lines 23 and 24 of RLE-3) :

```
k=select(input[i]==input[i-1],k,k+1);
j=select(input[i]==input[i-1],j+1,
  Integer8(1));
```

by

```
Integer8::Bit b=input[i]==input[i-1];
```

```
k=select(b,k,k+1);
j=select(b,j+1,Integer8(1));
```

In the second above code snippet, the (FHE) programmer explicitly avoids to evaluate twice input[i]==input[i-1]. Still, when programming in C or C++ (with optimization options activated) that same programmer would expect this kind of elementary optimizations to be taken care of automatically by the compiler (and to be able to preserve the first of the above two snippets which is easier to read). As a far as the Armadillo compiler chain is concerned, this is indeed the case. In the first case, the compiler front-end outputs a larger boolean circuit, but *the same circuit* as in the second case is obtained once the ABC optimization step is applied.

4.3.4 Multiplicative depth analysis

Another way of benefiting from a compiler infrastructure is that it eases the characterization of programs or algorithms. Using Armadillo, we have thus attempted to get some insights on the behavior of the multiplicative depth of RLE in terms of both the input sequence length as well as the number of output pairs. Table 4.2 summarizes our results from a "toy example" with 10 symbols and 5 pairs up to a first "realistic" example with a 64 symbols input sequence compressed onto 8 pairs. The latter would correspond to e.g., the compression of an 8×8 macro-block (i.e., 64 bytes) with a 25% rate (8×2 bytes). As such, it appears that the multiplicative depth is essentially driven by the input sequence length and that, running RLE-3 (in its raw form) on a sequence of more than 20 symbols appears irrealistic (we consider, from our experience, that a multiplicative depth of 30 is the threshold above which an homomorphic execution becomes prohibitive in time and memory). So we need to find ways of smashing that multiplicative depth down.

TABLEAU 4.2 – Sampling the multiplicative depth of raw RLE-3 (basic ABC optimization included). Each line gives for a given number of output pairs, the evolution of the multiplicative depth in terms of the input sequence length.

	10	20	30	40	50	60	64
1	24	34	44	54	64	74	78
2	21	31	43	53	61	71	75
5	22	32	42	52	62	72	76
8	21	31	41	51	61	73	75

4.3.5 Optimization of the incrementation of k and j

Let us now focus on the following two lines :

```
(1) k=select(input[i]==input[i-1],k,k+1);
(2) j=select(input[i]==input[i-1],j+1,
      Integer8(1));
```

where input[i]==input[i-1] is at multiplicative depth 3 (Sect. 4.3.2).

If we then start to work with a program which computes (and outputs) the final value of k, then we indeed explain part of the multiplicative depth of our algorithm as we have a depth of 17 for an input sequence of length 10 and a depth of 67 for an input sequence of length 60. In the former case, the homomorphic running time is of around 20 secs on 8 cores.

The main issue with line (1) above is that there is a cumulative effect on k in the sense that the next value for k will be either the preceding value of k (or that preceding value incremented by 1) multiplied by an encrypted bit (the result of the condition evaluation,

at depth 3). Hence, we accumulate 3 levels of multiplicative depth per loop iteration (except at the beginning since, as k is initialized to 0, a public constant, the first loop iteration will involve some hybridized cleartext/ciphertext calculations [8]).

In order to "break" this (multiplicative depth) accumulation, we can then rewrite our line as :

```
k+=input[i]!=input[i-1];
```

Although there still is an accumulation, of course, but this accumulation now occurs through an 8-bits adder and this does not result in an accumulation w.r.t. the multiplicative depth [9]. Having done so, we end up with a depth of 10 ($=3+7$ the sum of the depths of a comparison operator and of an adder on 8 bits) and the running time of the program computing only k gets down to 4 secs. Additionally, if we report this modification in RLE-3, the overall multiplicative depth (for an input sequence of length 10) gets down to 18 and the algorithm runs 1.7 times faster.

Line (2) above is more difficult to handle. Indeed, as for k, a program which computes and outputs the last value of j has a multiplicative depth dependent on the input sequence length (e.g., 17 for 10 symbols and 67 for 60) and, if the technique we used for k would allow to handle the incrementation part of j update, its conditional reset cannot be dealt with in the same way.

Let us first start by replacing line (2) above by :

```
j=Integer8(1)+j&(input[i]==input[i-1]);
```

which leads to a slightly better depth (without, however, breaking the dependency on the input sequence length).

Then, the leap of faith consists in considering that it may be interesting to perform *many* redundant computations (albeit with the hope of doing them at much lower depth) and look at the successive values of j produced during the execution of the loop.

Let

$$\begin{aligned} rClb_1 &= \mathsf{input[1]==input[0]}, \\ &\ldots, \\ b_i &= \mathsf{input[i]==input[i\text{-}1]}, \\ &\ldots \end{aligned}$$

Let also $j_0 = 1$. We then have,

$$\begin{aligned} lClj_1 &= 1 + j_0 b_1 = 1 + b_1 \\ j_2 &= 1 + j_1 b_2 = 1 + (1 + b_1) b_2 = 1 + b_2 + b_1 b_2 \\ j_3 &= \ldots \end{aligned}$$

And, more generally,

$$j_i = 1 + \sum_{l=1}^{i} \prod_{m=l}^{i} b_m,$$

with term of highest degree (i),

$$\prod_{m=1}^{i} b_m.$$

8. I.e. using simplified homomorphic operations when one of the operand is in clear form as : $[x]+0=[x]$, $[x]+1=[x]+[1]$, $[x]\times 0=0$ and $[x]\times 1=[x]$.

9. Writing down the equations of an n-bit adder reveals an invariant w.r.t. the multiplicative depth : summing two "fresh" inputs leads to a result for which the i-th power bit is at multiplicative depth i, and summing two numbers with that property lead to a results also with that property. As a result, we can additively accumulate over $\mathbb{Z}_2^n$ with a multiplicative depth independent of the number of (encrypted) values which are summed.

This latter term may then be evaluated with depth $\log_2 i$ by means of an arborescent product.

Hence, if we accept to proceed with a number of redundant multiplications, the calculation of j_i can be done with depth $3+7+\log_2 i$ since we need to account for the depth of the b_i's (3) as well as that of the 8-bits adder (7).

When we observe the previous computation of k and j, it is noted that by carrying out the additions like this : k + =input[i] != input[i-1] and j + = $\prod_{m=l}^{i} b_m$; this incites us to use directly the 8-bits adder while we handle the binary values for the first time. The use of this 8-bits adder increases systematically the depth of 7. To avoid this, we will change this manner to make addition, and replace it by *arborescent addition*. This new way of adding will allow us to use an adder of $\log_2(i)$ instead of 8-bits. Thanks to this technique, we can reduce the depth of k from 10 to as marked in the Table 4.3. With the similarly modification in the computation of j, its multiplicative depth decreases (from $3+7+\log_2(i)$) as shown in the Table 4.3.

TABLEAU 4.3 – Reduction of multiplication depht in the computation of k and j using the arborescent addition like method.

i	$2 < i < 66$	$i \geq 66$
depth(k)	$4+\log_2(i)$	10
depht(j)	$4+\log_2(i)+\log_2(i-2)$	$10+\log_2(i)$

Thanks to this optimization, the depth of the program *computing only k and j* decreases from ≈ 20 down to 11 (for an input sequence of length 10) and from ≈ 70 down to only 16 for a more realistic input sequence length of 64 (corresponding, as said, to an 8×8 macroblock). The amount of computation redundancy could be further optimized.

So far, we have thus shown that a program for computing the last value of k and j is amenable to a reasonably low multiplicative depth. Although the impact of integrating these optimizations within the whole RLE algorithm is not negligible (execution time decreases from 2 min 9 s down to 1 min 8 s in the same conditions as before), the overall depth of the algorithm still remains problematic due to the output_chr and output_ctr array assignments (with encrypted index k) which are the last residual "hotspot".

4.3.6 Optimization of the output arrays assignments

After working on the computation of j for optimization of the RLE algorithm, time has come to focus on the following loop :

```
for(int l=0;l<NUM_OF_PAIRS;l++)
{
     output_chr[l]=select(k==l,input[i-1],
     output_chr[l]);
     output_ctr[l]=select(k==l,j,
     output_ctr[l]);
}
```

which updates both the character table output_chr and the corresponding index counter table output_ctr.

To optimize this loop, we were inspired from the approach used for the previous calculation of j : namely, develop the line in the loop to express them as the functions who depend of the boolean factor l==k and the inputs. Before we present the step of our development, we observe that (1) the two instructions of this loop (output_chr[l] and output_ctr[l]) follow the same pattern ; we can generalize their writing using the generic

expression defined by equation 4.1, (2) the variation of temporary counter k depends of the incrementation of i ; then in the following, we replace it by $k^{(i)}$. We have then :

$$c_l^{(i)} = select\left(l == k^{(i)}, x_{i-1}, c_l^{(i-1)}\right) \tag{4.1}$$

where the counter $k^{(i)}$ is define by ;

$$k^{(i)} = select\left(input[i] == input[i-1], k^{(i)}, k^{(i)}+1\right), \tag{4.2}$$

with i in range [1;...;NUM_OF_CHARS], l in range [1;...;NUM_OF_PAIRS], and the initial values are $c_l^{(0)} = c_0$, $k^{(0)} = 0$.

For all values of l we define $b_l^{(i)} = \left(l == k^{(i)}\right)$ which denote the encrypted bit of the comparaison l==k at the iteration i. Equation (4.1) can be expanded as follow :

$$\begin{aligned} rClc_l^{(i)} &= b_l^{(i)} x_{i-1} + \left(1 + b_l^{(i)}\right) c_l^{(i-1)} \\ &= c_l^{(i-1)} + b_l^{(i)} \left(x_{i-1} + c_l^{(i-1)}\right). \end{aligned}$$

By recursively expanding this expression, and then factoring, we get the following general term,

$$\begin{aligned} rClc_l^{(i)} &= c_0 + (c_0 + x_{i-1})\, b_l^{(i)} \\ &\quad + \sum_{j=1}^{i-1} \left(c_0 + x_{j-1}\right) b_l^{(j)} \left(1 + \sum_{\mathcal{K} \in \mathcal{P}(j+1,\ldots,i)} \left(\prod_{u \in \mathcal{K}} b_l^{(u)}\right)\right), \end{aligned}$$

where $\mathcal{P}\left(j+1,\ldots,i\right)$ denote the power set of set $\{j+1,\ldots,i\}$.

With this last equation, we can note that the term of highest degree is the term for which $j = 1$ and $\mathcal{K} = \{2,\ldots,i\}$, or, in other words the term :

$$(c_0 + x_0) \prod_{u=1}^{i} b_l^{(u)}.$$

This term can be evaluated also with a multiplicative depth of $\text{depth}(b_l^{(i)}) + \log_2(i+1) = 12 + \log_2(i+1)$ if we carry out the multiplication in a tree-like manner as before. Using the last equation, we observe that we can obtain the values of output_chr[l] (respectively output_ctr[l])by replacing "c_0" by "1" (respectively by "0") and "x_i" by "input[i]" (respectively by "j_i") for all i. The advantage of this formula is that we can reuse the results of the computations of the products $\prod_u b_l^{(u)}$ to compute both output_chr[l] and output_ctr[l].

4.4 Towards automatic multiplicative depth optimization

4.4.1 Heuristic boolean circuit rewriting

We have seen previously that there is a huge amount of work needed for by-hand optimization of non-straightforward boolean circuits multiplicative depth. As one can expect this will be the case for many other applications and not only the RLE algorithm. In this section, we report the result obtained when using RLE (and the previous hand-crafted results) as a testbed for a multiplicative depth minimization heuristic which we are developing now as part of the Armadillo middle-end [CAS17]. Let us define the *direct multiplicative depth* of a node in a boolean circuit as the length of the longest path starting from circuit inputs to this node. Equivalently, the *reverse multiplicative depth* is the length of the longest path from this node to circuit outputs. The nodes for which these two values coincide are called *critical nodes*. The *critical circuit* contains all the critical nodes of a boolean circuit.

It is straightforward to see that optimizing critical circuit paths allows to minimize the overall multiplicative depth of a boolean circuit. We introduce a rewrite operator which when applied to the critical circuit will potentially decrease the original circuit multiplicative depth. The idea behind this operator is to rewrite critical paths of multiplicative depth 2 in such a way that its multiplicative depth decreases. A critical path of multiplicative depth 2 is a path in the critical circuit which starts and ends with AND gates and contains zero or more XOR gates in between [10]. Let $((((a_1 \& a_2) \oplus b_1) \oplus \ldots) \oplus b_m) \& a_3$ be the formula for a critical path of multiplicative depth 2 with m intermediate XOR gates. Using XOR gate distributivity and AND gate associativity rules this path can be rewritten as $(((a_1 \& (a_2 \& a_3)) \oplus (b_1 \& a_3)) \oplus \ldots) \oplus (b_m \& a_3)$. Under certain conditions on the multiplicative depth of nodes a_i the global multiplicative depth of the circuit will decrease.

The multiplicative depth minimization heuristic chooses, using a simple priority based on path length (shorter paths are privileged), a path of multiplicative depth 2 and rewrites it. The heuristic stops when no more critical paths, which minimize the global multiplicative depth, are available.

4.4.2 Experimental results

Here, our objective was to perform a test of running RLE-3 with a compression rate of 25% on a sequence of 64 values, representing an 8×8 macroblock. Thus with a budget of 16 bytes, which corresponds to 8 pairs symbol/counter.

Running our heuristic on RLE-3 circuit leads to the following results. We start from a circuit of depth ≈ 70 and ≈ 12000 AND gates to get to a circuit of depth 23 but with ≈ 60000 AND gates. The multiplicative depth has therefore been smashed down but at the cost of almost 5 times more AND gates. Still, this is cost-effective : when we were simply not able to execute the initial depth-70 circuit, we have been able to execute the new depth-23 circuit in around 30 mins (2003 secs) on a 48 cores computer (taking into account a speedup of 47.3, yet again almost linear and obtained transparently through the use of Armadillo).

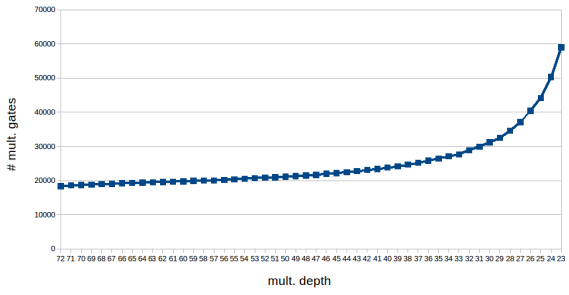

FIGURE 4.3 – Multiplicative depth vs number of AND gates during the execution of the circuit rewriting heuristic. See text.

Fig. 4.3 provides the evolution of the depth and the number of AND gates during the successive iterations of the rewriting heuristic. Of course, if for some algorithm the optimum performances are not reached for the minimum multiplicative depth (and it experimentally appears to maximize the number of AND gates), the intermediary circuit which realizes the best multiplicative depth/AND gates trade-off can be kept.

10. Without loss of generality we suppose that the boolean circuits contain only AND and XOR gates.

Conclusion

In this paper, we have tried to tackle the issue of running a compression algorithm over encrypted data, using homomorphic encryption. As already pointed at, running compression algorithms in the encrypted domain is an important topic for the longer-term applications of FHE techniques. In fact, stream transcoding in the encrypted domain would, if practical, unleash a number of very interesting applications. In this work, and although we cannot claim to have achieved practical performances, we have tried to avoid sweeping under the rug the intricacies we had to effectively deal with in order to regularize and optimize the algorithm in order to be able to run it in the encrypted domain. We have done this with the hope that our return on experiment may be useful to other "FHE programmers".

Chapitre 5

Towards video compression in the encrypted domain : a case-study on the H264 and HEVC macroblock processing pipeline

Sommaire

5.1 Introduction

Homomorphic encryption is an advanced cryptographic tool which allows to compute arbitrary algebraic functions on an encrypted input and obtain a result whose decryption is the result of this algebraic function applied on the unencrypted input. In 1978, Rivest and al. [RAD78] introduced the idea of homomorphic encryption as *privacy homomorphisms*. The scheme that they proposed was additively homomorphic and, unfortunately, was easily broken by Brickell and Yacobi [BY87] a few years later. The idea of a Fully Homomorphic Encryption (FHE) remained speculative for more than 30 years.

The first theoretical construction of an encryption scheme able to perform both homomorphic addition and multiplication operations in an arbitrary manner came from the work of Craig Gentry in 2009 [Gen09a]. To obtain a FHE scheme, Gentry's initial idea has been to use a somewhat homomorphic scheme (SHE) i.e., a scheme capable to perform a limited number of homomorphic additions and multiplications, and to associate it with what he has called *bootstrapping*. Indeed, SHE schemes necessarily introduce noise at the time of encryption, this noise increases in the ciphertext with operations to the point where the latter cannot be decrypted correctly. Bootstrapping solves this issue by allowing to "refresh" the ciphertext by reducing the noise back to a constant small-enough amplitude, thus allowing further homomorphic calculations. Starting from this technique, several SHE schemes have emerged, and among them we can mention the most know ones as BGV [BGV12] that can be found in the `HElib` library [HS15b] and FV [FV12] found in `SEAL` [CLP17a] and in `Cingulata` [SCS15, LIS]. We can also mention the scheme TFHE [CGGI16] found in the library of the same name [CGGI17b].

On homomorphic encryption mechanism. The implementation of these algorithms is carried out according to two main types of techniques : the first one (called Type I in this chapter) consists in working with a plaintext equal to $\mathbb{Z}_2$ and makes it possible to easily transform a binary operation in the clear domain to the encrypted one ; the second technique (called Type II) uses $\mathbb{Z}_p$ (for an integer $p > 2$) as plaintext which trades off better execution times for more limited functionalities in relation to the clear domain. In this chapter, our first result is to provide a detailed comparative study of these two types of FHE usage.

On image/video compression. To help us, we confront both approaches through the (encrypted-domain) execution of the compression pipeline of a video/image. More precisely, we study both the H264 and HEVC macroblock processing pipelines and their underlying algorithms : (i) color space conversion RGB to and from YCrCb, (ii) Discrete Cosine Transform (DCT) and (iii) quantization. For each of them, we study and compare the different implementation strategies for Type I and Type II FHE schemes and finally compare the final result when using the best strategy of each. We think that this case study is a good choice for studying homomorphic encryption schemes since it includes a number of well-defined algorithms of moderate complexity. It also corresponds to a real-world need since it may permit to manipulate images and videos by some non-trusted entities, in an encrypted way. Such tool can be very useful to protect sensitive data, and to protect the personal data (and thus the privacy) of individuals. In fact, our study gives a precise idea of which mechanism to use if we want to make the compression of image or video in the homomorphic domain.

5.1.1 Related work

Related work on homomorphic encryption scheme comparison. Some comparison work between certain FHE cryptosystems have already been done in the past. In particular, Le-

point and Naehrig [LN14] propose a comparison between the FV and YASHE schemes by using them both to (homomorphically) evaluate the symmetric block cipher SIMON. Their aim was to give meaningful insights into which scheme to use according to the desired application. For this purpose, they study the parameters to be used in each case and provide a C implementation of both schemes. In [CS16b], Costache and Smart provide a comparison between NTRU/YASHE and BGV/FV. The way they analyze both families is quite similar to the strategy used in [LN14], but they consider more variants (especially in relation to key and modulus switching), they do not restrict to characteristic two and they implement both schemes using the same API and the same optimizations. We here consider a very different approach by studying the way a given algorithm executed in the encrypted domain can be derived depending on the input SHE scheme (and its main properties). We do not re-implement those schemes but consider existing libraries, namely `Cingulata`, `SEAL` and `TFHE`.

Related work on video compression in the encrypted domain. Encrypted image compression is a problem that has also been addressed in the past, mainly in the medical field, for the protection of medical images to be stored on a remote server. With that respect, we can cite the work in [ZH13] which uses secret key encryption to encrypt images and resize them in the discrete cosine domain. We can also mention the work in [YGA+15] which uses a modified version of BGV to perform some functions on the images. In [CCNS17], the authors analyse the elementary Run-Length Encoding (RLE) compression algorithm using the `Cingulata` implementation of the FV scheme. As far as we know, no previous work has attempted to implement a complete version of real-world image compression algorithms, namely H264 and HEVC, when the data are encrypted.

5.1.2 Our contributions

As a summary, we investigate in this work the execution of a macroblock compression pipeline on encrypted data using two approaches : one, called Type I, with an FHE over $\mathbb{Z}_2$ (i.e. with bit-by-bit encryption) and another, called Type II, with an FHE over $\mathbb{Z}_p$ (where $p > 2$ is chosen large enough). For our case study, we take both H264 and HEVC compression algorithms and give details on their three main steps : (i) the conversion of RGB based original image to a YCrCb representation, (ii) the Discrete Cosine Transform (DCT) to transform the representation of block's coefficients into the frequency domain and (iii) the quantization to finally achieve (lossy) compression. We recall the way each step is usually performed, and then give details on the way it should be done in the encrypted domain when using either a Type I FHE or a Type II FHE. We finally compare the results we obtain in each case. As a proof that FHE is today quite close to a real maturity in the image/video compression domain, we apply our results to the H.264 and HEVC compression of a true 128×128 image and show that the performances and resulting images are quite good.

Our contribution is then both a complete study on the way such compression algorithms can be almost practically implemented in the encrypted domain, and a full comparison on the way to manipulate FHE schemes over $\mathbb{Z}_2$ and over $\mathbb{Z}_p$ ($p > 2$) in practice. We believe that both insights, of independent interest, will be in the future useful for homomorphic encryption real deployment in privacy-preserving services.

5.1.3 Organization of the chapter

This chapter is organized as follows. In Section 5.2, we give some preliminaries on both compression pipeline and homomorphic encryption. Section 5.3 compares the two

approaches (Type I and Type II) for homomorphic encryption, in terms of plaintext representation, ciphertext representation, and possible gate operations. In Section 5.4, we details our approaches for the execution of conversion, DCT and quantization in the encrypted domain. Section 5.5 is devoted to the implementation and show our practical results. We finally conclude in Section 5.5.

5.2 Preliminaries

5.2.1 Notation

Throughout this chapter, we use the following notation. $\mathbb{Z}$ is the ring of integers. For an integer $p > 0$, $\mathbb{Z}_p$ is the ring of integers modulo p. We note R to refer to the ring of polynomials $\frac{\mathbb{Z}[x]}{\Phi}$ modulo an irreducible polynomial monic Φ. In practice, one often chooses Φ to be the m-th cyclotomic polynomial, with m a power of 2. For an integer p, $\mathrm{R}_p = \frac{\mathbb{Z}[x]}{(\Phi,p)}$ is the ring of polynomials modulo Φ with coefficients in $\mathbb{Z}_p$. $\mathbb{T}$ refers to the torus $\mathrm{R}_p = \frac{\mathbb{R}[x]}{(\Phi,1)}$ which is the ring of polynomials modulo polynomial Φ, with coefficients that are equal to $0,1$ or no an integer on real ring $\mathbb{R}$. We note a polynomial and its coefficients in lowercase ; i.e. if $a \in \mathrm{R}_p$, $a = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$. A matrix and its coefficients are represented in bold uppercase ; i.e. if $\mathbf{A}$ is a matrix, its coefficient at row i and column j is $\mathbf{A}_{i,j}$. The transpose of a matrix $\mathbf{A}$ is denoted $\mathbf{A}^{\mathrm{T}}$.

We use $\oplus$ and $\otimes$ to represent the operations XOR and AND respectively between two clear or encrypted bits. When there is no ambiguity, we use $\cdot$ in place of $\otimes$. $\ll$ and $\gg$ represent the left and right shift respectively. We use MSB to indicate the most signifiant bit for both clear and encrypted bit representations (see Type I FHE below). We use $[x]$ to refer to a ciphertext which encrypts the plaintext x. For a rational number a, we note $\lfloor a \rceil$ to mean the integer nearest to a and $\lfloor a \rfloor$ the integer part of a. In the FHE field, one usually consider the (mutliplicative) depth of a circuit as the largest number of mutliplication gates (or gate AND, see below) in an input-to-ouput path in the circuit. Addition gates are generally ignored. We use this definition throughout this work.

5.2.2 Compressor/Decompressor

A digital image is the discretization of a natural image obtained through analog-digital conversion techniques. It is represented by a set of rectangular arrays, called pixels, composed of (vectors of) integers. A natural image that has just been captured usually contains a lot of redundant information. In order to used it later (for storage, distribution, etc.), it is necessary to delete this useless information by means of a *compression* algorithm.

Image and video compression involves applying a well-defined sequence of basic algorithms to an input digital image. Its purpose is to reduce its size and transmission rate without reducing the quality (or in a controlled way). Over the time, several standards for image and video compression have been defined and the most well-known ones are *JPEG*, *MPEG2 to MPEG4* or *H261 to H265/HEVC*. But in fact, all of them share the same set of core algorithms (given in Figure 5.1 for the case of the *JPEG* compression algorithm) :

- the **RGB to YCrCb color space conversion algorithm** which aims at transforming a RGB representation of the color space into a YCrCb representation with less redundancy and, thus, more interesting as a representation for storage and transmission. This conversion is given by a mathematical coordinate transformation ;
- the **Discrete Cosine Transform (DCT)** which is used for its capacity to compact a signal information since this latter can be concentrated into a few low-frequency components of the DCT. As the Discrete Fourier Transform, a DCT is a mathematical decomposition (a sum of sinusoids) of a finite number of regularly sampled

discrete data points ;

— the **quantization** is finally a process permitting to map a set of values from a large set to a set of values in another smaller set.

More details on each step will be given in Section 5.4, which also includes the decompression part, which one corresponds to the inverses of the three same steps, but in a different order, and with some specificities that will be detailed further.

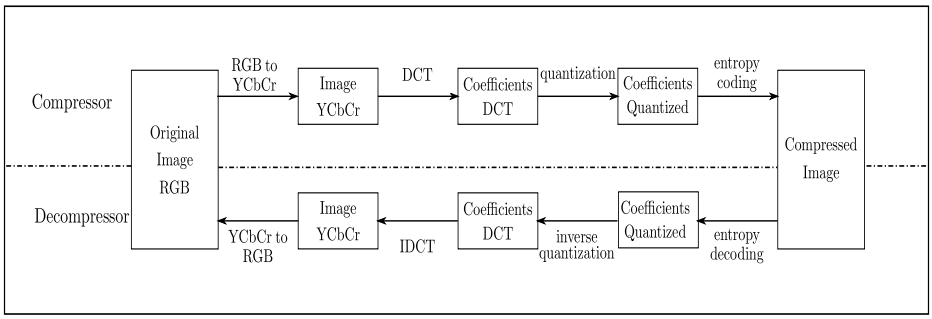

FIGURE 5.1 – Model of compressor/decompressor pipeline of type jpeg

In this chapter, our objective is to work on those three algorithms in the encrypted domain. For that purpose, we have to describe each of them so as to be compatible with FHE schemes.

5.2.3 FHE with binary/digital plaintext

An homomorphic encryption (HE) scheme consists in the specification of four algorithms KeyGen, Enc, Dec, Eval defined as follows :

— KeyGen takes as input the security parameter λ and returns a secret key sk, a public key k and an evaluation key evk ;

— Enc takes as input a message m from the plaintext space $\mathcal{M}$, a public key pk and returns a ciphertext ct which encrypts m ;

— Dec takes as input a ciphertext ct which encrypts m under pk, a secret key sk associated to pk and returns the plaintext m ;

— Eval takes as input a circuit $\mathscr{C}$ with n ciphertexts $ct_1, ct_2, \ldots, ct_n$ that respectively encrypt $m_1, m_2, \ldots, m_n$ under public key pk and the evaluation key evk and returns a ciphertext $ct_{\mathscr{C}}$ which encrypts message $\mathscr{C}(m_1, m_2, \ldots, m_n)$ under pk.

In practice, a circuit is composed of a succession of basic gates and a HE scheme is capable of executing such basic gates in the encrypted domain. Any Turing machine can be described by a succession of additions (XOR gate) and multiplications (AND gate) but there are also some other basic gates that can be used in a circuit (as we will explain in the next section).

In practical SHE constructions, each gate executed in the encrypted domain produces new ciphertexts with increased noise such that the latter, given on input to the Dec procedure, may give a wrong plaintext if the noise is too importat. This phenomenon is almost negligible for the XOR gate, and much more problematic for the AND one, so that most of the time, only the latter is taken into account. The purpose of the so-called *bootstrapping* operation [Gen09a] is to remove most of this noise, and then perform more operations in the encrypted domain. The notion of (multiplicative) *depth* is then usually used both for (i) the homomorphic encryption scheme to define the number of AND gates that the scheme can manage (before requiring a bootstrapping) and (ii) the studied circuit/gate, associated to a chosen FHE, to define the depth requirement for executing it in the encrypted domain.

An homomorphic encryption scheme is said to be "somewhat" (SHE) if it is able to manage a fixed but limited number of additions and multiplications (as discussed above, it

is mostly the number of chained multiplications which does really matter). A SHE scheme becomes "fully" (FHE), if it is homomorphic enough to support bootstrapping (meaning homomorphic enough to homomorphically execute its own decryption circuit). As most of existing SHE can be transformed into a FHE scheme by this way, we only talk about FHE in the sequel.

A lot of FHE schemes now exist in the literature and the most interesting ones (in regards of both security and efficiency) are today considered to be :

- the Brakerski-Gentry-Vaikuntanathan [BGV12] scheme (**BGV**) which is implemented, with a bootsptrapping mechanism, in `HElib` [HS15b] ;
- the Fan-Vercauteren [FV12] scheme (**FV**), implemented in `SEAL` [CLP17a] and in the compiler toochain `Cingulata` [1] [LIS]. Today, both implementations do not integrate the bootstrapping even if this is feasible for the FV scheme, as shown in [FV12] and [CH18] (related to the `SEAL` library) ;
- the **TFHE** [CGGI17a] scheme which directly includes a bootstrapping and for which an implementation can be found in [CGGI17b].

FHE schemes are also split into several so-called *generations*, defined in function of the algebraic structure on which they are built (integer, lattice, ideal-lattice, torus structure), their capacities (simple SHE, leveled-FHE, SHE with bootstrapping, etc) and also their effectiveness. The two first schemes above belong to the so-called *2nd generation*, and the last one to the *4th generation*.

Our aim is now to compare those different FHE instantiations and implementations and the way to use them for concrete use cases (in our case, image/video compression). A first step is done in the next section.

5.3 Capacities and limitations of main stream FHE class

To evaluate an algorithm in the encrypted domain, we need to transform it into a circuit that is compatible with a FHE scheme. In practice, a circuit can be described as a succession of basic gates that can be evaluated with encrypted inputs using the Eval procedure of the FHE scheme.

Regarding the details of existing FHE implementations (see above), one can remark that there are different strategies on the way to encode the input plaintext. More than that, from one FHE scheme, several instantiations are possible (see the NOTE below). The main point is that, when considering the circuit to be executed in the encrypted domain, the way input data are encoded is an essential information since it influences the way to describe the circuit, and the basic gates to be used. The aim of this section is exactly to describe the different possible strategies.

- **Type I FHE** : the input is encoded bitwise and the encryption then consists in one ciphertext for each bit of the plaintext.
- **Type II FHE** : the input is directly encoded using a polynomial or an integer and, in this case, the ciphertext is directly the encryption of the whole plaintext.

We now present in details these two encoding types and their respective properties. We first give some precisions on the input representation, then on the ciphertext one. We then focus on the possible basic operations that can be performed using such representation. In fact, even if the data is not encrypted, the way to describe an algorithm depends on the way the inputs are encoded. Moreover, as we want to execute such algorithms in the encrypted domain, the different options are closely related to the real feasibility to execute such gates and the efficiency of the resulted execution (as we will see below), in particular regarding the depth's growth.

1. `Cingulata` transforms a C++ program in the boolean circuit and execute it over bitewise encrypted data. It also enables to choose the underground FHE which encrypts input data. The current version is based on the FV FHE scheme.

NOTE : each FHE scheme can be instantiated in different manners, and implementing a FHE scheme necessitates to choose first one possible instantiation. It is easy to remark that a Type II FHE scheme can also be instantiated (and then the implementation) as a Type I FHE. For example, the FV scheme [FV12] is instantiated as a Type I FHE in `Cingulata` [LIS] and as Type II FHE in `SEAL` [CLP17a]. Similarly, the BGV scheme [BGV12] can described in both types, and the `HElib` library [HS15b] has chosen to instantiate it in the Type II case.

5.3.1 Type I : FHE over binary plaintext space

In Type I FHEs, the idea is to reproduce the way a classic microprocessor is working. We here give some more details on the plaintext space and then go the case of the ciphertext. We then redefine some useful gates such that they become compatible with the ciphertext representation of a Type I FHE.

Plaintext space $\mathscr{M}_{\mathrm{I}}$. In existing works, there are three main classes of plaintext spaces for a Type I FHE : (i) the first one is simply $\mathbb{Z}_2$ and is used in the BGV scheme [BGV12], (ii) the second one is defined as a discrete subset of $\mathbb{T}$ which is the case of TFHE [2][CGGI16] and the last one is R_2, e.g., in FV [FV12].

In general, the plaintext space associated to a Type I FHE can simply be seen as the set $\{0,1\}$. Then, the encoding of an input consists in decomposing it bit-by-bit and considering that each of them corresponds to an element in the plaintext space.

Ciphertext representation. Using the bitwise encoding, each bit of the input is then encrypted independently. Thus, the ciphertext related to the input is a set of encrypted bits and the relation between them is done as for the plaintext.

Operations. We now consider a set of gates that are at the basis of most of circuits, and especially those useful in our running example of compression. We consider the way each of them can be performed using a Type I FHE. We then consider two plaintexts a and b and their corresponding ciphertexts denoted, using the bitwise representation, $[a] = [a_n][a_{n-1}]\dots[a_0]$ and $[b] = [b_n][b_{n-1}]\dots[b_0]$.

— Addition : taking on input both ciphertexts $[a]$ and $[b]$, the addition in the encrypted domain is given by the relation

$$[a+b] = [s] \quad = \quad [c_{n+1}][s_n][s_{n-1}]\dots[s_0]$$

where encrypted bits of $[s]$ are defined by the following relation, $[c_i]$ being the carry,

$$\begin{aligned} & [s_0] \quad = [a_0] \oplus [b_0], \quad [c_0] = [a_0] \cdot [b_0]; \\ \text{and} \quad & [s_{i>0}] \quad = [a_i] \oplus [b_i] \oplus [c_{i-1}], \quad [c_{i>0}] = [a_0] \cdot [b_0] \oplus [c_{i-1}]([a_i] \oplus [b_i]). \end{aligned}$$

— Multiplication : using on input $[a]$ and $[b]$, the multiplication can here be constructed in "a pencil and paper way", using some tricks with additions and shifts (see [XCWF16]).

— Scalar multiplication : on input a ciphertext $[a]$ and a scalar $t = t_0 + t_1 2 + \dots t_n 2^n$, the scalar multiplication is done as

$$[t \cdot a] \quad = \quad (t_0 \cdot [a]) + ((t_1 \cdot [a]) \ll 1) + \dots + ((t_n \cdot [a]) \ll (n-1)).$$

2. For this case, we choose $\mathscr{M} = \{0, \frac{1}{2}\}$

— Shift : let $r < n$ and $[x] = [x_r]\,[x_{r-1}]\dots[x_0]$ be a ciphertext, and let m be an integer. The shift operation ($\ll$ or $\gg$) is given by

$$[x \ll m] = [x_r]\,[x_{r-1}]\dots[x_0]\,\underbrace{[0][0]\dots[0]}_{m} \quad \text{for } m \le n-r$$
$$[x \gg m] = \underbrace{[0][0]\dots[0]}_{m}\,[x_r]\,[x_{r-1}]\dots[x_{r-m}] \quad \text{for } m \le r.$$

— Scalar division : for the scalar division, the idea is to obtain $[a/t]$ on input a ciphertext $[a]$ and a scalar $t = t_0 + t_1 2 + \dots t_n 2^n$. A simple approach consists in first multiplying (in the clear domain) $1/t$ by a power of two, say 2^m for a chosen m, obtaining t' and then returning $\left[(a \cdot t') \gg m\right]$ using scalar multiplication and shift gates above. When t is not equal to a power of two, this result is only an approximation of the real scalar division. To get an appropriate precision, one should take a larger value for m.

— Comparison : using bitwise representation, one possible strategy to perform comparison, that is obtain $[a > b]$ on input $[a]$ and $[b]$, is to do

$$[a > b] = \mathrm{MSB}([a-b]), \text{the most significant encrypted bit of } [a-b].$$

One can also find some more sophisticated strategies in the literature [GSV07], essentially constructed for Multi-Party Computation but which can be adapted to the case of homomorphic encryption, but we only consider this one in the sequel, essentially for its simplicity and better efficiency.

— Conditional assignment : finally, the conditional assignment `if` ϵ, `then` a, `else` b, on input an encrypted bit $[\epsilon]$ and two ciphertexts $[a]$ and $[b]$, is given by :

$$[c] = [\epsilon]?[a] : [b] \Leftrightarrow [c] = [\epsilon \cdot a + (1 \oplus \epsilon) \cdot b]$$

which can be executed using the above basic methods.

Using a Type I FHE scheme, all these algorithms can then be efficiently implemented. The corresponding multiplicative depth is then given in Table 5.1 (left part) and they will be used as a basis of compression algorithms in the next section, with performances are given in Section 5.5.

Additional remarks. The advantage of Type I FHEs is that they have no real limitations on what can be executed in the encrypted domain (at least in principle, see below note). Some of them are even very cheap, such as the scalar multiplication by a power of 2 which can be done with a simple left shift.

NOTE : the intrinsic semantic security of a FHE scheme definitely comprise the exact execution of some circuits. This is well-known for the "while" loop but this has also been noted for more complex algorithms, such as in [CCNS17]. In fact, this is only feasible when the algorithm can be described as a finite set of arithmetic instructions.

Remark 1 *With Type I FHE the number of operations (and the multiplicative depth) required to perform a multiplication or a division by a public scalar depends on the scalar value itself. In fact, when a computation mixes encrypted and cleartext inputs, some FHE operations can be simplified (e.g., adding 0 to a ciphertext gives the same ciphertext, as multiplying a ciphertext by 1, etc.).*

5.3.2 Type II : FHEs with polynomial ring over $\mathbb{Z}_{p>2}$

We now consider Type II FHE schemes and again give some details on the plaintext and ciphertext spaces and then give the way to implement basic gates in this case.

	TYPE I		TYPE II	
Operations	**Possible ?**	**Depth**	**Possible ?**	**Depth**
addition	yes	$n-1$	yes	0
Multiplication	yes	$n-1$	yes	1
Scalar Division	yes	see Remark 1	yes	1
Scalar Mult.	yes	see Remark 1	yes	0
Shift	yes	0	yes	1
Comparison	yes	$\log_2 n$	no	-
Cond. Assignment	yes	$\log_2 n$	no	-

TABLEAU 5.1 – Comparison of functional capabilities of Type I and Type II FHEs.

Plaintext space $\mathscr{M}_{\text{II}}$. In Type II FHEs, the plaintext space is either an integer ring modulo $p > 2$, denoted $\mathbb{Z}_p$, or a polynomial ring modulo a cyclotomic polynomial having coefficients in $\mathbb{Z}_p$, denoted R_p. Even if there may be some other mathematical structures falling into this case, these are the two main used ones in FHE schemes. The case of integers has been used in several theoretical papers in the past [DGHV10, CMNT] but, essentially for efficiency reasons, the polynomial representation is today preferred (e.g., in the `HElib` and `SEAL` libraries).

To encode the inputs of an algorithm in this case, we need to transform them (i) into integers in $\mathbb{Z}_p$, thanks for example to first a scalar multiplication by a big integer and then a rounding to the nearest integer, or (ii) into elements in R_p, using for example the method given in [CLP17b, CSVW16a]. In general, this is done by decomposing a given input in some fixed basis B and by defining the plaintext as a polynomial which coefficients are those of the decomposition in that base.

Example 1 *For $\mathrm{B} = 3$ in the `SEAL` library, we have the following decomposition, where n is the degree of the cyclotomic polynomial for the ring R_p*

$$a = 13 = 1 + 3 + 3^2 \quad \rightarrow \quad \textit{plaintext } a(x) = 1 + x + x^2;$$
$$b = \frac{328}{27} = 3^2 + 3 + 3^{-2} + 3^{-3} \quad \rightarrow \quad \textit{plaintext } b(x) = x^2 + x - x^{n-2} - x^{n-3}$$

NOTE : the way to represent a plaintext in $\mathscr{M}_{\text{II}}$ is in practice done in accordance with the homomorphic operations.

Ciphertext representation. The ciphertext corresponding to an input is then obtained by the encryption of the polynomial representing the plaintext. For an input a, the ciphertext is equal to $[a(x)]$.

Operations. From two plaintexts a and b and their corresponding ciphertexts $[a]$ and $[b]$, we again consider the basic gates that have been studied for Type I and consider now the case of Type II.

- Addition : the way to perform an addition depends on used FHE and is defined in accordance with the addition of elements of $\mathbb{Z}_p$ or R_p.
- Multiplication : the way to proceed is similar to the one for addition.
- Scalar multiplication : simple multiplication of the coefficients of the ciphertext by the input scalar.
- Scalar Division : on input a ciphertext $[a]$ and a scalar t, the best idea is to encode the inverse of t and use it as in a scalar multiplication. For example, the computation of $[5/8]$ is done by encoding $5/8 = 2^{-1} + 2^{-3}$ in $x^{n-1} + x^{n-3}$ and then returning $\left[a \cdot (x^{n-1} + x^{n-3})\right]$. Another (less efficient) way to proceed is to encrypt t^{-1} and then multiply it by $[a]$.

— Shift : this operation is not directly provided in Type II FHE (contrary to the Type I). It can however be obtained by scaling the ciphertext by an encoding of a power of 2. For example, to compute $[a \ll m]$, one first encodes 2^m in x^m and then returns $[a \cdot x^m]$. Similarly, for $[a \gg m]$, one encodes 2^{-m} in polynomial x^{n-m} and returns $[a \cdot x^{n-m}]$.
— Comparison : There is no known method for Type II FHE to provide a comparison in the encrypted domain. Some solutions exist for Multi-Party Computation [DGK08] but cannot be transposed into the world of fully homomorphic encryption in which we focus. More than that, the way to provide a test that a given value is or is not equal to 0 using only additions and multiplications over $\mathbb{Z}_p$ seems infeasible !
— Conditional Assignment : similarly, the conditional assignment seems hard to obtain since necessitating to make one addition modulo 2 on the plaintext.

Again, the corresponding multiplicative depth is given in the Table 5.1 (right part).

Additional remarks. Because of the compact form of a polynomial-based input encoding, several operations of a usual processor cannot be performed using a Type II FHE, while this is always possible with a Type I FHE (see Table 5.1). The consequence is that, using a Type II FHE scheme, some circuits cannot be transformed to be executed in the encrypted domain (e.g., a circuit necessitating a comparison or a conditional assignment).

However, this approach presents some major advantages. Looking at Table 5.1, most of basic operations only consume zero or one level of multiplicative depth, while the impact is much more important when using a Type I FHE scheme. This has also an impact on the real execution time of a circuit evaluation in the encrypted domain, as we will show in Section 5.5.

Last but not list, a Type II FHE permits to store all the information about an input into only one single polynomial. This contributes to the reduction of the expansion factor, compare to a Type I FHE.

5.4 Macroblock compression in the encrypted domain

In this section, we present the different algorithms involved in an image compression pipeline at the macroblock level. For each of them, we give some details on how to implement them using both Type I and Type II FHE. As image and video compression is a wide domain, we focus our work on algorithms used in the mainstream video compression standards H.264 and HEVC. In the sequel, we consider colored images rather than grayscale ones.

5.4.1 Color space conversion : RGB to and from YCrCb.

A digital image is made of pixels which are triplets of integers R, G and B. Each integer R, G, B is encoded on the same number of bits (say 8 to 12 bits) and represents the quantity of colors red, green and blue respectively making the pixel color. This representation system is appropriate for display devices, but for compression and image transmission, the YCrCb system is more relevant due to biological reasons (the human eye has different sensitivity to colors and brightness). In a nutshell, the YCrCb system has one component of luminance Y, and two of chrominance blue, Cb, and red, Cr. The luminance contains informations about the pixel brightness, being similar to pixel grayscale in the original image. Chrominance components, on the other hand, contain informations about the pixel color. Using this encoding method, one can reduce the amount of information in chrominance components without the human eye observing a too significant difference in the image. Letting the latter details aside, the conversion from RGB to YCrCb

is a linear transformation defined as follows :

$$\begin{bmatrix} \mathrm{Y} \\ \mathrm{C}b \\ \mathrm{C}r \end{bmatrix} = \begin{bmatrix} ca & (1-ca-cb) & cb \\ \frac{-ca}{2(1-ca)} & \frac{(ca+cb-1)}{2(1-ca)} & \frac{(1-cb)}{2(1-ca)} \\ \frac{1-ca}{2(1-cb)} & \frac{(ca+cb-1)}{2(1-cb)} & \frac{-cb}{2(1-cb)} \end{bmatrix} \cdot \begin{bmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{bmatrix}, \tag{5.1}$$

where the coefficients ca and cb are defined following the biology experience on human eye. In ITU-R recommendation BT.601 [BT95], these coefficients are set to $ca = 0.299$ and $cb = 0.114$. This transformation is invertible and its inverse is also a linear transformation depending of coefficients ca and cb.

After this first step of color space change, each component Y, Cr, Cb is divided into blocks of a few pixels, known as macroblocks, and the subsequent transformations of the compression pipeline are applied independently on each of these blocks. We now turn to implementation details for this algorithm targeting the two types of FHE considered in this chapter.

RGB to and from YCrCb with Type I FHE. The first step towards porting this algorithm for an execution over encrypted data is to transform coefficients of the matrix given in Equation (5.1) into integers representing fixed-precision numbers. To do this, we simply scale each coefficient of that matrix by a power of two and round the result. Then, we compute the matrix/vector multiplication and shift the result to the left to get the correct values for coefficients YCrCb. For example, to compute the luminance Y component, we make the following operations :

$$[\mathrm{Y}] = (ca'\,[\mathrm{R}] + (1 - ca' - cb')\,[\mathrm{G}] + cb'\,[\mathrm{B}]) \gg n;$$

where $ca' = \lfloor ca \times 2^n \rceil$ and $cb' = \lfloor cb \times 2^n \rceil$.

The second step corresponds to the management of the output range. In fact when RGB coefficients take their values in $[0,255]$, Y, Cr and Cb end up with values in $[0,255]$, $[-180,180]$ and $[-225,226]$ respectively. Still, in practice, it is required that the luminance and chrominance coefficients ends up in the same range. To accomplish that, there are two standard methods.

— *Method 1* : normalize the chrominance coefficients such that they range in -128 to 128 and add an offset equal to 128 to shift the result to the appropriate range (see [BT95]).

— *Method 2* [BT13] : clip the chrominance coefficients which are below 0 to 0 and above 255 to 255. E.g., if x is the output value of line 2 of equation (5.1), then

$$\mathrm{C}_b = \mathrm{clip}(x) = \min\{255; \max\{0; x\}\}.$$

For the first method, the normalization operation can be included directly in the chrominance coefficients of (5.1) before we transform it into an integer. In the second method we need to transform the clip function in the circuit matching with operations of Type I. For that, we can use conditional assignment as follows :

$$[z] = \mathrm{clip}([x]) \quad \Leftrightarrow \quad [z] = [y < 255] \cdot [y] + (1 \oplus [y < 255]) \cdot [255]$$
$$\text{where } [y] = [0 < x] \cdot [x] + (1 \oplus [0 < x]) \cdot [0].$$

Using a Type I FHE, we now compare 3 different strategies for our conversion : (i) using the basic RGB to YCrCb Equation (5.1) without adjusting the range of the coefficients, (ii) using *Method 1* or (iii) using *Method 2*. For each strategy, we count the number of XOR, AND and OR (bit-level) operations that need to be performed. Our results are given in Table 5.2 (left side).

A simple implementation of Equation (5.1) requires 6 additions, 9 scalar multiplications, which corresponds to line 1 in Table 5.2 (left side). When we apply *Method 1*, we

Algorithm	XOR	AND	OR
RGB to YCrCb	1556	484	1
Method 1	1556	482	3
Method 2	1992	712	11

Algorithm	ADD	SCALAR MULT.
RGB to YCrCb	6	9
Method 1	9	12
Method 2	no	no

TABLEAU 5.2 – Number of operations for two methods in conversion RGB to YCrCb with Type I (left) and Type II (right)

only add a scalar addition for each row in Equation (5.1). This does not increase much the number of bit-level operations (see line 2 in Table 5.2 left side). Applying *Method 2* adds 12 more additions and 12 more ciphertextwise multiplications, leading to a significant increase of bit-level operations (see line 3 in Table 5.2) (left side).

RGB to and from YCrCb with Type II FHE. A Type II FHE allows to encode both integers and rational numbers so that both solutions are possible for the conversion of the coefficients of the matrix in Equation (5.1). So as to truly compare Type I and Type II FHEs, we consider the same three strategies : basic equation, *Method 1* and *Method 2*. This first means that shift operations ($\gg n$) should be replaced, in this case, by a multiplication by a rational encoding of $(\frac{1}{2^n})$.

Yet, among the two above methods, only *Method 1* can really be implemented using a Type II FHE. In fact, to implement *Method 2*, one needs to perform a comparison, or execute at least some form of conditional assignment to realize the clip operator, and we have explained in Section 5.3 that this is not possible with a Type II FHE.

Our results in the Type II FHE case are given in Table 5.2 (right side). Our complexity is here defined in the number of additions, multiplications and scaling, since this type of FHE does not implement bit-level FHE operations.

5.4.2 Discrete cosine transform (DCT)

The DCT is generally the second algorithm in a macroblock compression pipeline. It changes the representation of the block coefficients to the frequency domain. This transformation uncorrelates the coefficients and concentrates their energy in the low frequencies, i.e., after the DCT, the coefficients that contain the important informations are on the top-left side of the resulting block. For a block $\mathbf{X}$ of size $\mathrm{N}\times\mathrm{N}$, its DCT is given by the following equation

$$\mathbf{Y} = \mathrm{DCT}(\mathbf{X}) = \mathbf{AXA}^{\mathrm{T}}, \text{ where } \mathbf{A} = (\mathbf{A}_{ij})_{ij} \text{ is a matrix with}$$
$$\mathbf{A}_{ij} = c_i \cos\frac{(2j+1)\,i\pi}{2\mathrm{N}}; \quad c_i = \sqrt{\frac{1}{\mathrm{N}}}\,(i=0); \quad c_i = \sqrt{\frac{2}{\mathrm{N}}}\,(i>0). \tag{5.2}$$

The DCT is a reversible transformation. Its inverse, denoted IDCT, is similar to forward DCT and is used in the decompressor. Its equtaion is defined by

$$\mathbf{X} = \mathrm{IDCT}(\mathbf{Y}) = \mathbf{A}^{\mathrm{T}}\mathbf{YA}, \text{ where } \mathbf{A} \text{ is defined as in the forward DCT.}$$

DCT for Type I FHE. The implementation of the DCT in the encrypted domain, using a Type I FHE, cannot directly use the coefficients of the matrix A in Equation (5.2). In fact, this necessitates using floating points, requiring many multiplications (about 2×64 for a 8×8-DCT), which one is the most costly operation of nowadays homomorphic encryption schemes. Then, instead of using Equation (5.2) directly, one can use an integer approximation of the DCT. There are three ways to obtain this :

— scale each matrix coefficient with some large number (typically between 2^5 and 2^{16}) and then round to the closest integer [MHKK03];

— decompose the matrix **A** as a product of two matrices $\mathbf{D}\cdot\mathbf{B}$ where **D** is a diagonal matrix with rational coefficients and **B** is a matrix with integer coefficients. The matrix **B** is then decomposed into a product of sparse integer matrices. Following the decomposition obtained for **B**, the number of multiplications is quite better [BC12];
— hand-turn the coefficients of matrix **A** such that they satisfy some conditions defined by standard HEVC [BFB+13] (orthogonality, good compaction energy, equal norm of column matrix, etc).

In the sequel, we focus our attention on the two last methods, using the standard implementations defined in H.264 (which uses second method on 4×4 blocks) and HEVC, which gives us several advantages. At first, these two standards are among the most deployed ones in practice. Moreover, for our purpose, they propose approximate DCT with integer coefficients (the rational part of the DCT being included in the quantization step studied below). Indeed, they can be implemented using only additions and shifts. Table 5.3 then gives the number of bit-level operations for both standards to compute such approximate integer DCT.

Algorithms	XOR	AND	OR
Approximate DCT for H.264 [R.04]	1828	569	20
Approximate DCT for HEVC [BFB+13]	7738	2369	26

TABLEAU 5.3 – Number of operations in DCT-4×4 for H.264 and HEVC with Type I FHE

DCT for Type II FHE. For a Type II FHE, it may be possible to work with the basic definition of the DCT, since it permits to handle rational numbers. But, as in the non-encrypted domain, this leads to some accuracy errors between the encoder and the decoder. We then similarly work with the expressions defined in H.264 and HEVC, given above. The way to proceed is similar to Type I FHE, except that the shift operation becomes a scalar (rational) multiplication. Table 5.4 gives the number of basic operations for performing DCT in this context.

Algorithm	ADD	SCALAR MULT.
Approximate DCT for H.264 [R.04]	96	16
Approximate DCT for HEVC [BFB+13]	64	76

TABLEAU 5.4 – Number of operations in DCT-4×4 for H.264 and HEVC with Type II FHE

5.4.3 Quantization

Quantization is directly correlated to the DCT step defined previously. It consists in dividing the frequency coefficients derived from DCT by a matrix **Q**, and to further round them. The coefficients of the matrix **Q** are defined following a quantization parameter which regulates the compression ratio. For a DCT coefficient **Y**, the equation of quantization is the following :

$$\mathbf{Z}_{i,j} = \left\lfloor \frac{\mathbf{Y}_{i,j}}{\mathbf{Q}_{i,j}} \right\rceil . \tag{5.3}$$

The quantization parameter defines the quality of the output image. Quantization is then a lossy transformation since it is not possible to determine the exact value of the original fractional number from the rounded integer. In the decompression step, one uses a transformation which is an approximate reversion of the forward quantization. Its equation is

the following :

$$\mathbf{Y}_{i,j} = \mathbf{Z}_{i,j} \cdot \mathbf{Q}_{i,j}.$$

For further implementation details, we refer the reader to the description of quantization in the H.264 standard [R.04]. In this case, the DCT transformation is decomposed in the product $\mathbf{AXA}^{\mathrm{T}} \otimes \mathbf{E}_f$ where the coefficients of $\mathbf{A}$ are in $\{-1,-2,1,2\}$ and those of $\mathbf{E}_f$ in $\{a^2, \frac{b^2}{4}, \frac{ab}{2}\}$, $a = \frac{1}{2}, b = \sqrt{\frac{2}{5}}$. The symbol $\otimes$ here represents the coefficient-wise multiplication of the matrix. Then, $\mathbf{AXA}^{\mathrm{T}}$ represents the forward DCT in H.264 and $\mathbf{E}_f$ is included in the quantization step. So Equation (5.3) becomes

$$\mathbf{Z}_{i,j} = \left\lfloor \mathbf{Y}_{i,j} \frac{\mathbf{E}_{i,j}}{\mathbf{Q}_{i,j}} \right\rceil, \tag{5.4}$$

where $\mathbf{E}_{i,j}$ refers to the coefficients of matrix $\mathbf{E}_f$. In order to simplify arithmetic implementations, the H.264 standard suggests to replace the fraction $\frac{\mathbf{E}_{i,j}}{\mathbf{Q}_{i,j}}$ by $\frac{\mathbf{M}_{i,j}}{2^{qbits}}$ such that $\frac{\mathbf{M}_{i,j}}{2^{qbits}} = \frac{\mathbf{E}_{i,j}}{\mathbf{Q}_{i,j}}$ and to implement Equation (5.4) as

$$|\mathbf{Z}_{i,j}| = \left(|\mathbf{Y}_{i,j}| \cdot \mathbf{M}_{i,j} + f\right) \gg qbits, \quad sign(\mathbf{Z}_{i,j}) = sign(\mathbf{Y}_{i,j}) \tag{5.5}$$

where $qbits = 15 + \lfloor \mathrm{QP}/6 \rfloor$ and $f = \frac{2^{qbits}}{3}$.

Quantization for Type I FHE. In this case, we use Equation (5.5) for implementing the quantization step. To obtain the absolute value of an integer y, we first extract the (encrypted) sign bit of y and then use the conditional assignment defined above. With a Type I FHE, the sign of a ciphertext $[y]$ is simply the leftmost ciphertext bit. For a ciphertext $[y]$ represented by 16 ciphertext bits, Equation (5.5) can then be computed as follows :

$$\begin{aligned} [b] &= \mathrm{MSB}([y]); \\ \left[y'\right] &= [b]\left(2^{16} - 1 - [y]\right) + (1 \oplus [b])[y]; \\ [z] &= \left(\left[y'\right] \cdot \mathbf{M}_{i,j} + f\right) \gg qbit, \quad msb([z]) = [b]. \end{aligned}$$

Quantization for Type II FHE. In the case of Type II FHE we cannot use Equation (5.5) as recommended in the H.264 standard. Indeed, extracting the (encrypted) sign of a given ciphertext is not possible. So, we rather use Equation (5.4) which evaluation does not depend on the sign of the ciphertext. In order to avoid the problem of the round function of this equation, we add the scalar $\frac{1}{2}$ to $\mathbf{Y}_{i,j}\frac{\mathbf{E}_{i,j}}{\mathbf{Q}_{i,j}}$ and after decryption of a quantized ciphertext, we decode it as an integer rather than as a rational number.

NOTE : the last step of the compression pipeline is the *entropic coding*. It is composed of an RLE compression followed by an arithmetic or Huffman coding. Running RLE compression on encrypted data has already been studied in [CCNS17]. It remains the arithmetic or Huffman coding, which one is out of the scope of our chapter and remains an open problem.

5.5 Our results

The previous section has given our theoretical view of a macroblock processing pipeline in the encrypted world. We have shown that, depending on the FHE instantiation (Type I or Type II), the strategy on the way to implement each step (color conversion, DCT and quantization) can be quite different. We have proposed several versions and

gave the advantages and drawbacks of each, before giving our theoretical conclusions. We now go one step further by going deep inside a real implementation.

More precisely, in this section, we evaluate the performances, in terms of execution time, when the compression algorothms is executed using two different FHE implementations : one falling in the Type I case and the other based on a Type II FHE instance.

Parameters. To obtain our real-world benchmarks, we have used both

— the `Cingulata` library, which implements the FV FHE scheme with a Type I instance and

— the `SEAL` library, which also implements the FV FHE scheme but using a Type II instance.

In our implementations, we have set the parameters of the two FHEs to reach a 128-bit security, using [LP16]. More precisely, we have chosen $n = 2048$ and $\log_2 q = 56$ for the degree of the cyclotomic polynomial and the ciphertext modulus respectively. For the plaintext modulus, we have set $t = 2$ for `Cingulata` and $t = 2^8$ for `SEAL`. Finally, regarding the compression algorithm we have implemented in the encrypted world, we have also made some choices on the input parameters, depending on the FHE type.

— *Color space conversion* : we have first used as input the encryption of a single RGB triplet (YCrCb one for the inverse function), but the results can easily be extended to a full image. Using `Cingulata`, the input is encoded with the Integer8 class. During homomorphic operations, we have used instead the Integer32 class as an accumulator to store our ciphertext since it permits to avoid the precision error due to carry propagation. More precisely, we multiply the coefficients of the matrix color conversion by 2^15 (see Section 5.4). Since the encrypted input coefficients are 8-bit ciphertexts, we should use at least a Integer24 class for this computation. For the `SEAL` library, as it corresponds to a Type II FHE, we do not have such problem since the inputs are encoded as coefficients of a polynomial.

— *DCT and Quantization* : in these algorithms, we have chosen a 4×4 macroblock. This size is the basic one for blocks used in H264. For our comparison, we use the same size when testing the HEVC compression algorithm. In `Cingulata`, we represent the encrypted input in the Integer16 class. As for color space conversion, we use an intermediary Integer32 representation to avoid precision error.

We have implemented our algorithms using an Intel(R) Xeon(R) with a E5-2620 CPU with 16 core running at 2.10GHz under a 64-bit Arch-linux OS.

The Table 5.5 gives the benchmarks of our implementation in term of numbers of operations for each type of FHE It also gives the execution time for each pipeline compression algorithm.

		TYPE I						TYPE II				
		AND	XOR	OR	depth	*2nd generation* time(s)	*4th generation* time(s)	ADD	MULT	SCALAR MULT.	depth	*2nd generation* time(s)
RGB to YCrCb	*method 1*	482	1556	3	22	164	3.062	9	0	12	0	0.0029
	method 2	712	1992	11	37	250	4.073	n.a.	n.a.	n.a.	n.a.	n.a.
YCrCb to RGB	*method 1*	377	1129	2	21	132	2.262	7	0	8	0	0.0021
	method 2	346	1027	444	30	135	2.726	n.a.	n.a.	n.a.	n.a.	n.a.
DCT for H.264	forward	516	1456	4	15	54	2.964	96	0	16	0	0.0078
	inverse	1186	3455	8	16	63	6.974	96	0	48	0	0.0147
DCT for HEVC	forward	2369	7738	26	24	343	15.200	64	0	76	0	0.0213
	inverse	372	1326	4	26	88	2.553	64	0	76	0	0.0210
Quantization for H.264	forward	2100	6892	0	29	64	13.448	16	0	32	0	0.0072
	inverse	216	892	12	12	65	1.680	0	0	32	0	0.0066
Quantization for HEVC	forward	1968	6688	0	29	365	12.984	16	0	48	0	0.0075
	inverse	144	624	16	10	30	1.176	16	0	48	0	0.0100

TABLEAU 5.5 – Execution times of algorithms of pipeline image/video compression in function of Type I and Type II FHE. We use FV as a *2nd generation* FHE and TFHE as *4th generation* FHE.

NOTE : the execution times given for Type I/*4th generation* FHE in the Table 5.5 correspond to a theoretical estimate of a real execution time for TFHE. Indeed, in the TFHE scheme, all bitwise operations take exactly the same time : about 15ms. Then, using the information about binary gates of compression sub-algorithms taken from our implementation using `Cingulata`, we obtain the execution times for TFHE using the equation

$$\text{time}(\text{TFHE}) = (\#\text{OX} + \#\text{AND} + \#\text{OR}) \times 15ms. \tag{5.6}$$

Results. The first thing we observe in Table 5.5 is the gap between the execution time for a Type I and for a Type II FHE. In fact, this gap was quite predictable since a Type I FHE is by definition some Type II FHE for a modulus plaintext set to 2 and arranged in such a way that it reproduces the functionalities of a classic processor. Yet, as shown is Table 5.5, the Type I/Type II gap is significantly reduced when using a 4th generation Type I FHE such as TFHE.

Still, a Type I FHE gives far more possibilities regarding the operations that can be executed in the encrypted domain. This is due to the used structure that is more accessible than the polynomial structure of a Type II FHE. Clearly, this latter limits the operations in the encrypted world to the existing ones in the polynomial ring.

Application to an image. We put together our above implementations to execute the compression/decompression pipeline on a complete real 128×128 image. We have done so for the algorithms sets of both H264 and HEVC standards. Yet, as the `SEAL` library seems to be the most efficient, we limited ourselves to our results on Type II FHE. Hence, the image crosses successively the following algorithms : encryption of pixels, DCT with block size 4×4, quantization, inverse quantization, inverse DCT with block size 4×4 and finally a decryption of the resulting pixel. Figures below give the resulting images obtained after applying the above pipeline to the grayscale well-known picture of Lena with size 128×128.

Compressing the full image takes about 244s (H.264) and 149s (HEVC) on a single core, which is clearly not prohibitive. Needless to emphasize, of course, that, since the image macroblocks are processed independently, significant speedups can be further obtained by means of parallelization.

To the best of our knowledge, this is the first "crypto-compression" of a full image, yet of moderate resolution.

FIGURE 5.2 – Lena original

FIGURE 5.3 – Lena (H.264)

FIGURE 5.4 – Lena (HEVC)

Conclusion

In this work, we show that there are today two different ways to use FHE schemes : Type I which performed encrypted bitwise operations, and Type II which makes use of polynomial ring operations. We apply these two types of FHE to the main blocks of a pipeline of image/video compression. We observe, without more surprise, that Type II FHEs

are faster than Type I, but that the latter has more flexibility in term of operations that can be executed in the encrypted domain. This work opens many other interesting problems, such as the optimization of our circuit to reduce the execution time, the definition of new operations like conditional assignment for a Type II FHE, and why not the definition of an operator which allows to switch between Type I and Type II, then obtaining the advantage of both types.

Troisième partie

Manipulation de messages chiffrés en homomorphe

Chapitre 6

Towards More Flexible Slot Management in Somewhat Homomorphic Encryption Schemes

Sommaire

6.1 Introduction

Homomorphic encryption is an advanced cryptographic technique which allows to encrypt some data and carry out arithmetic operations directly on these encrypted data. The interesting property is that the decryption of the ciphertexts resulting from this handling gives the same results as if the operations had been carried out in the clear domain. This idea was born in 1978 and proposed for the first time by Rivest, Shamir and Dertouzos [RAD78]. But it has remained speculative for more than 30 years.

Several attempts have been made to obtain such homomorphic encryption schemes, but most of them have led to limited schemes allowing only to carry out either an unlimited number of additions [Pai99] or an unlimited number of multiplications [RSA78, Gam84], but never both at once.

The first *fully* homomorphic encryption (FHE) scheme was proposed in 2009 by Craig Gentry [Gen09b]. The proposed scheme is able to perform both additions and multiplications an unlimited number of times. To obtain such a scheme, Gentry first designed a *somewhat* homomorphic encryption (SHE) scheme, for which a limited number of both additions and multiplications are possible, due to the increase of the amplitude of a noise term after each operation (with the property that if the noise amplitude is too large, the plaintext cannot be retrieved). Then, he introduced a so-called *bootstrapping* procedure which consists in (1) representing the decryption algorithm as a circuit, (2) encrypting the secret decryption key using the SHE scheme and (3) evaluating the decryption circuit on the encrypted ciphertext. The resulting ciphertext still encrypts the same initial plaintext but the noise has decreased, so that new operations can now be further performed.

Henceforth, several SHE schemes have emerged, among which, the most relevant ones for a practical implementation are the Brakerski-Gentry-Vaikuntanathan scheme [BGV12] (BGV), the Fan-Vercauteren one [FV12] (FV), and YASHE [BLLN13a]. All these schemes have their security based on the *Learning With Error* (*LWE*) problem, defined over lattices, or on its ring variant, the *Ring-LWE* problem, defined on ideal lattices, on which we will focus on this chapter. In fact, in this latter case, the plaintext is an element $\mathbf{m}$ of a polynomial ring, and can then be written as $\mathbf{m} = \sum_{i=0}^{n-1} m_i x^i$, where the m_i's are the coefficients of $\mathbf{m}$.

In this work, we want to address the following problem. Given a *composite* ciphertext ct of a *composite* plaintext $\mathbf{m}$ (for which this is possible to perform homomorphic operations, and to decrypt it, using the procedures given by the SHE scheme, e.g., FV, BGV or YASHE), we want to mathematically define an *atomic* ciphertext $\mathbf{c}_j^*$ related to the *atomic* plaintext m_j. Then we would like to benefit from homomorphic operations (including decryption) directly over the atomic plaintext space, manipulating the resulting atomic ciphertext.

6.1.1 Our Contributions

Given the above problem, our contribution in this chapter is multiple.

1. We introduce the general framework for a SHE (and then a FHE) defined over a polynomial ring (e.g., FV, BGV or YASHE) so that the result becomes relevant to our transformation. For this purpose, we introduce some extended procedures that should be added to the traditional ones for a SHE (namely, key generation SecretKeyGen and PublicKeyGen, encryption Encrypt, operations Add, Mult and decryption Decrypt). More precisely, (i) `CipherProj` permits to go from a composite ciphertext (output by Encrypt and related to a ring element) to an atomic one (related to an integer in $\mathbb{Z}$), (ii) AtomicAdd (resp. AtomicMult) defines additions (resp. multiplications) on atomic plaintexts, manipulating the related ciphertexts, (iii) AtomicDecrypt

is used to decrypt an atomic ciphertext and (iv) CipherRet permits to come back to a composite ciphertext, related to a composite plaintext (an element in the ring).

2. We describe our general ideas for such a transformation for any SHE scheme defined over a polynomial ring, namely FV, BGV and YASHE. We then give the mathematical details for the specific case of the Fan-Vercauteren SHE scheme [FV12]. In particular, we also detail the addition and multiplication that should be performed in the atomic world.
3. We exhibit several different procedures to go from the atomic to the composite representation of plaintexts. This permits us to go back to the initial SHE/FHE scheme, and then make use of the initial methods.
4. We give some practical considerations, explaining how our solution can be used in practice, and the advantages one can have in using our method, in particular compared to and in cunjunction with transciphering.

Prior results.

As far as we know, the prior work on our studied problem is rather scarce.

There are some *transciphering* methods [NLV11, GHS12, CCF+16] for which the idea is to use a symmetric encryption scheme (advantageously a stream cipher, as pointed in [CCF+16]) in order to treat the problem of an important ciphertext expansion, inherent to lattice-based cryptography. In a nutshell, the plaintext m is encrypted with a symmetric encryption scheme E under a randomly chosen session secret key k, as $c' = \mathsf{E}_k(m)$. Then, the secret key k is in turn encrypted using the FHE scheme, as $ct = \mathsf{FHE}_{\mathsf{pk}}(k)$. Given both ciphertexts, it is then possible to exploit the homomorphic property of ct to compute $ct' = \mathsf{FHE}_{\mathsf{pk}}(m)$, by homomorphically evaluating the decryption circuit related to E^{-1}. Our problem is quite different and the existing transciphering methods are obviously not relevant in our context. However, as we will see (just below and in Section 6.6), our method can be used to reduce even more the ciphertext expansion, since, compare to [CCF+16], it is no more necessary to encrypt bit-by-bit the secret key k : one single ciphertext is enough.

In parallel, some *batch* methods have been proposed [GHS12, SV11]. These techniques make use of the SIMD (Single Instruction Multiple Data) concept, which consist in packing multiple plaintexts into the slots of a single ciphertext, and then perform arithmetic operations on the whole set of slots. Using rotations and permutations, it is also possible to isolate each plaintext into a separate ciphertext and then execute a given circuit on the latter. There are however several problems related to this method. At first, SIMD needs a larger plaintext modulus ($t > 2n$) which involves a reduction on the depth of ciphertext. Secondly, the message expansion is very high when considering the difference between a single plaintext and the whole ciphertext (since all slots are manipulated while only one of them is really relevant). Finally, each arithmetic operation is necessarily executed on all slots, which make the results quite impractical in some cases (typically when only one plaintext is relevant). Our new method does not have such drawbacks.

The approach which is the closest to ours is the one in [CS16a]. In fact, the authors propose a ciphertext (resp. plaintext) packing method, permitting to go from several ciphertexts (resp. plaintexts) to one single, using a polynomial ring element which coefficient are the initial set of ciphertexts (resp. plaintexts). They also propose the reverse unpacking method. However, their solution suffers from some important drawbacks. At first, arithmetic operations are not optimised for both packed and unpacked ciphertexts since they are always similar. Moreover, the transformation is done using homomorphic operations, since it is based on a specific circuit dedicated to the (un)packing procedure.

In particular, this induces a significant noise increase, which may be prohibitive regarding performances.

In opposition, our solution is to focus on the mathematical structure of the ciphertext, so that there is no noise increase during the transition between a composite and an atomic plaintext. Moreover, we succeeded in describing optimized homomorphic additions and multiplications procedures.

Our approach.

Given a SHE scheme defined over a polynomial ring, we have noticed several mathematical properties that can be used to solve our problem. The polynomial structure allows us to use the arithmetic operations defined on the polynomial coefficients to deduct our extensions. More precisely, the central part of the decryption operation (e.g., $\mathbf{c}_0 + \mathbf{c}_1\mathbf{s}$ for FV, and $\langle \mathbf{c}, \mathbf{s} \rangle$ for BGV and YASHE) gives us a polynomial, denoted F in the sequel, which coefficients f_j have the following two interesting properties :

— they can be expressed using the coefficients of the secret key $\mathbf{s}$ and the ciphertext ct ;

— they can be used to independently obtain the coefficients m_i of the message $\mathbf{m}$ by simply executing the rest of the decryption procedure (e.g., $m_i = \left[\left\lfloor \frac{t f_i}{q} \right\rceil \right]_t$ for FV and YASHE, and $m_i = [[f_i]_q]_2$ for BGV).

Focusing on a particular SHE scheme, namely FV [FV12], we are then able to concretely specify how to extract from ct the necessary information (in the form of operations over the coefficients of F) that allows us to create the ciphertext $\mathbf{c}_i^*$ related to the plaintext m_i. As this is only an extraction, we do not increase the noise from the composite to the atomic ciphertext.

Reverse transformation.

As explained before, it can be interesting, in some cases, to come back to the initial setting of a polynomial ring, and make use again of the procedures defined by the initial SHE scheme. We then propose three different methods to perform such a reverse transformation. Each of them has some pros and cons.

The first strategy permits to go back to a global composite form. This strategy makes use of n ciphertexts on input and outputs one single ciphertext related to a ring element with its n coefficients related to the n inputs. This makes things quite interesting in practice since we are free on the atomic plaintexts we will use to create the composite ciphertext (and plaintext). The main drawback of this method is that it requires to perform some *bootstrapping*-like operations, which makes things not as practically efficient as we may expect.

The second strategy is a returning to some unitary composite form for which we have on input one single atomic ciphertext. The good point, especially compared to the previous method, is that the complexity is quite good. The main drawback is that, with this method, we do not control the other slots of the resulting plaintext which become useless. However, this idea is very useful to manage an efficient homomorphic multiplication, since, as we will see, it permits to directly apply the one by Fan and Vercauteren [FV12], and its improvements.

Finally, the last strategy, close to the first one, gives a compact composite form. In fact, compared to the first strategy, we come back to a new type of ciphertext which aims at compacting the messages $m_0, m_1, \ldots m_{r-1}$. This transformation is much more efficient, but the price to pay is that this new ciphertext cannot be handled and decrypted by the basic decryption algorithm. We have then to define a new algorithm to decrypt it.

6.1.2 Discussion

Fan-Vercauteren FHE case.

In the rest of the chapter, we focus on the Fan-Vercauteren [FV12] SHE scheme, which is currently, to the best of our knowledge, the most efficient one. Applying our above method to this scheme gives us good performances regarding time and space complexities. More precisely, the atomic ciphertext $\mathbf{c}_i^*$ related to the plaintext m_i is given by

$$\mathbf{c}_i^* = \left(c_{0,i}, c_{1,i}, c_{1,i-1}, \ldots, c_{1,0}, -c_{1,n-1}, \ldots, -c_{1,[i+1]_n}\right),$$

and we then show that $\langle \mathbf{c}_i^*, \mathbf{s}^* \rangle \mod q = f_i$, which permits us to define the coefficient f_i of our polynomial F.

Additionally, we describe the way homomorphic additions and multiplications can be performed on atomic ciphertexts. In fact, addition is quite easy, since it simply corresponds to $\mathbf{c}_{Add}^* = \left(\mathbf{c}_i^* + \mathbf{c}_j^*\right) \mod q$. Regarding multiplication, our idea is to use one reverse transformation above to go back to a unitary composite form, then directly apply the Fan-Vercauteren homomorphic multiplication, and finally go back to an atomic form by using our ciphertext projection. As we will show, this permits us to obtain a very efficient homomorphic encryption.

Open problems.

One natural open problem is to find more efficient and complete ways to provide the reverse transformation. The ones we give in this chapter are either complete (going back to a global composite form where we manage all atomic values) but not enough efficient (typically using bootstrapping-like solutions) or efficient but not complete (only managing a limited number of atomic values, or necessitating to define a new decryption procedure for composite ciphertexts). One possibility could be to first define a linear system based on the coordinates of the atomic ciphertexts and the ones from the secret key, and where the solution correspond to the coordinate of the new composite ciphertext. The second step then consists in blinding the coordinates of the secret key in the system, using e.g., encryption.

6.1.3 Organization of the Chapter

The chapter is organized as follows. We first give some preliminaries in the next section. Section 6.3 is then dedicated to the main extensions we propose, focusing on the Fan-Vercauteren SHE scheme. In Section 6.4, we give the new ways to perform additions and multiplications with our extensions. Section 6.5 is dedicated to the different possible strategies to do the reverse transformation. Before concluding, we finally give in Section 6.6 some performance estimations with respect to communication overheads and show how our techniques can articulate with transciphering to further decrease the overhead.

6.2 Preliminaries and Model

The objective of this section is to remind the main components we will use in the rest of the chapter. We start with some notations, and then gives some useful routines. We also describe the model for homomorphic encryption schemes and then give the additional procedures we need to solve our problem. We finally give some details on the Fan-Vercauteren SHE [FV12] since this scheme will be a common thread all along the chapter.

6.2.1 Notations

We begin by introducing the polynomial ring on which we will work in this chapter : $\mathrm{R} = \mathbb{Z}[x]/(\Phi(x))$, where $\Phi(x) \in \mathbb{Z}[x]$ is a monomial irreducible polynomial of degree n. In practice, we use the polynomial $\Phi(x) = x^n + 1, n = 2^k$, which is the $(n+1)$-th cyclotomic polynomial. We note elements of R in lowercase bold and the coefficients of a polynomial $\mathbf{a}$ are denoted a_i. This way, we have $\mathbf{a} = \sum_{i=0}^{n-1} a_i x^i$. We also represent vectors in lowercase bold $\mathbf{v} \in \mathbb{Z}^n$ and matrices in upper case bold $\mathbf{A} \in \mathbb{Z}^{n \times m}$. The i-th coordinate of a vector $\mathbf{v}$ is represented as v_i and sometime it is noted $\mathbf{v}[i]$. A ring element $\mathbf{a}$ can also be identified by its vector of coefficients $(a_0, a_1, \ldots, a_{n-1})$. The inner product of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^n$ is denoted $\langle \mathbf{u}, \mathbf{v} \rangle$.

For an integer $a \in \mathbb{Z}$, we denote $|a|$ the absolute value of a, for a polynomial $\mathbf{a}$ of R, we define $\|\mathbf{a}\|_\infty$ to be the maximum of the absolute values of coefficients of $\mathbf{a}$, i.e. $\|\mathbf{a}\|_\infty = \max_i |a_i|$. The expansion factor of ring R is defined to be $\delta_\mathrm{R} = \max\{\frac{\|\mathbf{ab}\|_\infty}{\|\mathbf{a}\|_\infty \|\mathbf{b}\|_\infty} : \mathbf{a}, \mathbf{b} \in \mathrm{R}\}$, e.g. : for $\Phi(x) = x^n + 1$, we have $\delta_\mathrm{R} = n$. We define the norm of a vector to be the *l_2-norm* , that is, for a vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1}) \in \mathbb{Z}^n$, $\|\mathbf{v}\|_2 = \left(v_0^2 + v_1^2 + \cdots + v_{n-1}^2\right)^{\frac{1}{2}}$.

Let $q > 1$ be an integer, we denote by $\mathbb{Z}_q$ the integers of $]-\frac{q}{2}, \frac{q}{2}]$, i.e., $\mathbb{Z}_q = \mathbb{Z} \cap]-\frac{q}{2}, \frac{q}{2}]$. We can also define R_q to be the set of polynomials of R with coefficients in $\mathbb{Z}_q$. For $a \in \mathbb{Z}$ we denote by $[a]_q$ the rest of euclidean division of a by q, i.e. $[a]_q = a \mod q$. Similarly, when $\mathbf{a} \in \mathrm{R}$(resp. is a vector), we define $[\mathbf{a}]_q$ as the polynomial of R_q (resp. the vector) obtained by applying $[\cdot]_q$ to all coefficients (resp. coordinates) of $\mathbf{a}$.

For a positive real σ, we denote by $\mathrm{D}_{\mathbb{Z},\sigma}$ the discrete gaussian distribution of mean 0 and the deviation σ over the integers, i.e., a distribution of probability which maps an integer $x \in \mathbb{Z}$ to a proportional probability to $exp(-\pi|x|^2/\sigma)$. For a good real $\mathrm{B} > 0$, we show that the distribution $\mathrm{D}_{\mathbb{Z},\sigma}$ with a support in $\mathbb{Z}$ is statistically indistinguishable from a distribution which support is on $\mathscr{S} = [-\mathrm{B}, \mathrm{B}]$ denoted $\mathrm{D}_{\mathscr{S},\sigma}$ [RVV13]. In practice one would usually set $\mathrm{B} = 10\sigma \ll q$. We denote by χ the probability distribution on R_q obtained by drawing any coefficient of R_q'elements from $\mathrm{D}_{\mathbb{Z}_q,\sigma}$.

6.2.2 Some Useful Routines

Let $\omega > 1$ be an integer, and let $\ell_\omega = \lceil \log_w q \rceil + 1$. For all integers $a \in \mathbb{Z}_q$, there is a unique representation of a in base ω, i.e., there are coefficients $a_0, a_1, \ldots, a_{\ell_\omega - 1} \in]-\omega/2, \omega/2]$ such that $a = a_0 + a_1\omega + \cdots + a_{\ell_\omega - 1}\omega^{\ell_\omega - 1}$. We can now define [BV11a, BGV12] the function $\mathsf{WordDecomp}_\omega$ (noted in the following as WD_ω) by $\mathsf{WD}_\omega(a) = \left(a_0, a_1, \ldots a_{\ell_\omega - 1}\right)$.

Another useful function is $\mathsf{PowersOfWord}_\omega$ (noted PW_ω in the following) which can be defined as

$$\mathsf{PW}_\omega(a) = \left(a, [a\omega]_q, \ldots, [a\omega^{\ell_\omega - 1}]_q\right).$$

These functions can easily be extended to polynomials of R_q or vectors of $\mathbb{Z}_q^n$. It is easy to prove [BV11a, BGV12] that for all polynomials $\mathbf{a}, \mathbf{b}$ in R_q (or vectors of $\mathbb{Z}_q^n$),

$$\langle \mathsf{WD}_\omega(\mathbf{a}), \mathsf{PW}_\omega(\mathbf{b}) \rangle = \mathbf{a} \cdot \mathbf{b}.$$

6.2.3 Homomorphic Encryption Model and Additional Procedures

A homomorphic encryption scheme is given by a set of procedures that can be defined as follows.

— $\mathsf{SecretKeyGen}(1^\lambda)$ outputs the secret decryption key sk.

— $\mathsf{PublicKeyGen}(sk)$ outputs the public key pk related to the secret sk.

— $\mathsf{Encrypt}(pk, \mathbf{m})$ outputs a ciphertext ct which encrypts $\mathbf{m}$ under pk.

— Decrypt(sk, ct) outputs the plaintext $\mathbf{m}$.

There are also homomorphic operations that can naturally be resumed to Add (to homomorphically obtain the addition of plaintexts) and Mult (to homomorphically obtain the multiplication of plaintexts).

For our purpose, as we need to transform a composite ciphertext to an atomic one, we introduce new procedures that can be defined as follows.

— SKProj(sk) outputs a new secret key sk^* that will be used to decrypt an atomic ciphertext.

— CipherProj(ct, i) is the transformation of the composite ciphertext ct into an atomic ciphertext $\mathbf{c}_i^*$ related to the i-th coordinate of the plaintext in ct.

— AtomicDecrypt$(sk^*, \mathbf{c}_i^*)$ is the new decryption procedure dedicated to an atomic ciphertext.

Our aim is then to define these three new procedures in the case of the Fan and Vercauteren [FV12] SHE scheme, together with the new related homomorphic procedures AtomicAdd and AtomicMult for addition and multiplication respectively. Finally, CipherRet is the procedure to go back to a composite form.

6.2.4 Fan-Vercauteren SHE

The FV scheme is a somewhat homomorphic scheme introduced by Fan and Vercauteren [FV12] in 2012. It is one of best candidates in the field of homomorphic cryptography because it gives non-prohibitive performances, for a reasonable range of multiplicative depths as well as a simple choice of parameters. Its security depends on the *Ring-LWE* problem, that is the ring version of the LWE problem [Reg05], which can be defined as follows.

Definition 1 Ring-LWE problem

Let λ to be a security parameter, $q = q(\lambda) > 1$ an integer. For a random element $\boldsymbol{s} \in \mathrm{R}$ and a distribution $\chi = \chi(\lambda)$ over R, we denote by $\mathrm{A}^{(q)}_{\boldsymbol{s},\chi}$ the distribution on $\mathrm{R}_q \times \mathrm{R}_q$ obtained by drawing uniformly at random a element $\boldsymbol{a} \leftarrow \mathrm{R}_q$, and $\boldsymbol{e} \leftarrow \chi$ and returning the pair of polynomials $(\boldsymbol{a}, [\boldsymbol{as} + \boldsymbol{e}]_q)$. The Ring-LWE *problem consists in finding a secret $\boldsymbol{s}$, given the samples $\mathrm{A}^{(q)}_{\mathbf{s},\chi}$.*

This problem has been (quantumly and classically) reduced in the worst-case to ideal lattice problems, such as the shortest vector problem [LPR13, Pei09]. In the following, we will recall the algorithms of the homomorphic scheme FV.

For a plaintext modulus $t > 1$ which defines the plaintext space R_t, we define $\Delta = \lfloor q/t \rfloor$ and $r_t(q) = q \mod t (q = \Delta t + r_t(q))$ to be the quotient and the remainder of the Euclidean division of q by t. The FV homomorphic scheme is then defined as described in Figure 6.1.

— SecretKeyGen$\left(1^\lambda\right) : \mathbf{s} \leftarrow \chi$ and output $sk = \mathbf{s}$

— PublicKeyGen(sk) : set $\mathbf{s} = sk$, sample $\mathbf{a} \leftarrow \mathrm{R}_q$, $\mathbf{e} \leftarrow \chi$ and output $pk = \left([-(\mathbf{as} + \mathbf{e})]_q, \mathbf{a}\right)$.

— Encrypt$\left(pk, \mathbf{m}\right)$: to encrypt a plaintext $\mathbf{m} \in \mathrm{R}_t$, let $\mathbf{p}_0 = pk[0], \mathbf{p}_1 = pk[1]$, sample $\mathbf{u}$ uniformly from R_2 and $\mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$
and return $ct = \left(\left[\mathbf{p}_0\mathbf{u} + \mathbf{e}_1 + \Delta\mathbf{m}\right]_q, \left[\mathbf{p}_1\mathbf{u} + \mathbf{e}_2\right]_q\right)$.

— Decrypt(sk, ct) : set $\mathbf{s} = sk, \mathbf{c}_0 = ct[0], \mathbf{c}_1 = ct[1]$ and compute

$$\left[\left\lfloor \frac{t[\mathbf{c}_0 + \mathbf{c}_1\mathbf{s}]_q}{q} \right\rceil\right]_t.$$

FIGURE 6.1 – Fan-Vercauteren SHE

6.3 From Composite to Atomic Ciphertexts

In this section, we precisely present our extensions of the FV scheme. A similar technique can be found in [CGGI17b] with the operation SampleExtract, but the authors work on Torus FHE, and they do not define the operation to handle two extracted ciphertexts, which ones seem to be not as easy to describe. In this work, we define one technique which works on Ring FHE. Again, we do not need SIMD which involve small plaintext modulus and, compare to [CGGI17b], we describe homomorphic operations to handle extracted ciphertext (see next section). In the sequel of this section, we first give the main idea before giving the details of the new procedures. We finally show how the noise is in the resulting atomic ciphertext.

6.3.1 Overview of Our Method

Focusing on the FV decryption procedure, we remark that the main operation is $[\mathbf{c}_0 + \mathbf{c}_1\mathbf{s}]_q$. In fact, this naturally gives us a polynomial that we denote F, such that its coefficients can be expressed as a function of the coefficients of both the ciphertext ct and the secret key $\mathbf{s}$:

$$\mathrm{F}(x) = \sum_{i=0}^{n-1} f_i x^i = [\mathbf{c}_0 + \mathbf{c}_1\mathbf{s}]_q.$$

Moreover, we can also remark that the coefficients of the message $\mathbf{m}$ can be expressed using the coefficients of the polynomial F.
More precisely, let ct be a FV ciphertext which encrypts a plaintext $\mathbf{m} = \left(m_0 + m_1 x + \cdots + m_{n-1}x^{n-1}\right)$ under the secret key $sk = \mathbf{s} = \left(s_0 + s_1 x + \cdots + s_{n-1}x^{n-1}\right)$. As shown in the previous section, the ciphertext $ct = (\mathbf{c}_0, \mathbf{c}_1) \in \mathrm{R}_q^2$ can be written as $\left(\sum_{i=0}^{n-1} c_{0,i}x^i, \sum_{i=0}^{n-1} c_{1,i}x^i\right)$. By rewriting each coefficient of F as a function of $\mathbf{c}_0$, $\mathbf{c}_1$ and $\mathbf{s}$, we have the following relations : $\forall i \in \{0,\ldots,n-1\}$,

$$\begin{aligned} f_i \quad = \quad & c_{0,i} + c_{1,i}s_0 + c_{1,i-1}s_1 + \cdots + c_{1,0}s_i \\ & - c_{1,n-1}s_{i+1} - \cdots - c_{1,[i+1]_n}s_{n-1} \mod q. \end{aligned}$$

These expressions can be written as the inner products

$$\begin{aligned} f_i \quad = \quad & \langle \left(c_{0,i}, c_{1,i}, c_{1,i-1}, \ldots, c_{1,0}, -c_{1,n-1}, \ldots, -c_{1,[i+1]_n}\right), \\ & (1, s_0, s_1, \ldots, s_{n-2}, s_{n-1})\rangle \mod q; \end{aligned}$$

We then set $\mathbf{s}^* = (1, s_0, s_1, \ldots, s_{n-2}, s_{n-1})$ and, $\forall i$

$$\mathbf{c}_i^* = \left(c_{0,i}, c_{1,i}, c_{1,i-1}, \ldots, c_{1,0}, -c_{1,n-1}, \ldots, -c_{1,[i+1]_n}\right).$$

Now considering the whole decryption procedure, using the coefficient of F, it is easy to see that

$$\langle \mathbf{c}_i^*, \mathbf{s}^* \rangle \mod q = f_i \Rightarrow m_i = \left[\left\lfloor \frac{t f_i}{q} \right\rceil \right]_t, \forall i \in \{0,\ldots,n-1\}.$$

As sketched in the introduction, the same kind of treatment can also be done for BGV and YASHE schemes since the decryption procedures are quite close to the one of the FV scheme.

6.3.2 Details of the Extensions

Using the above results, we have all the material to describe the new procedures (as introduced in the section on preliminaries) and then manage both composite and atomic ciphertexts for the FV SHE scheme. The resulting extensions are then properly defined in Figure 6.2. We remark that as we do not really modify the FV scheme, as the

— $\mathsf{SKProj}(sk)$: parse $sk = \mathbf{s} = (s_0, s_1, \ldots, s_{n-1})$
and output $sk^* = \mathbf{s}^* = (1, s_0, s_1, \ldots, s_{n-1}) \in \mathbb{Z}^{n+1}$.
— $\mathsf{CipherProj}(ct, i)$: set $\left(\sum_{j=0}^{n-1} c_{0,j} x^j, \sum_{j=0}^{n-1} c_{1,j} x^j\right) = ct$
and output $ct_i^* = \mathbf{c}_i^* \in \mathbb{Z}_q^{n+1}$.
— $\mathsf{AtomicDecrypt}\left(sk^*, ct_i^*\right)$: output

$$m_i = \left[\left\lfloor \frac{t[\langle \mathbf{c}_i^*, \mathbf{s}^* \rangle]_q}{q} \right\rceil\right]_t \quad \forall i \in \{0, \ldots, n-1\}.$$

FIGURE 6.2 – Our extensions for Fan-Vercauteren SHE

concatenation of all atomic ciphertexts does not contain more information than the initial composite ciphertext, the security of our extension directly follows from the one of the initial FV scheme.

6.3.3 Discussion on Resulting Noise

We finally focus on the noise induced by our new atomic ciphertext. In fact, considering the FV ciphertext $ct = (\mathbf{c}_0, \mathbf{c}_1)$ for a plaintext $\mathbf{m} = \sum_{i=0}^{n-1} m_i x^i$, under secret key $\mathbf{s} = sk$, the following relationship holds : $[\mathbf{c}_0 + \mathbf{c}_1 \mathbf{s}]_q = \Delta \mathbf{m} + \mathbf{v}$,
where $\mathbf{v} = \left(v_0 + v_1 x + \cdots + v_{n-1} x^{n-1}\right)$ represents the noise of the ciphertext ct. Then, according to Lemma 1 in [FV12], $\|\mathbf{v}\|_\infty \leq 2 \cdot \delta_R \cdot B^2 + B$. We then deduce that :

$$\forall i \left[\langle \mathbf{c}_i^*, \mathbf{s}^* \rangle\right]_q = [\mathbf{c}_0 + \mathbf{c}_1 \mathbf{s}]_q [i] = \Delta m_i + v_i.$$

It is then easy to show that $|v_i| \leq \|\mathbf{v}\|_\infty$.

The next step is to describe the arithmetic operations that one should use on the atomic ciphertexts. This is the aim of the next section.

6.4 Arithmetic Operations over the Atomic Ciphertext Representation

In this section, we define the arithmetic operations required by our scheme to handle the new atomic ciphertexts. We then describe both additions and multiplications together with their impact on the noise.

6.4.1 Addition

Managing additions

With the previous representation, it is natural to define the addition $\mathsf{AtomicAdd}$ of our new ciphertexts as follows.

Let $\mathbf{c}_i^*, \mathbf{c}_j^*$ two ciphertexts that encrypt the messages m_i and m_j respectively. The ciphertext corresponding to the message $m_i + m_j$, denoted $\mathbf{c}_{Add}^*$, can be defined as

$$\begin{aligned} \mathbf{c}_{Add}^* &= \left(\mathbf{c}_i^* + \mathbf{c}_j^*\right) \mod q \\ &= \left(\left[\mathbf{c}_i^*[0] + \mathbf{c}_j^*[0]\right]_q, \ldots, \left[\mathbf{c}_i^*[n] + \mathbf{c}_j^*[n]\right]_q\right) \in \mathbb{Z}_q^{n+1}. \end{aligned}$$

Noise management.

Let $\mathbf{c}_i^*$ and $\mathbf{c}_j^*$ such that $\left[\langle \mathbf{c}_i^*, \mathbf{s}^* \rangle\right]_q = \Delta \cdot m_i + v_i$ and $\left[\langle \mathbf{c}_j^*, \mathbf{s}^* \rangle\right]_q = \Delta \cdot m_j + v_j$ be the ciphertexts of the messages m_i and m_j respectively, then the ciphertext which encrypts the message $m_i + m_j$ is $\mathbf{c}_{Add}^* = \mathbf{c}_i^* + \mathbf{c}_j^* \mod q$ and we have

$$\begin{aligned} \langle \mathbf{c}_{Add}^*, \mathbf{s}^* \rangle \mod q &= \langle \mathbf{c}_i^*, \mathbf{s}^* \rangle + \langle \mathbf{c}_j^*, \mathbf{s}^* \rangle \mod q \\ &= \Delta \cdot m_i + v_i + \Delta \cdot m_j + v_j \mod q \\ &= \Delta \left[m_i + m_j \right]_t + v_{Add} \mod q, \\ &\quad \text{with } v_{Add} = v_i + v_j - \epsilon t r \end{aligned}$$

where $\epsilon = q/t - \Delta = r_t(q)/t < 1$ and $m_i + m_j = \left[m_i + m_j\right]_t + tr, |r| \leq 1$. It follows that $|v_{Add}| \leq |v_i| + |v_j| + \epsilon t r$, which shows that the noise in the addition of ciphertext increases additively at most of the factor t.

6.4.2 Multiplication

The multiplication is more tricky, and necessitates a new treatment. In fact, we propose two variants, one directly based on FV multiplication, and another one, more complex, necessitating to add a projection key, as for BGV. We here detail the first solution, as this is the most efficient one. The second solution, of more theoretical interest, is only sketched here and the details are postponed to the appendices.

Managing multiplications.

Let $\mathbf{c}_i^*, \mathbf{c}_j^*$ be the ciphertexts of m_i and m_j respectively. Our proposed multiplication AtomicMult, executed on $\mathbf{c}_i^*$ and $\mathbf{c}_j^*$, is divided into three steps :

1. we modify $\mathbf{c}_i^*$ (resp. $\mathbf{c}_j^*$) to return to a unitary composite form ct_i (resp. ct_j), for which the first coefficient/slot corresponds to m_i (resp. m_j) ;
2. we apply the FV multiplication on ciphertexts ct_i and ct_j and obtain ct_{Mult} ;
3. we use our projection algorithm CipherProj on ct_{Mult} to obtain $\mathbf{c}_{Mult}^*$, which is exactly the ciphertext corresponding to the homomorphic multiplication related to the initial inputs $\mathbf{c}_i^*$ and $\mathbf{c}_j^*$.

We now give the details of each step.

Returning to unitary composite form Our first step consists in returning to the unitary composite form, which transformation is independently executed on the inputs $\mathbf{c}_i^*$ and $\mathbf{c}_j^*$ of the homomorphic multiplication.

More generally, we consider an atomic ciphertext $\mathbf{c}^* = (c_0, c_1, \cdots, c_n)$. Using our notation, we also have $\mathbf{s}^* = (1, s_0, s_1, \cdots, s_{n-1})$. Let $ct = (c_0, c_1 - c_n x - c_{n-1} x^2 - \cdots - c_2 x^{n-1})$ be a transitional Fan-Vercauteren ciphertext. The FV decryption procedure first consists in computing $ct(\mathbf{s}) = ct[0] + ct[1] \cdot \mathbf{s}$, where $\mathbf{s} = s_0 + s_1 x + \cdots + s_{n-1} x^{n-1}$ is the FV secret key. We then have

$$ct(\mathbf{s}) = \langle \mathbf{c}, \mathbf{s}^* \rangle + \sum_{i=0}^{n-1} \Big(\sum_{j=0}^{n-i-1} c_{j+1} s_{i+j} - \sum_{j=n-i}^{n-1} c_{j+1} s_{[i+j]_n} \Big) x^i$$

We then define the polynomial

$$\mathrm{P}(x) = \sum_{i=0}^{n-1} \Big(\sum_{j=0}^{n-i-1} c_{j+1} x^i \cdot s_{i+j} - \sum_{j=n-i}^{n-1} c_{j+1} x^i \cdot s_{i+j-n} \Big)$$

and we have $ct(\mathbf{s}) = \langle \mathbf{c}, \mathbf{s}^* \rangle + \mathrm{P}(x)$.

The idea is now to minimize this polynomial P in such a way that

$$\left[\left\lfloor \frac{t}{q} \cdot \mathrm{P}(x) \right\rceil\right]_t = 0.$$

For this purpose, we use a relinearisation key, denoted rok, and defined using the routines given in Section 6.2.2. Let $\omega > 1$ be an integer and let $\ell = \ell_\omega = \lceil \log_\omega q \rceil$. The relinearisation key rok is given by the couple $(\mathsf{rok}[0], \mathsf{rok}[1])$ where

$$\mathsf{rok}[0] = \begin{pmatrix} -(\mathbf{a}_0\mathbf{s} + \mathbf{e}_0) + \mathsf{PW}_\omega(s_0) \\ -(\mathbf{a}_1\mathbf{s} + \mathbf{e}_1) + \mathsf{PW}_\omega(s_1) \\ \vdots \\ -(\mathbf{a}_n\mathbf{s} + \mathbf{e}_n) + \mathsf{PW}_\omega(s_n) \end{pmatrix}, \text{ and } \mathsf{rok}[1] = \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{pmatrix},$$

where $\mathbf{a}_i = (a_{i,0}, a_{i,1}, \cdots, a_{i,\ell-1})$, $\mathbf{e}_i = (e_{i,0}, e_{i,1}, \cdots, e_{i,\ell-1})$ and s_i is the i-th coefficient of polynomial $\mathbf{s}$. Both $\mathsf{rok}[0]$ and $\mathsf{rok}[1]$ are $n \times \ell$ matrices with polynomial coefficients.

We have now all the material to describe our transformation of an atomic ciphertext $\mathbf{c} = (c_0, c_1, \cdots, c_n)$ into the unitary composite form $ct' = (ct'[0], ct'[1])$. We then define

$$\begin{aligned} ct'[0] &= c_0 - \sum_{i=1}^{n-1} \Big(\sum_{j=0}^{n-i-1} \mathsf{WD}_\omega(c_{j+1}x^i) \cdot \mathsf{rok}[0][i+j] - \\ &\qquad \sum_{j=n-i}^{n-1} \mathsf{WD}_\omega(c_{j+1}x^i) \cdot \mathsf{rok}[0][i+j-n] \Big) \end{aligned}$$

and

$$\begin{aligned} ct'[1] &= (c_1 - c_n x - c_{n-1}x^2 - \cdots - c_2 x^{n-1}) - \\ &\quad \sum_{i=1}^{n-1} \Big(\sum_{j=0}^{n-i-1} \mathsf{WD}_\omega(c_{j+1}x^i) \cdot \mathsf{rok}[1][i+j] - \\ &\qquad \sum_{j=n-i}^{n-1} \mathsf{WD}_\omega(c_{j+1}x^i) \cdot \mathsf{rok}[1][i+j-n] \Big) \end{aligned}$$

using the relinearisation key defined as above, and the subroutine WD given in Section 6.2.2. Such a transformation is executed on both $\mathbf{c}_i^*$ and $\mathbf{c}_j^*$. The correctness of the transformation is given below.

Applying FV homomorphic multiplication The second step consists in executing the FV homomorphic multiplication on the resulting ct'_i and ct'_j, and obtain ct_{Mult}. The details of such multiplication can be found in [FV12] and are not repeated here. One important point is that, using such method, we can directly make use of the polynomial multiplication optimizations already used in FV implementations.

Applying the projection The main point we have to manage with the resulting ciphertext ct_{Mult} is that it is in a composite form. But we can here simply use our projection procedure $\mathsf{CipherProj}(ct_{Mult}, 0)$, given in Section 6.3, to obtain our atomic ciphertext $\mathbf{c}^*_{Mult}$, which eventually corresponds to the multiplication of $\mathbf{c}_i^*$ and $\mathbf{c}_j^*$, as expected.

Noise management.

Let us consider a ciphertext ct' given by our above transformation of an atomic ciphertext $\mathbf{c}^*$. Let $s = s_0 + s_1 x + \cdots + s_{n-1}x^{n-1}$ be the FV secret key. We consider the first step of the FV decryption step :

$$ct'(\mathbf{s}) = ct'[0] + ct'[1] \cdot \mathbf{s}.$$

By applying the description of $ct'[0]$ and $ct'[1]$ given by equations (6.1) and (6.1) respectively, and considering that

$$\begin{aligned} \mathsf{rok}[0][i] + \mathsf{rok}[1][i] \cdot \mathbf{s} &= -(\mathbf{a}_i \mathbf{s} + \mathbf{e}_i) + \mathsf{PW}_\omega(s_i) + \mathbf{a}_i \cdot \mathbf{s} \\ &= -\mathbf{e}_i + \mathsf{PW}_\omega(s_i) \text{ and} \end{aligned}$$

$$\mathsf{WD}_\omega(c_j x^k)\big(\mathsf{rok}[0][i] + \mathsf{rok}[1][i] \cdot \mathbf{s}\big) = -\mathsf{WD}_\omega(c_j x^k)\mathbf{e}_i + c_j x^k \cdot s_i,$$

due to relations given in Section 6.2.2. It follows that

$$\begin{aligned} ct'(\mathbf{s}) &= \langle \mathbf{c}, \mathbf{s}^* \rangle + \sum_{i=1}^{n-1} \Big(\sum_{j=0}^{n-i-1} \mathsf{WD}_\omega(c_{j+1} x^i) \cdot \mathbf{e}_{i+j} - \\ &\qquad \sum_{j=n-i}^{n-1} \mathsf{WD}_\omega(c_{j+1} x^i) \cdot \mathbf{e}_{i+j-n} \Big) \end{aligned}$$

We then remark that

$$\mathsf{WD}_\omega(c_j x^k) \cdot \mathbf{e}_l = c_{j,0} \cdot e_{l,0} x^j + c_{j,1} \cdot e_{l,1} x^j + \cdots + c_{j,\ell-1} \cdot e_{l,\ell-1} x^j$$

where $c_i = c_{i,0} + c_{i,1}\omega + \cdots + c_{i,\ell-1}\omega^{\ell-1}$. Moreover, $e_{l,0} x^j$ is a rotation of the coefficients of $e_{l,0}$ of order j, then $e_{l,0} x^j \sim \chi$. It follows

$$\begin{aligned} \|\mathsf{WD}_\omega(c_j x^k)\mathbf{e}_l\|_\infty &= \ell \cdot \max_v \|c_{j,v} e_{l,v} x^j\| \\ &\leq \omega \cdot \mathrm{B} \cdot \ell \end{aligned}$$

where B is an upper limit on the distribution of the $e_{l,v}$.
We can conclude that the noise related to ct' is $\Delta m + v'$ with $\|v'\| \leq \|v\| + \omega \cdot \mathrm{B} \cdot \ell n^2$ when $\langle \mathbf{c}, \mathbf{s}^* \rangle = \Delta m + v$.

6.4.3 Noise Management

Let $\langle \mathbf{c}_i^*, \mathbf{s}^* \rangle = \Delta m_i + \mathbf{v}[i] + r_i q$ and $\langle \mathbf{c}_j^*, \mathbf{s}^* \rangle = \Delta m_j + \mathbf{v}[j] + r_j q$, with $|\mathbf{v}[k]| < \mathrm{E}, k = i, j$. We denote by h the Hamming weight of the secret key sk. We can show that $|r_k| < 1 + \frac{h+1}{2}\|sk\|$ for $k = i, j$. We also consider the following relations :

$$\begin{aligned} [m_i]_t \left[m_j\right]_t &= t \cdot r_m + \left[m_i m_j\right]_t, \\ &\quad \text{with} |r_m| \leq \frac{1}{t}\left(\frac{t^2}{4} + \frac{t}{2}\right) < \frac{t}{2}; \\ \mathbf{v}[i]\mathbf{v}[j] &= t \cdot r_v + \left[\mathbf{v}[i]\mathbf{v}\left[j\right]\right]_\Delta, \\ &\quad \text{with} |r_v| \leq \frac{1}{\Delta}\left(\left|\mathbf{v}[i]\mathbf{v}\left[j\right]\right| + \frac{\Delta}{2}\right) < \frac{\mathrm{E}}{2}. \end{aligned}$$

Let $\mathbf{c}_{\tilde{Mult}}$ to be the coefficients of polynomial $\mathrm{P}(\mathbf{y})$ as defined in the AtomicMult algorithm. We note

$$\tilde{\mathbf{c}} = \left(\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{Mult}}\right) \cdot \mathsf{rdk}[0], \mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{Mult}}\right) \cdot \mathsf{rdk}[1]\right),$$

and then

$$\begin{aligned} \mathbf{c}^*_{Mult} &= \left\lfloor \frac{t}{q}\tilde{\mathbf{c}} \right\rceil \Rightarrow \frac{t}{q}\tilde{\mathbf{c}} = \mathbf{c}^*_{Mult} + \mathbf{r}_0, \\ &\quad \text{with } |\mathbf{r}_0[k]| < \frac{1}{2} \quad \text{for all } k. \end{aligned}$$

We can now compute

$$\begin{aligned}
\langle \mathbf{c}^*_{\mathrm{M}ult}, \mathbf{s}^* \rangle &= \frac{t}{q}\langle \tilde{\mathbf{c}}, \mathbf{s}^* \rangle - \langle \mathbf{r}_0, \mathbf{s}^* \rangle \\
&= \frac{t}{q}\left(\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{\mathrm{M}ult}}\right) \cdot \left(-\left(\mathbf{A}\mathbf{s}+\mathbf{e}\right) + \mathsf{PW}_\omega\left(\mathbf{s}'\right)\right)\right) \\
&\quad + \frac{t}{q}\left(\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{\mathrm{M}ult}}\right) \cdot \mathbf{A}\mathbf{s}\right) - \mathbf{r}_0\mathbf{s}^* \\
&= \frac{t}{q}\left(\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{\mathrm{M}ult}}\right) \cdot \mathsf{PW}_\omega\left(\mathbf{s}'\right)\right) \\
&\quad - \frac{t}{q}\left(\mathsf{WD}_\omega(\mathbf{c}_{\tilde{\mathrm{M}ult}}) \cdot \mathbf{e}\right) - \mathbf{r}_0\mathbf{s}^* \\
&= \frac{t}{q}\left(\mathbf{c}_i^*\mathbf{s}^*\mathbf{c}_j^*\mathbf{s}^* - \mathsf{WD}_\omega(\mathbf{c}_{\tilde{\mathrm{M}ult}}) \cdot \mathbf{e}\right) - \mathbf{r}_0\mathbf{s}^* \\
&= \frac{t}{q}\left(\mathbf{c}_i^*\mathbf{s}^*\mathbf{c}_j^*\mathbf{s}^*\right) - r_a, \\
&\quad \text{with } r_a = \frac{t}{q}\left(\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{\mathrm{M}ult}}\right) \cdot \mathbf{e}\right) + \mathbf{r}_0\mathbf{s}^*.
\end{aligned}$$

The vectors $\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{\mathrm{M}ult}}\right)$ and $\mathbf{e}$ are of size $\mathrm{N} = \frac{(n+1)(n+2)}{2} \times l_\omega$, where $l_\omega = \lfloor \log_\omega q \rceil$. The coefficients of $\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{\mathrm{M}ult}}\right)$ are bounded by $\omega/2$ in absolute value, those of $\mathbf{e}$ by B. The right term of the above equation r_a can be bounded by :

$$\begin{aligned}
|r_a| &\leq \frac{t}{q}|\mathsf{WD}_\omega\left(\mathbf{c}_{\tilde{\mathrm{M}ult}}\right) \cdot \mathbf{e}| + |\mathbf{r}_0\mathbf{s}^*| \\
&\leq \frac{t}{q}\frac{\mathrm{N}}{2}\frac{\omega}{2}\mathrm{B} + (h+1)\frac{1}{2}\|sk\| \\
&\leq \frac{\mathrm{N}}{4q}t\omega\mathrm{B} + \frac{h+1}{2}, \text{ for } \|sk\| = 1.
\end{aligned}$$

We now focus on the term in the calculation of $c_{mult}\mathbf{s}^*$:

$$\begin{aligned}
\frac{t}{q}\left(\mathbf{c}_i^*\mathbf{s}^*\mathbf{c}_j^*\mathbf{s}^*\right) &= \Delta\left[m_i m_j\right]_t + \left(m_i\mathbf{v}[j] + m_j\mathbf{v}[i]\right) \\
&\quad + t\left(\mathbf{v}[i]\mathbf{r}[j] + \mathbf{v}[i]\mathbf{r}[j]\right) + r_v \\
&\quad + \left(q - r_t(q)\right)\left(r_m + m_i\mathbf{r}[j] + m_j\mathbf{r}[i]\right) \\
&\quad + qt\mathbf{r}[i]\mathbf{r}[j] + \frac{t}{q}\left[\mathbf{v}[i]\mathbf{v}[j]\right]_\Delta \\
&\quad - \frac{r_t(q)}{q}\left(\Delta m_i m_j + \left(m_i\mathbf{v}[j] + m_j\mathbf{v}[i]\right) + r_v\right).
\end{aligned}$$

By noting by r_r the term corresponing to the last line in the above equation, we have :

$$\begin{aligned}
\langle \mathbf{c}^*_{\mathrm{M}ult}, \mathbf{s}^* \rangle &= \Delta\left[m_i m_j\right]_t + \left(m_i\mathbf{v}[j] + m_j\mathbf{v}[i]\right) \\
&\quad + t\left(\mathbf{v}[i]\mathbf{r}[j] + \mathbf{v}[i]\mathbf{r}[j]\right) \\
&\quad + r_v + \left(q - r_t(q)\right)\left(r_m + m_i\mathbf{r}[j] + m_j\mathbf{r}[i]\right) \\
&\quad + qt\mathbf{r}[i]\mathbf{r}[j] + r_r - r_a.
\end{aligned}$$

As $\langle \mathbf{c}_{mult}, \mathbf{s}^* \rangle$ is an integer, then $r_r - r_a$ is also an integer. We can write

$$\begin{aligned}
\left[\langle \mathbf{c}^*_{\mathrm{M}ult}, \mathbf{s}^* \rangle\right]_q &= \Delta\left[m_i m_j\right]_t + v_{mult} \text{ where} \\
v_{mult} &= \left(m_i\mathbf{v}[j] + m_j\mathbf{v}[i]\right) + t\left(\mathbf{v}[i]\mathbf{r}[j] + \mathbf{v}[i]\mathbf{r}[j]\right) \\
&\quad + r_v - r_t(q)\left(r_m + m_i\mathbf{r}[j] + m_j\mathbf{r}[i]\right) \\
&\quad + r_r - r_a.
\end{aligned}$$

This equation shows that $\mathbf{c}^*_{\mathrm{M}ult}$ is the ciphertext which encrypts the message $m_i \times m_j$. We

can then compute a bound related to the noise in this multiplication algorithm as follows :

$$\begin{aligned}
|v_{mult}| &\leq \left|m_i\mathbf{v}[j]+m_j\mathbf{v}[i]\right|+t\left|\mathbf{v}[i]\mathbf{r}[j]+\mathbf{v}[i]\mathbf{r}[j]\right| \\
&\quad +|r_v|-r_t(q)\,|r_m|+\left|m_i\mathbf{r}[j]+m_j\mathbf{r}[i]\right|+|r_r|+|r_a| \\
&\leq \frac{t}{2}\left(|\mathbf{v}[i]|+|\mathbf{v}[j]|\right)+t\left(\frac{h+1}{2}+1\right)\left(|\mathbf{v}[i]|+|\mathbf{v}[j]|\right) \\
&\quad +\frac{\mathrm{E}}{2}+r_t(q)\left(\frac{t}{2}+t\left(\frac{h+1}{2}+1\right)\right)+\frac{1}{2}+r_t(q)\,t \\
&\quad +\frac{\mathrm{N}}{4q}\,t\omega\mathrm{B}+\frac{h+1}{2} \\
&\leq t\mathrm{E}+t\mathrm{E}\left(\frac{h+1}{2}+1\right)+\frac{\mathrm{E}}{2} \\
&\quad +t^2\left(\frac{3}{2}+\frac{h+1}{2}\right)+\frac{1}{2}+t^2+\frac{\mathrm{N}}{4q}\,t\omega\mathrm{B}+\frac{h+1}{2} \\
&\leq \mathrm{E}\left(t\left(\frac{h+1}{2}+2\right)+\frac{1}{2}\right) \\
&\quad +t^2\left(2+\frac{h}{2}\right)+1+\frac{h}{2}+\frac{\mathrm{N}}{4q}\,t\omega\mathrm{B}.
\end{aligned}$$

We can see that this bound does not include the expansion factor as in FV, and that the noise does not grow quadratically, but it is multiplied by the plaintext modulus and the Hamming weight of secret key, which is better.

Remark 1 *The previous operation may also apply to the projections of ciphertexts derived from two different ciphertexts of FV, but encrypted with the same public key. Namely, if* $ct = Encrypt(\boldsymbol{m}, pk), ct' = Encrypt(\boldsymbol{m}', pk)$ *and* $\boldsymbol{c}_i^* = CipherProj(ct, i), \boldsymbol{c}_j'^* = CipherProj(ct', j)$ *for arbitrary* i, j*, then the outputs of* $AtomicAdd\left(\boldsymbol{c}_i^*, \boldsymbol{c}_j'^*\right)$ *and* $AtomicMult\left(\boldsymbol{c}_i^*, \boldsymbol{c}_j'^*, rdk\right)$ *are the ciphertexts of plaintexts* $m_i + m_j'$ *and* $m_i \times m_j'$ *respectively.*

6.4.4 Another Mutliplication on Atomic Ciphertexts

The multiplication is much more tricky, and necessitates a new treatment.

Overview

In fact, with our modification of the ciphertext structure, multiplication can no longer be carried out as defined in the FV scheme. We will therefore present, in this section, another way to do the multiplication of our ciphertexts. We first remark that the vector of our ciphertexts has a form that is analogous to the representation of ciphertexts in the BGV scheme [BGV12]. We then propose a way to perform homomorphic multiplications on atomic ciphertexts using a method close to the one introduced in the BGV scheme.

More precisely, as in the major part of somewhat homomorphic schemes, our multiplication will be made in two steps : the first one is the multiplication of initial ciphertexts which extends the dimension of our resulting ciphertext, the second one is a projection step which uses one resizing key to get the output ciphertext back to the right space dimension. Before presenting our multiplication algorithm, we first introduce our key projection algorithm.

Projection key

To compute the projection key which will be used for the multiplication, we propose to use the BGV SwitchKeyGen algorithm.

Let $\omega > 1$ be an integer. We recall that $\mathbf{s}^* \in \mathbb{Z}_q^{(n+1)}$ is the decryption key for atomic ciphertext. We therefore proceed as follows.

1. We fixe $\mathbf{s}' \leftarrow \mathbf{s}^* \otimes \mathbf{s}^*$ as the tensorial product of vector $\mathbf{s}^*$. Then $\mathbf{s}'$ is a vector of size $\frac{(n+1)(n+2)}{2}$ (see KeyGen of BGV).
2. Now, we use the same type of operations as in algorithm SwitchKeyGen of BGV with $\mathbf{s}_1 = \mathbf{s}'$ and $\mathbf{s}_2 = \mathbf{s}$:
   — sample a matrice $\mathbf{A} \leftarrow \mathbb{Z}_q^{\mathrm{N}\times n}$ uniformly and a vector $\mathbf{e} \leftarrow \chi^{\mathrm{N}}$, where $\mathrm{N} = \frac{(n+1)(n+2)}{2} \cdot \lceil \log_2(q) \rceil$;
   — and compute the projection key noted rdk as follows

$$\mathsf{rdk}[0] = \left(\left[-(\mathbf{A}\mathbf{s} + \mathbf{e}) + \mathsf{PW}_\omega(\mathbf{s}') \right]_q \right) \in \mathbb{Z}_q^{\mathrm{N}}$$
$$\mathsf{rdk}[1] = \mathbf{A} \in \mathbb{Z}_q^{\mathrm{N}\times n}.$$

   Where $\mathbf{s} = (\mathbf{s}^*[1], \ldots, \mathbf{s}^*[n]) \in \mathbb{Z}_q^n$.

Managing multiplications

Let $\omega > 1$ be an integer and $\mathbf{c}_i^*, \mathbf{c}_j^*$ be the ciphertexts of m_i and m_j respectively. Our new multiplication algorithm AtomicMult takes as input both ciphertexts $\mathbf{c}_i^*$ and $\mathbf{c}_j^*$, together with the projection key rdk defined as previously. We then proceed as follows :

1. set $\mathbf{c}_{\tilde{\mathrm{M}ult}}$ to be the coefficents of product

$$\mathrm{P}(\mathbf{y}) = \langle \mathbf{c}_i^*, \mathbf{y} \rangle \cdot \langle \mathbf{c}_j^*, \mathbf{y} \rangle$$

   $\mathbf{y}$ is a vector of size $n+1$. We can organize the coeffients of $\mathrm{P}(\mathbf{y})$ such that the vector $\mathbf{c}_{\tilde{\mathrm{M}ult}}$ is an element of $\mathbb{Z}_q^{\frac{n(n+1)}{2}}$;
2. at this step, we get back the size of our ciphertext from $\frac{(n+1)(n+2)}{2}$ to n. To achieve this, we use the key rdk defined above during the projection key procedure. We then make the following calculation to do the resizing, and then gives us the ciphertext which correspond to the multiplication of our messages m_i, m_j :

$$\mathbf{c}_{\mathrm{M}ult}^* = \left\lfloor \frac{t}{q} \left(\mathsf{WD}_\omega(\mathbf{c}_{\tilde{\mathrm{M}ult}}) \cdot \mathsf{rdk}[0], \mathsf{WD}_\omega(\mathbf{c}_{\tilde{\mathrm{M}ult}}) \cdot \mathsf{rdk}[1] \right) \right\rceil .$$

It is important to note that, in the last equation, the first product is a product between two vectors of size N, and that the second one is the product between a N-size vector and $\mathrm{N} \times n$-size matrix. This gives us a n-dimensional vector. Finally, the size of $\mathbf{c}_{\mathrm{M}ult}^*$ is obviously equal to $(n+1)$.

Remark 2 *Managing multiplications procedure is the same as that used in the algorithm BGV.Mult. Indead, the first step corresponds to its first step and the second one to algorithm BGV.KeySwitch for which we added the multiplication by $\frac{t}{q}$ and the operation of rounding to nearest integer $\lfloor \cdot \rceil$. As we work with only one modulus q, we don't need to use algorithm BGV.SwitchModuli.*

6.5 From Atomic back to Composite Ciphertexts

In this section, we try to return our atomic ciphertexts to the form of FV ciphertext representation ; namely the polynomial form. The problem that we want to solve can be presented as follows.

Problem

Given r atomic ciphertexts $\mathbf{c}_0^*, \mathbf{c}_1^*, \cdots \mathbf{c}_{r-1}^*$, which encrypt the messages $m_0, m_1, \cdots m_{r-1}$ respectively under the secret key $\mathbf{s}^* = (1, s_0, s_1, \cdots, s_{n-1})$ where $\mathbf{s} = sk$ the FV's secret key, and $\mathbf{c}_i^* = (c_{i0}, c_{i1}, \cdots c_{in})$. How to transform these atomic ciphertexts such that, they give us a FV polynomial ciphertext which encrypt the plaintext $m_0 + m_1 x + \cdots + m_{r-1} x^{r-1}$?

To answer this issue, we have establishment three different strategies. One has already been given in Section 6.4.2 and consists in returning to a unitary composite form. We now detail the two other solutions

6.5.1 Retuning to Global Composite Form

This strategy is more as a proof that the above problem is solvable. It consists of apply another layer of encryption on the atomic ciphertext, and use one technique as the *bootstrapping* technique on these over ciphertext. The items of this strategy are the following.

— at the step of key generation in the FV scheme, adds as other parameter the encryption of coefficients of secret key sk by one public key pk. We make the assumption here that the circular security assumption holds. Then, we choose pk from the same secret key sk. The encrypted coefficients of the secret key are noted by $[1]_{pk}, [s_0]_{pk}, [s_1]_{pk}, \ldots, [s_{n-1}]_{pk}$;

— for each atomic ciphertext $\mathbf{c}_i^* = (c_{i0}, c_{i1}, \cdots c_{in})$, computes the FV encryption of its coefficients c_{ij} which is noted $\left[c_{ij}\right]_{pk}$ for i from 0 to $r-1$ and j from 0 to n ;

— computes the polynomials

$$\mathrm{P}_i = [c_{i0}]_{pk}\,[1]_{pk} + [c_{i1}]_{pk}\,[s_0]_{pk} + \cdots + [c_{in}]_{pk}\,[s_{n-1}]_{pk}$$

for i from 0 to $r-1$; and computes the compact polynomial

$$\mathrm{P} = \sum_{i=0}^{r-1} \mathrm{P}_i x^i;$$

— assuming we are able to homomorphically make the following operations $[\cdot]_q$, times by t/q, $\lfloor \cdot \rceil$ and $[\cdot]_t$, computes the polynomial noted

$$\mathrm{Q} = \left[\left\lfloor \frac{t}{q} [\mathrm{P}]_q \right\rceil \right]_t.$$

The polynomial Q is a new FV ciphertext which encrypts the plaintext $m_0 + m_1 x + \cdots + m_{r-1} x^{r-1}$ under the public key pk. Then, this ciphertext can be handled and decrypted as an original FV ciphertext.

This strategy is directly conditioned by the computation of operations $[\cdot]_q$, *times by* t/q, $\lfloor \cdot \rceil$ and $[\cdot]_t$ in the homomorphic domain. Unfortunately, it is these operations which make bootstrapping very difficult and slow. It is why we do not implement this technique here. We only use it as a theoretical prove that we can return in the polynomial form.

6.5.2 Returning to Compact Composite Form

The problem for application of the first strategy is that some operations are complicated and costly to be evaluated in the homomorphic domain. In this subsection, we propose a strategy which tries to avoid these costly operations. This strategy is inspired from the first strategy, but instead of returning a ciphertext which can be handled and decrypted by FV scheme, we come back to a ciphertext which compacts all the messages $m_0, m_1, \ldots m_{r-1}$. This new ciphertext cannot be handled and decrypted by the basic decryption algorithm of FV. To recover our messages $m_0, m_1, \ldots m_{r-1}$ from this compact ciphertext, we will define a new algorithm to decrypt it.

To explain our strategy, we need to define a new ciphertext space $\mathrm{R}_{q'}$, still under the cyclotomic polynomial x^n+1, but with a modulus $q^{'} > q$ (see below for a discussion on possible choices for q and $q^{'}$). From the secret key sk, we define a public key $pk^{'}$ in $\mathrm{R}_{q'}$. Under the circular security assumption, we use this new public key to encrypt coefficients of the secret key $[1]_{pk'}$, $[s_0]_{pk'}$, $[s_1]_{pk'}$,..., $[s_{n-1}]_{pk'}$. This encrypted secret key then helps us to compact our atomic ciphertexts into a single polynomial ciphertext.

To compact our atomic ciphertext into a single polynomial one, we proceed as follows :

— from the encrypted ciphertext of coefficients of the secret key $[1]_{pk'}$, $[s_0]_{pk'}$, $[s_1]_{pk'}$, ..., $[s_{n-1}]_{pk'}$, computes the polynomials

$$\begin{aligned} \mathrm{P}_i &= c_{i,0}\,[1]_{pk'} + c_{i,1}\,[s_0]_{pk'} + \cdots + c_{i,n}\,[s_{n-1}]_{pk'} \\ &= \left[c_{i,0} + c_{i,1}s_0 + \cdots + c_{i,n}s_{n-1}\right]_{pk'} \in \mathrm{R}_{q'}; \end{aligned}$$

— constructs the compact polynomial as in the first strategy, as

$$\mathrm{P} = \sum_{i=0}^{r-1} \mathrm{P}_i x^i \in \mathrm{R}_{q'}.$$

The compact polynomial P holds all information about the messages m_0, $m_1,\ldots,m_{r-1}$, because it holds all about atomic ciphertexts $\mathbf{c}_0^*, \mathbf{c}_1^*, \cdots \mathbf{c}_{r-1}^*$. But a traditional application of the FV decryption algorithm does not allow to get back the messages $m_0, m_1, \cdots, m_{r-1}$. We then have to define a new decryption algorithm, which is given by Figure 6.3.

DecryptCompact$\left(\mathrm{P}, sk, q^{'}\right)$:
— $\mathrm{Q} \leftarrow$ FV.Decrypt$\,(sk, \mathrm{P})$ for modulus $q^{'}$;
— return $\left[\left\lfloor \frac{t}{q}\,[\mathrm{Q}]_q \right\rceil\right]_t$.

FIGURE 6.3 – Decryption of Compact Ciphertext

Correctness.

In fact, for a good choose of the modulus $q^{'}$, the decryption of compact polynomial P is possible and this give us the polynomial

$$\mathrm{Q} = \sum_{i=0}^{r-1} \left(c_{i0} + c_{i1}s_0 + c_{i2}s_1 + \cdots + c_{in}s_{n-1}\right) x^i.$$

The polynomial Q matches to the result of the first step of decryption algorithm. The only way we have to do is completed the decryption algorithm by the operations that produced us the problem in the first strategy. That what we do at the second item of the DecryptCompact algorithm.

About choose of $q^{'}$.

Algorithm DecryptCompact works only if we choose a *good* modulus $q^{'}$. Then the question is : how choose a such modulus parameter? In the lemma 1 of FV [FV12] it is show that a ciphertext $ct = (c_0, c_1)$ can be decrypted only if $c_0 + c_1 \cdot s = \Delta\mathbf{m} + \mathbf{v} \mod q$ and $\left(\frac{t}{q}\right)\|\mathbf{v} + \epsilon\mathbf{m}\| < 1/2$. where $\epsilon = q/t - \Delta$ and $\|\mathbf{v}\| < 2\delta_{\mathrm{R}}\mathrm{B}^2 + \mathrm{B}$. We use this lemma and the two following remaks to compute the modulus $q^{'}$.

Lemma 1 Noise in external multiplication
Lets t (resp. q) the plaintext (resp. ciphertext) modulus of FV scheme, and p a integer such that $p \leq t/2$. Let $ct = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ a FV ciphertext of message m under the secret key $\boldsymbol{s}$, i.e. $\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \Delta m + \boldsymbol{v} \mod q$. The noise in the ciphertext $ct' = p \times ct$ is $\boldsymbol{v}' = \boldsymbol{v} + t\boldsymbol{r}$, where $\|\boldsymbol{r}\| \leq (p+1)/2$.

Proof 1 *As $ct = (\boldsymbol{c}_0, \boldsymbol{c}_1)$, we have*

$$\begin{aligned} \boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} &= \Delta [\boldsymbol{m}]_t + \boldsymbol{v} \mod q \\ \Rightarrow \quad & p(\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) = \Delta p [\boldsymbol{m}]_t + p\boldsymbol{v} \mod q \\ \Rightarrow \quad & p(\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}) = \Delta [p\boldsymbol{m}]_t + p\boldsymbol{v} + t\boldsymbol{r} \mod q \end{aligned}$$

where $p[\boldsymbol{m}]_t = [p\boldsymbol{m}]_t + t\boldsymbol{r} \Rightarrow \|\boldsymbol{r}\| \leq \frac{1}{t}\left(p\frac{t}{2} + \frac{t}{2}\right) = \frac{1}{2}(p+1)$.
Then, we obtain

$$\|\boldsymbol{v}'\| \leq \|\boldsymbol{v}\| + \frac{1}{2}(p+1)$$

.

Lemma 2 Noise in the sum of n ciphertexts
Let t, q as the lemma 1 and $ct_0, ct_1, \ldots, ct_{n-1}$ n FV-ciphertext of the messages $m_0, m_1, \ldots m_{n-1}$ respectively under the secret key $\boldsymbol{s}$, i.e. for all i in range $\{0, 1, \ldots, n-1\}, ct_i = (\boldsymbol{c}_{i0}, \boldsymbol{c}_{i,1})$ and $\boldsymbol{c}_{i0} + \boldsymbol{c}_{i1} \cdot \boldsymbol{s} = \Delta m_i + \boldsymbol{v}_i \mod q$.

Then, the noise in the ciphertext $\sum_{i=0}^{n-1} ct_i$ is $\boldsymbol{v} = \sum_{i=0}^{n-1} (\boldsymbol{v}_i + \epsilon t \boldsymbol{u}_i)$, where $\|\boldsymbol{u}_i\| \leq 1$.

Proof 2 *For two ciphertexts ct_1, ct_2 such that $ct_i = (\boldsymbol{c}_{i0}, \boldsymbol{c}_{i1})$ and $\boldsymbol{c}_{i0} + \boldsymbol{c}_{i1}\boldsymbol{s} = \Delta[\boldsymbol{m}_i]_t + \boldsymbol{v}_i \mod q$, the addition noise is $\boldsymbol{v}_{Add} = \boldsymbol{v}_1 + \boldsymbol{v}_2 + \epsilon t \boldsymbol{r}$ where $\epsilon < 1$ and $[\boldsymbol{m}_1]_t + [\boldsymbol{m}_2]_t = [\boldsymbol{m}_1 + \boldsymbol{m}_2]_t + t\boldsymbol{r} \Rightarrow \|\boldsymbol{r}\| \leq 1$.*

Applying recursively this formula, we obtain

$$\boldsymbol{v} = \sum_{i=0}^{n-1} (\boldsymbol{v}_i + \epsilon t \boldsymbol{u}_i), \text{ where } \|\boldsymbol{u}_i\| \leq 1.$$

With these two lemmas, and the lemma 1 of FV [FV12], we able now to deduce the relation between modulus q' and q such that the compact ciphertext P can be decrypted. In fact, if we note $\mathbf{v}_{\mathrm{P}_i}$ the noise in the polynomials P_i, we have $\left(\frac{q}{q'}\right)\|\mathbf{v}_{\mathrm{P}_i} + \epsilon \mathbf{m}'\| < 1/2 \Rightarrow q' > 2q\|\mathbf{v}_{\mathrm{P}_i} + \epsilon \mathbf{m}'\|$,
where $\mathbf{m}' = [c_{i0} + c_{i1}s_0 + c_{i2}s_1 + \cdots + c_{in}s_{n-1}]_q$ which is bounded by $q/2$, $\epsilon = q'/q - \lfloor q'/q \rfloor$. Combining the above two lemma, we have

$$\|\mathbf{v}_{\mathrm{P}_i}\| \leq n\left(\left(2\delta_{\mathrm{R}_q}\mathrm{B}^2 + \mathrm{B}\right) + a + q^2/2\right)$$

this allows us to deduce that the polynomials P_i are decrypted only if the modulus q^2 satisfy the condition

$$q' > 2qn\left(2\delta_{\mathrm{R}'_q}\mathrm{B}^2 + \mathrm{B} + q + q^2/2\right).$$

As the compact polynomial P is the sum of polynomial P_i, the noise does not increase significantly and then we can deduce that the decryption condition of P_i is the same to P.

6.6 Practical Considerations

The techniques presented in this chapter are bent towards increasing the communication efficiency of FHE. In terms of communication efficiency, we wish to compare

our techniques with the conceptually different approach of stream-cipher-based transciphering [CCF+16]. Rather than sending large-size FHE-encrypted data, this technique consists in sending data xored with some keystream and feeding these data into the FHE domain by an homomorphic-xoring with the same keystream but this time precomputed homomorphically. Of course this latter technique assumes knowledge of a FHE-encryption of the secret stream-cipher key which needs to be transmitted during setup and is then amortized in terms of communication. However, despite of its practical relevance, transciphering has two main drawbacks : 1/ it increases the multiplicative depth at which the FHE computations have to take place (to absorb the homomorphic keystream calculations) and 2/ it works only when feeding data into the FHE domain.

6.6.1 Choosing Parameters

As an example to illustrate our point, let us assume that we have to perform a depth 4 FHE computation and that transciphering induces an extra depth of 4 (which is realized for example by FLIP [PAFXC16] which is presently the lowest multiplicative-depth symmetric primitive known). Thus we will use the parameters given in Table 6.1 which were derived from the noise growth analysis of [FV12] and checked against the lwe-estimator [1] of [RAMS15, Alb17] against a $\lambda >> 100$ security parameter target. Note that these parameters are consistent with those of SEAL ([HKR17], Table 4).

	$\times$-depth	n	$\log_2 t$	$\log_2 q$	α
w/o transciphering	4	4096	1	112	$8/q$
with transciphering	8	8192	1	174	$8/q$

TABLEAU 6.1 – FV parameter sets for a depth 4 FHE calculation without (depth 4) and with (depth 8) transciphering. See text.

6.6.2 Communication overhead estimations

Since our technique enables the sending of dense composite ciphertexts, for a depth 4 calculation, the overhead will be a factor 112 (which simply corresponds to the modulus size). Conversely, if we use tranciphering naively, we have to first transmit 128 depth-8 ciphertexts, each of which encoding a single message, hence around 22 Mbytes of data before we can transmit the rest of the (payload) data with no further (communication) overhead. In this setting, it thus takes around 400 4096-bit blocks of data before transciphering becomes more efficient than using batching as proposed in this chapter. Still, our technique can also be combined with transciphering in order to lower the cost of transmitting the bits of the FHE-encrypted key. In our illustrative case, we can transmit a depth-8 8192-bit block (with only 128 active slots) for a communication cost of around 174 kbytes before we can transmit the rest of the data with no further overhead. In this setting, it takes only around 4 4096-bit blocks for transciphering (paired with this chapter techniques) to be more efficient showing that the two techniques can be combined nearly optimally.

Quite interestingly, the techniques developped in this chapter also help with respect to decreasing the FHE output overhead for which *transciphering is not applicable* (due to the all-or-nothing nature of FHE-decryption). Still, the effective output overhead depends on the strategy implemented in order to reassemble a set of atomic ciphertexts into a denser composite one. The first strategy described in Sect. 6.5.1 allows to end up with the same overhead in output as in input since it has the ability to turn a set of atomic ciphertexts back to the initial FV ciphertext representation, albeit at large extra

1. `bitbucket.org/malb/lwe-estimator/overview` (commit : 6295aa5).

computational cost. More practically (in terms of computational overhead), the strategy in Sect. 6.5.2, allows to switch back to a new composite representation but relative to a larger modulus leading to an overhead in output which is 3 times larger then for the input. For example, following Table 6.1, for a depth 4 calculation, we would switch from a 112-bit module in input to a 336-bit module in output. This also diminishes the interest of pairing our techniques with transciphering *for application setups where the output overhead needs to be optimized* as in that case the output modulus would be on 522-bits (starting from an input modulus on 174 bits to cope with the multiplicative depth increase of tranciphering as already stated).

Conclusion

Nous avons étudié dans ce mémoire les prémisses de la compression et de la manipulation d'images et de vidéos dans le domaine homomorphe, donnant ainsi une base pour la protection de ce type d'informations grâce à l'outil prometteur de la cryptographie avancée qu'est le chiffrement homomorphe.

Nous avons abordé cet outil aujourd'hui majeur en cryptographie en faisant tout d'abord un rappel et un tracé de l'évolution du chiffrement homomorphe dans le temps. Nous avons continué avec une présentation des solutions qui sont déployées de nos jours et que nous avons utilisé dans nos implémentations. Nous avons aussi rappelé la preuve selon laquelle il existe une machine de Turing homomorphe c'est-à-dire une machine capable, en théorie, de simuler le fonctionnement de tout algorithme sur des entrées chiffrées.

Sachant désormais qu'il est possible de simuler le fonctionnement d'un compresseur vidéo en homomorphe, nous nous sommes ensuite focalisés sur l'étude et l'analyse des blocs algorithmiques qui constituent un tel compresseur (transformation RGB vers YCrCb, DCT, etc...). Nous avons décidé de commencer par transformer de façon isolée ces différents blocs en leurs équivalents homomorphes. C'est ainsi qu'au chapitre 4, nous avons commencé par l'algorithme de compression RLE. L'implémentation de cet algorithme lorsqu'on manipule les données non chiffrées ne nécessite pas plus de 5 lignes de code. Cela n'est pas le cas lorsque les données deviennent chiffrées. Nous avons donc mis en exergue l'une des difficultés applicative du chiffrement homomorphe qui est la transformation des algorithmes classiques vers le mode chiffré où on ne voit rien de ce qu'on manipule. Nous avons aussi proposé des améliorations de notre circuit RLE homomorphe qui nous a permis de mettre en avant plusieurs techniques qui peuvent s'appliquer à d'autres types de circuits homomorphes.

Nous avons ensuite groupé l'étude sur les transformations des autres blocs algorithmiques du compresseur dans le chapitre 5. Au cours de cette opération, nous avons obtenu les résultats suivants. Tout d'abord, puisque les outils offerts par le chiffrement homomorphe dépendent de la représentation des messages, il était important de faire un rappel des deux principales représentations utilisées pour l'application du chiffrement homomorphe (représentation en binaire et entière ou polynômiale) ainsi que les opérations arithmétiques qui leur sont associées. L'ojectif de l'analyse de ces deux types de représentations était de pouvoir ensuite mieux adapter les algorithmes que nous allions manipuler. En effet, les algorithmes de la transformation spatiale d'une vidéo (transformation RGB vers YCrCb, DCT et quantization ainsi que leurs inverses) manipulent des nombres réels dans leur définition de base. Dans le domaine clair, on dispose déjà d'architectures permettant de facilement les manipuler. Cela n'est pourtant pas le cas en homomorphe, surtout lorsqu'on veut des solutions efficaces. Nous avons alors proposé des solutions algorithmiques utilisant des entiers en entrée qui sont beaucoup plus adaptées au chiffrement homomorphe et les avons implémentées dans notre solution. Nous avons également proposé des solutions déduites des algorithmes des standards H264 et HEVC qui ont eu aussi une représentation entière et s'accordent assez bien avec le chiffrement homomorphe. Nous avons finalement mis tous ces blocs transformés ensemble et proposé ce qui est à notre connaissance le premier pipeline de compression d'une

image utilisant les algorithmes de chiffrement homomorphe.

Au Chapitre 6, nous nous sommes enfin intéressés à la problématique de l'expansion des chiffrés par rapport à leurs messages clairs associés dans les schémas de chiffrement homomorphe. Nous avons proposé une solution qui permet d'utiliser les slots d'un chiffré au maximum ainsi que des algorithmes permettant d'extraire chacun des slots d'un chiffré et de le manipuler indépendamment des autres. Nous avons également proposé de nouvelles opérations homomorphes pour manipuler ces slots, parmi lesquelles une qui permet de revenir à un chiffré composite à partir de plusieurs slots indépendants. Les transformations que nous proposons dans ce chapitre se greffent facilement aux schémas de chiffrement sur les réseaux idéaux et préservent également leur sécurité.

Perspectives

A ce jour, lorsqu'on associe compression et cryptographie homomorphe dans une même phrase, cela semble à première vue paradoxal. En effet l'un des gros problèmes que rencontre le chiffrement homomorphe est la taille des clés et des chiffrés qui est liée à sa sécurité. Des travaux ont déjà permis de réduire de manière importante cette taille en définissant la sécurité non plus sur les problèmes des réseaux euclidiens, mais ceux des réseaux idéaux, réduisant ainsi la taille de l'ordre d'une matrice à celle d'un polynôme. La prochaine étape pourrait-elle être un passage de l'échelle d'un polynôme à celle d'un coefficient ? C'est grâce à la recherche qu'on pourra peut-être un jour répondre à cette question.

Les travaux que nous avons menés sur l'étude de la compression de l'algorithme RLE montrent qu'il y a encore fort à faire pour aboutir à une meilleure solution. On pourrait par exemple examiner la problématique de préfixer la taille du nombre de couples de sortie de l'algorithme par une étude statistique liée aux types de pixels que l'on manipule. Et du point de vu de son implémentation, l'examen de nouvelles solutions de décomposition de l'équation de mise à jour des symboles et de décompte du nombre d'occurences pourrait aboutir à une solution beaucoup plus efficace.

Notre travail sur la transformation homomorphe des autres blocs de la transformation spatiale d'une vidéo nous montre que, bien que nous ayons beaucoup d'opérations associées au schéma de Type I (binaire) et également une simplicité à décrire les circuits pour ce type, celui-ci reste tout de même beaucoup plus long lorsqu'on souhaite manipuler de grandes quantités d'informations comme une image ou une vidéo. Nous avons remarqué toutefois que le Type II avait énormément de potentiel en terme d'efficacité. Une proposition de recherche serait de regarder la possibilité de construire un pont entre les schémas de Type I et de Type II pour profiter des avantages qu'ils offrent. On pourrait aussi continuer à s'intéresser à ce problème majeur du chiffrement homomorphe qu'est le temps d'exécution des opérations de base. Des solutions intéressantes comme le schéma TFHE existent, mais restent encore à l'améliorer pour se rapprocher le plus possible des opérations qui sont réalisées dans le monde clair.

Pour les améliorations propres à la compression vidéo, on pourrait également rechercher d'autres types de systèmes de représentation des algorithmes comme la transformation RGB vers YCbCr, la DCT et la quantification qui seraient beaucoup plus adaptés aux systèmes de chiffrement homomorphe et donneraient de meilleurs qualitées d'image dans notre pipeline de compression. Notre travail s'est arreté à la compression d'une image bien qu'utilisant des solutions d'un compresseur vidéo. Une de nos perspectives serait alors de pouvoir étendre ce travail à un pipeline de compression vidéo en gérant de manière optimale les opérations de transformation temporelle. Nous pourrions également regarder les opérations de redimensionnement d'une image. En effet, un transcodeur étant un opérateur qui transforme un flux vidéo en un autre de caracté-

ristiques différentes, l'une des transformations qu'il applique souvent est l'interpolation des images qui constituent la vidéo transcodée. Se pencher sur la manière dont les algorithmes d'interpolation (bilinéaire, cubiques...) dans domaine clair peuvent être convertis vers le domaine chiffré serait ainsi un sujet fort d'intérêt pour la suite de nos travaux.

La solution que nous avons apportée au problème d'expansion des chiffrés vis-à-vis des messages clairs est prometteuse mais souffre d'un problème de perte d'efficacité (pour l'opération de multiplication) dû à la structure associée à notre représentation des nouveaux chiffrés. En effet, le fait de passer d'une structure polynômial à une structure vectorielle nous empêche de bénéficier des accélérateurs de calcul existant dans la première structure (le théorème des restes chinois CRT, la transformé de fourier des polynomes, etc), ce qui rend notre solution moins optimale qu'elle ne pourrait l'être. Une perspective serait de rechercher une autre représentation pour nos nouveaux chiffrés qui permettrait d'améliorer les opérations qu'il est possible de réaliser sur ceux-ci.

Liste des figures

Liste des tableaux

Bibliographie

[ABD16] Martin Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched ntru assumptions : Cryptanalysis of some fhe and graded encoding schemes. Cryptology ePrint Archive, Report 2016/127, 2016. https://eprint.iacr.org/2016/127. 16

[Alb17] Martin R. Albrecht. On dual lattices attacks against small-secret LWE and parameter choices in HELib and SEAL. In *Proceedings of EUROCRYPT*, pages 103–129, 2017. 83

[AMBGH11] Carlos Aguilar-Melchor, Slim Bettaieb, Philippe Gaborit, and Javier Herranz. Improving additive and multiplicative homomorphic encryption schemes based on worst-case hardness assumptions. *IACR Cryptology ePrint Archive*, 2011 :607, 2011. 12

[AMGH08] Carlos Aguilar-Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. Cryptology ePrint Archive, Report 2008/378, 2008. https://eprint.iacr.org/2008/378. 11

[AMGH10] Carlos Aguilar-Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with *d*-operand multiplications. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 138–154, 2010. 11, 12

[BC12] FM Bayer and RJ Cintra. Dct-like transform for image compression requires 14 additions only. *Electronics Letters*, 2012. 59

[Ben94] Josh Benaloh. Dense probabilistic encryption. May 1994. 10

[BFB+13] Madhukar B., Arild F., Gisle B., Vivienne S., and Mangesh S. Core transform design in the high efficiency video coding (hevc) standard. *IEEE Journal of Selected Topics in Signal Processing*, 2013. 59

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341, 2005. 11

[BGV12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012. 12, 16, 33, 34, 48, 52, 53, 66, 70, 78

[BLLN13a] J. W Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. pages 45–64. Springer, 2013. 66

[BLLN13b] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. Cryptology ePrint Archive, Report 2013/075, 2013. https://eprint.iacr.org/2013/075. 16

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. *IACR Cryptology ePrint Archive*, 2012 :78, 2012. 12

[BT95] Recommendation ITU-R BT. Studio encoding parameters of digital television for standard 4 : 3 and wide-screen 16 : 9 aspect ratios. 1995. 57

[BT13] Recommendation ITU-T BT. ITU-T H.265 : High efficiency video coding. 2013. 57

[BV11a] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. pages 97–106. IEEE Computer Society, 2011. 70

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *FOCS*, pages 97–106, 2011. 12

[BY87] E F Brickell and Y Yacobi. On privacy homomorphisms. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1987. 10, 48

[CAS17] S. Carpov, S. Aubry, and R. Sirdey. A multi-start heuristic for multiplicative depth minimization of boolean circuits. In *Proceedings of the 28th International Workshop on Combinatorial Algorithms*, 2017. 43

[CCF+16] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey. Stream ciphers : a practical solution for efficient homomorphic-ciphertext compression. In *Proceedings of the 23rd International Conference on Fast Software Encryption*, pages 313–333, 2016. 67, 83

[CCNS17] S Canard, S Carpov, D Nokam, and R Sirdey. Running compression algorithms in the encrypted domain : a case-study on the homomorphic execution of RLE. 2017. 6, 49, 54, 60

[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, 2015. 38

[CGGI16] I Chillotti, N Gama, M Georgieva, and M Izabachene. Faster fully homomorphic encryption : Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2016. 48, 53

[CGGI17a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Improving tfhe : faster packed homomorphic operations and efficient circuit bootstrapping. Cryptology ePrint Archive, Report 2017/430, 2017. https://eprint.iacr.org/2017/430. 3, 52

[CGGI17b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE : Fast Fully Homomorphic Encryption Library over the Torus. https://github.com/tfhe/tfhe, 2017. 13, 16, 48, 52, 71, 72

[CH18] H. Chen. and K. Han. Homomorphic lower digits removal and improved FHE bootstrapping. In *EUROCRYPT 2018*. Springer, 2018. 52

[CLP17a] H Chen, K Laine, and R Player. Simple encrypted arithmetic library (SEAL), 2017. 48, 52, 53

[CLP17b] H Chen, K Laine, and R Player. Simple encrypted arithmetic library-seal v2. 1. In *International Conference on Financial Cryptography and Data Security*. Springer, 2017. 55

[CMNT] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *CRYPTO 2011*. Springer. 55

[CS16a] S. Carpov and R. Sirdey. Another compression method for homomorphic ciphertexts. SCC '16, pages 44–50. ACM, 2016. 67

[CS16b] A. Costache and N. P. Smart. Which ring based somewhat homomorphic encryption scheme is best ? In *CT-RSA 2016*. Springer, 2016. 49

[CSVW16a] A Costache, N P Smart, S Vivek, and A Waller. Fixed-point arithmetic in she schemes. In *International Conference on Selected Areas in Cryptography*. Springer, 2016. 55

[CSVW16b] A. Costache, N.P. Smart, S. Vivek, and A. Waller. Fixed point arithmetic in SHE scheme. Technical Report 2016/250, Cryptology ePrint Archive, 2016. 3

[DGHV10] M. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT 2010*. Springer, 2010. 55

[DGK08] I. Damgard, M. Geisler, and Mikkel K. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 2008. 56

[DM14] Léo Ducas and Daniele Micciancio. Fhew : Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. https://eprint.iacr.org/2014/816. 13

[FK94] Michael Fellows and Neal Koblitz. Combinatorial cryptosystems galore ! *Contemporary Mathematics*, 168 :51–51, 1994. 11

[FSF+13a] S. Fau, R. Sirdey, C. Fontaine, C. Aguilar, and G. Gogniat. Towards practical program execution over fully homomorphic encryption schemes. In *Proceedings of the 8th IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 284–290, 2013. 14, 15

[FSF+13b] Simon Fau, Renaud Sirdey, Caroline Fontaine, Carlos Aguilar Melchor, and Guy Gogniat. Towards Practical Program Execution over Fully Homomorphic Encryption Schemes. In *3PGCIC*, pages 284–290, 2013. 33

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012 :144, 2012. 6, 13, 16, 33, 34, 38, 48, 52, 53, 66, 67, 68, 69, 71, 73, 75, 81, 82, 83

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. volume 196 of *LNCS*, pages 10–18. Springer, 1984. 10, 66

[Gen09a] C. Gentry. A fully homomorphic encryption scheme. phd, 2009. 48, 51

[Gen09b] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, 2009. 3, 11, 12, 66

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 107–109. IEEE, 2011. 12

[GH11b] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 129–148, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 12

[GHS12] C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. pages 465–482. Springer, 2012. 67

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM. 10

[GSV07] J. A. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *PKC 2007*. Springer, 2007. 54

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors : Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, 2013. 13, 16

[HKR17] C. Hao, L. Kim, and P. Rachel. Simple encrypted arithmetic library - seal v2.1. Technical Report 224, IACR ePrint Archive, 2017. 83

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In *CRYPTO*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571, 2014. 38

[HS15a] Shai Halevi and Victor Shoup. Bootstrapping for HElib. In *EUROCRYPT*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, 2015. 38

[HS15b] Shai Halevi and Victor Shoup. HElib An Implementation of homomorphic encryption. https://github.com/shaih/HElib, 2015. 48, 52, 53

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 575–594, 2007. 11

[LdVMPT09] Françoise Levy-dit Vehel, Maria Grazia Marinari, Ludovic Perret, and Carlo Traverso. A survey on polly cracker systems. In *Gröbner Bases, Coding, and Cryptography*, pages 285–305. Springer, 2009. 11

[LdVP04] Françoise Levy-dit Vehel and Ludovic Perret. A polly cracker system based on satisfiability. In *Coding, Cryptography and Combinatorics*, pages 177–192. Springer, 2004. 11

[Le06] Van-Ly Le. Polly two : A new algebraic polynomial-based public-key scheme. *Appl. Algebra Eng. Commun. Comput.*, 17(3-4) :267–283, 2006. 11

[LIS] CEA LIST. Cingulata : Compiler toolchain and RTE for running programs over encrypted data. https://github.com/CEA-LIST/Cingulata. 16, 48, 52, 53

[LLN15] Kristin E. Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. *IACR Cryptology ePrint Archive*, 2015 :133, 2015. 3

[LN14] T Lepoint and M Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. In *International Conference on Cryptology in Africa*. Springer, 2014. 49

[LP16] Kim Laine and Rachel Player. Simple encrypted arithmetic library-seal (v2. 0). Technical report, Technical report, September, 2016. 61

[LPR13] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. page 43, 2013. 13, 71

[LV09] M. Li and P. M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2009. 32, 37

[MHKK03] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. Low-complexity transform and quantization in h. 264/avc. *IEEE Transactions on circuits and systems for video technology*, 2003. 58

[NKCS18] Donald Nokam Kuate, Sebastien Canard, and Renaud Sirdey. Towards video compression in the encrypted domain : A case-study on the h264 and hevc macroblock processing pipeline. In Jan Camenisch and Panos Papadimitratos, editors, *Cryptology and Network Security*, pages 109–129, Cham, 2018. Springer International Publishing. 6

[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*, CCSW '11, pages 113–124, 2011. 67

[PAFXC16] M. Pierrick, J. Anthony, S. François-Xavier, and C. Claude. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *Proceedings of EUROCRYPT*, pages 311–343, 2016. 83

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. pages 223–238, 1999. 10, 66

[Pei09] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. pages 333–342. ACM, 2009. 71

[R.04] Iain E R. *H. 264 and MPEG-4 video compression : video coding for next-generation multimedia*. John Wiley & Sons, 2004. 59, 60

[RAD78] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. pages 169–178, 1978. 3, 10, 48, 66

[RAMS15] P. Rachel R. A. Martin and S. Sam. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9 :169–203, 2015. 83

[Reg05] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, 2005. 13, 71

[Ric10a] I. E. Richardson. *The H.264 advanced video compression standard*. Wiley, 2010. 20, 23

[Ric10b] I. E. Richardson. *The H.264 advanced video compression standard*. Wiley, 2010. 32

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. pages 120–126, 1978. 10, 66

[RVV13] S. S. Roy, F. Vercauteren, and I. Verbauwhede. High precision discrete gaussian sampling on fpgas. pages 383–401. Springer, 2013. 70

[SBS14] Vivienne Sze, Madhukar Budagavi, and Gary J Sullivan. High efficiency video coding (hevc). *Integrated Circuit and Systems, Algorithms and Architectures. Springer*, 39 :40, 2014. 20

[SCS15] Paul Dubrulle Sergiu Carpov and Renaud Sirdey. Armadillo : A compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*. ACM, 2015. 48

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2) :303–332, 1999. 2

[SSA+17] Kalpana Singh, Renaud Sirdey, François Artiguenave, David Cohen, and Sergiu Carpov. Towards confidentiality-strengthened personalized genomic medicine embedding homomorphic cryptography. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy, ICISSP 2017, Porto, Portugal, February 19-21, 2017.*, pages 325–333, 2017. 3

[Ste10] Rainer Steinwandt. A ciphertext-only attack on polly two. *Applicable Algebra in Engineering, Communication and Computing*, 21(2) :85–92, 2010. 11

[SV10] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, pages 420–443, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 12

[SV11] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *IACR Cryptology ePrint Archive*, 2011. 13, 67

[VDGHV10] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010. 12

[XCWF16] C Xu, J Chen, W Wu, and Y Feng. Homomorphically encrypted arithmetic operations over the integer ring. In *International Conference on Information Security Practice and Experience*. Springer, 2016. 53

[YGA+15] P. Yang, X. Gui, J. An, F. Tian, and J. Wang. An encrypted image editing scheme based on homomorphic encryption. In *Computer Communications Workshops*. IEEE, 2015. 49

[ZH13] P Zheng and J Huang. An efficient image homomorphic encryption scheme with small ciphertext expansion. In *Proceedings of the 21st ACM International Conference on Multimedia*. ACM, 2013. 49

université PARIS-SACLAY
ÉCOLE DOCTORALE
Physique et ingénierie : électrons, photons, sciences du vivant (EOBE)

Titre : Cryptographie homomorphe et transcodage d'image/vidéo dans le domaine chiffré

Mots clés : Cryptographie, Chiffrement homomorphe, Transcodage d'image/vidéo, Optimisation, gestion de slots

Résumé : L'apparition de nouvelles technologies comme l'informatique en nuages (cloud computing) offre de nouvelles opportunités de traitement de l'information. Par exemple, il est désormais facile de stocker ses photos ou vidéos personnelles sur des serveurs distants. Il est également possible de partager ces contenus à travers ces mêmes serveurs, ou encore via les réseaux sociaux ou les plateformes de téléchargement. Cependant, ces données personnelles sont bien souvent accessibles par le fournisseur de service, essentiellement pour des raisons pratiques : par exemple adapter une vidéo pour qu'elle s'affiche au bon format quel que soit l'appareil utilisé pour la visionner, permettre le partage de ses contenus avec d'autres personnes, etc. Cela soulève cependant un problème de confidentialité de ces données personnelles, et de confiance dans le fournisseur du service.

La cryptographie classique apporte des solutions à ce problème, mais soulève malheureusement celui de la maniabilité des données : il devient par exemple impossible d'adapter un contenu vidéo au bon format d'affichage puisque le fournisseur ne peut plus « voir » la vidéo.

Une solution alternative réside toutefois dans le chiffrement homomorphe. Cet outil un peu magique de la cryptographie avancée apporte la même sécurité que les algorithmes de cryptographie classique, mais permet de plus de manipuler les données tout en conservant leur forme chiffrée. Il offre ainsi une nouvelle perspective pour les fournisseurs puisque ceux-ci peuvent continuer à traiter l'information sans être capable de la voir, et donc sans atteinte à la vie privée de leurs utilisateurs, se conformant ainsi au nouveau Règlement Général sur la Protection des Données (RGPD).

Bien que le chiffrement homomorphe soit très souvent considéré comme insuffisamment mature, du fait de sa complexité algorithmique, cette thèse cherche à montrer son caractère prometteur, en s'intéressant à son usage pour le traitement d'images et de vidéos chiffrées à la source. Nous regardons ainsi les différents algorithmes qui constituent un encodeur d'image/vidéo (JPEG/H264 et HEVC) et les transformons en des circuits qui sont manipulables par des systèmes de chiffrement homomorphes. Nous proposons ainsi dans cette thèse le tout premier pipeline de compression d'images de type JPEG ("homomorphic-JPEG") sur des pixels qui sont chiffrés de bout-en-bout. Pour optimiser la gestion des données ainsi protégées, nous proposons également de nouveaux outils applicables à tous les schémas de chiffrement homomorphe sur les réseaux idéaux. Notre approche permet de maximiser le nombre de slots dans un chiffré et introduit de nouvelles fonctions pour manipuler ces différents slots de manière indépendante les uns des autres. Ces travaux de thèse ont abouti à la publication de deux articles dans des conférences internationales ainsi qu'à la soumission d'un article supplémentaire. .

Title : Homomorphic encryption and image/video transcoding in the encrypted domain

Keywords : Cryptography, Homomorphic encryption, Image/Video transcoding, Optimization, slot management

Abstract : The emergence of new technologies like cloud computing brings new opportunities in information processing. For example it is easy today to send our personal pictures or videos to a remote server. We can also share this content among the same servers or via social networks and streaming services. However, this personal data is often also available to the service provider, mainly for practical reasons e.g. to configure a video to have the right format regardless of the displayer (smartphone or computer), to share our data with other people, etc. This raises issues of privacy and trust into the service provider.

Classical cryptography brings some answers to this kind of issues, yet leaving the problem of handling the encrypted data : e.g., it becomes impossible to reconfigure a video because the provider can no longer "see" it.

An alternative solution is "homomorphic encryption". It is a powerful tool of advanced cryptography which provides the same security as classical cryptography algorithms, but it still allows us to manipulate ciphertexts such their underlying plaintexts are modified. Consequently, it offers a new perspective to service providers since they can continue to process their clients'information without knowing what it contains. This allows them to provide privacy-preserving services and comply with the new General Data Protection Regulation (GDPR).

Although it is considered that homomorphic encryption does not have enough maturity due to its large algorithmic complexity, in this thesis, we are trying to show its potential by using it in the context of image and video processing over the encrypted data. In this context, we look at the different algorithms in an image/video encoder (JPEG/H264 and HEVC) and transform them to circuits which can be manipulated by homomorphic encryption schemes. Our main contribution is to propose the first pipeline for an image compression of type JPEG (homomorphic-JPEG) running on end-to-end encrypted pixels. To optimize the management of the encrypted data, we also propose new tools applicable to existing homomorphic encryption schemes over the ring version of lattices. Our approach allows us to maximize the number of slots in some ciphertext and we introduce new functions allowing to handle these slots independently in the encrypted domain. This thesis work also lead to two publications to international conferences as well as the submission of an additional article .